Replication Server

Version 7

Built at 2024-04-17T21:35:15

1 Replication Server

EDB Postgres Replication Server (EPRS) replicates data between Postgres databases, or from non-Postgres databases to Postgres databases, for ongoing data synchronization as part of a migration.

2 Release notes

The Replication Server documentation describes the latest version including minor releases and patches. The release notes in this section provide information on what was new in each release. For new functionality introduced in a minor or patch release, there are also indicators within the content about what release introduced the feature.

Version	Release Date
7.7.0	14 Dec 2023
7.6.0	07 Sep 2023
7.5.1	26 May 2023
7.5.0	14 Feb 2023
7.4.0	29 Nov 2022
7.3.0	15 Nov 2022
7.2.1	25 Jul 2022
7.2.0	24 Jun 2022
7.1.0	21 Mar 2022
7.0.1	03 Mar 2022
7.0.0	01 Dec 2021

Supported upgrade paths

You can only upgrade to Replication Server 7.x from 6.2.15 or later 6.2.x point version. If you are running a version earlier than 6.2.15, you need to first upgrade to 6.2.15 or later 6.2.x point version before upgrading to 7.x.

> **Note**
>
> Version 7.x provides a non-breaking upgrade path for existing 6.2.x based cluster deployments; however, we strongly recommended that you verify the upgrade in a staging or nonproduction environment before applying the upgrade in a production environment. There is no downgrade path from version 7.x to version 6.2.x so it is essential to test the upgrade first before applying it to the production environment.

2.1 Replication Server 7.7.0 release notes

Released: 14 Dec 2023

New features, enhancements, bug fixes, and other changes in Replication Server 7.7.0 include the following:

Type	Description

Type	Description
Enhancement	Implemented CPU and memory usage related optimizations specifically applicable for publications with a large number of tables. [Support ticket #91588 #94343]
Enhancement	EDB Postgres Replication Server is now certified to use EDB Postgres Distributed (PGD) v5.3.0 as a publishing database for trigger mode and as a subscription database for both trigger and wal modes.
Enhancement	EDB Replication Server is now certified for use on Windows Server 2022.
Bug fix	Fixed memory issues observed for a multi-master setup with a fairly heavy load resulting in the Replication Server restart. [Support ticket #91588]
Bug fix	Fixed the null pointer exception observed for an edge case when a publication table is truncated and immediately removed from the publication. [Support ticket #96054]
Bug fix	Fixed an issue where the synchronize operation failed to replicate a table with a `BIGINT` value greater than integer range. [Support ticket #99264]
Bug fix	Fixed an issue where triggers manually configured by users on the target PostgreSQL or EDB Postgres Advanced Server databases are disabled during a data snapshot. [Support ticket #97637]

End-of-Support notice

We recommend you update to the most recent software version. *If you have not yet updated to the most current version, please see the end-of-support notes below*:

Software: Replication Server Version: 6.2 End of Standard Support: June 1, 2023

Additional details can be found at EDB Platform Compatibility.

> **Note**
>
> Version 7.x provides a non-breaking upgrade path for existing 6.2.x-based cluster deployments; however, we strongly recommend that the upgrade be verified in a staging or non-production environment before applying the upgrade in a production environment.

2.2 Replication Server 7.6.0 release notes

Released: 07 Sep 2023

New features, enhancements, bug fixes, and other changes in Replication Server 7.6.0 include the following:

Type	Description
Enhancement	EDB Replication Server now supports logging publication and subscription server logs in the English language, overriding the default locale, by way of the `logging.default.locale` configuration parameter. [Support ticket #89877]
Enhancement	The snapshot operation now uses the table-level parallel loading capability. This capability reduces overhead on the source database by using range-based criterion for loading each individual table data chunk instead of a fetch-offset approach. This optimization applies when the table primary key/unique constraint is based on a noncomposite numeric type attribute. [Support ticket # 93360]
Enhancement	To help investigate data synchronization gaps, Replication Server's logging now logs when rows are skipped due to filter criteria. [Support ticket #91296]
Bug fix	Fixed an issue where metadata from the primary controller database wasn't replicated when a SQL Server or an Oracle publication database is added as a standby controller database. [Support ticket #82050 and #91884]
Bug fix	Fixed the issues related to foreign key violations in the standby controller database that prevented upgrading from version 6.2.x to 7.x. [Support ticket #93129, #92056, and #91588]

Type	Description
Bug fix	Corrected a few code paths to release unused resources for timely garbage collection and optimized memory utilization. [Support ticket #91588]
Bug fix	Fixed a data validator Oracle edge case resulting in a `String index out of range` error for an Oracle-to-EDB Postgres Advanced Server validation.
Bug fix	Fixed an issue resulting in a synchronization failure for `nchar`, `nvarchar`, `xml`, and `sqlvariant` when using the `mssql-jdbc-10.2.1.jre8.jar` file for a SQL Server-to-EDB Postgres Advanced Server cluster setup.
Bug fix	Updated database type name references of “Postgres Plus Advanced Server” in the Replication Console and Replication CLI to “EDB Postgres Advanced Server.”
Bug fix	Fixed an issue that prevented logging of changed configuration parameters at publication and subscription server start or when the `reloadconf` command is executed.
Bug fix	Fixed a regression that led to an `Invalid custom column type mapping` error being observed for publication tables with no column mapping.

2.3 Replication Server 7.5.1 release notes

Released: 26 May 2023

New features, enhancements, bug fixes, and other changes in Replication Server 7.5.1 include the following:

Type	Description
Bug fix	Fixed an Oracle shadow table cleanup issue by adding the ability to set a threshold period via the `TX_MONITOR_PENDING_THRESHOLD_TIME` parameter. [Support ticket: #88095]
Bug fix	Fixed an issue that causes the removal of certain manually created views with a constraint dependency from the target subscription database on execution of re-snapshot. [Support ticket: #89491]
Bug fix	Fixed an issue in the upgrade routine that causes a `NumberFormatException` error while upgrading from version 6.2.18 to version 7.5.0. [Support ticket: #92056]
Bug fix	Fixed an issue that causes an `ArrayInexOutOfBoundsException` error and prevents publication server startup while upgrading from version 6.2.15 to version 7.5.0. [Support ticket: #91588]
Bug fix	Fixed a corner-case issue that results in data loss during replication from a SQL Server publication database when the transaction size exceeds the `txSetMaxSize` value. [Support ticket: #92797]
Bug fix	Fixed the packaging to add the ability to specify different `JAVA_HEAP` sizes for the Publication and Subscription servers. [Support ticket: #92552]
Bug fix	Fixed a Data Validator issue where `NUMERIC(n,0)` data type columns in Postgres were being represented as non-integer values when comparing the data in these columns to the corresponding `INT`, `BIGINT`, `SMALLINT`, or `BIT` data type columns in SQL Server, which are represented as integer values.
Bug fix	Fixed an issue that causes replicateDDL to fail when using the mssql-jdbc driver with SQL Server Publication database.
Bug fix	Increased the default `JAVA_HEAP` size for the publication server to reduce the possibility of `OutOfMemoryError` errors when default settings are used.
Bug fix	Fixed the issue where invalid values could be supplied when specifying a column mapping. A validation of the specified column name is now performed and throws an error if the value is invalid.

2.4 Replication Server 7.5.0 release notes

Released: 14 Feb 2023

New features, enhancements, bug fixes, and other changes in Replication Server 7.5.0 include the following:

Type	Description
Enhancement	EDB Replication Server now supports EDB Postgres Advanced Server version 15 as a source publication and/or target subscription database in trigger and WAL based replication cluster configurations.
Enhancement	EDB Replication Server has been enhanced to support snapshot replication of materialized views from Oracle to EDB Postgres Advanced Server. [Support ticket #86033]
Bug fix	Fixed an issue where the synchronize replication fails from an EDB Postgres Advanced Server publication database to an Oracle subscription database when another Oracle database is registered as a publication database. [Support ticket #89152]
Bug fix	Fixed a corner case issue where snapshot replication fails with a halt state when table-level parallel loading is enabled with the `snapshotParallelTableLoaderLimit` option. [Support ticket #87586]
Bug fix	Fixed an issue where the replication of a foreign key constraint is skipped in an MMR cluster when the FK is based on a unique index constraint in a source PostgreSQL/EDB Postgres Advanced Server publication database. [Support ticket #89491]
Bug fix	Fixed a regression issue introduced in EDB Replication Server 7.x that might result in synchronize replication failure when a PostgreSQL/EDB Postgres Advanced Server publication table contains a geometric type column.

2.5 Replication Server 7.4.0 release notes

Released: 29 Nov 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.4.0 include the following:

Type	Description
Enhancement	EDB Replication Server is now certified to work with PostgreSQL version 15, which is the newest major version of PostgreSQL. PostgreSQL 15 is now supported both as a source Publication and/or target Subscription database in trigger and WAL based replication cluster configurations.
Bug fix	Fixed an issue where a redundant duplicate copy of `commons-lang3-3.12.0.jar` is placed as part of the EDB Replication Server installation on Windows OS.
Other	Removed support of PostgreSQL 10 and EDB Postgres Advanced Server (EPAS) 10 for use with Replication Server. These versions of PostgreSQL and EPAS have reached end of life and are no longer supported under the normal support agreements.
Other	Removed support of Oracle 10g for use with Replication Server. This version had previously been tested and certified for use with Replication Server. Although Oracle 10g may continue to work with Replication Server, EDB is no longer performing ongoing verification of it.

2.6 Replication Server 7.3.0 release notes

Released: 15 Nov 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.3.0 include the following:

Type	Description
Enhancement	Replication Server now provides the ability to define custom column data type mappings for the replication of tables between Oracle and EDB Postgres Advanced Server or PostgreSQL. The data types in the source and target databases must be of compatible types.
Enhancement	Replication Server is now certified to work with the Microsoft provided SQL Server JDBC driver versions 10.2 and 11.2. Although the open source jTDS driver can still be used for connections to Microsoft SQL Server, it is recommended that the Microsoft provided SQL Server JDBC driver be used instead, as it provides support for the more recent versions of SQL Server.
Enhancement	Replication Server is now certified to work on the Debian 11 platform.

Type	Description
Enhancement	The synchronize process has been enhanced to optimize the `replicateDDL` operation so that it is able to complete within a reasonable time window during peak activity. [Support ticket #84057]
Enhancement	Removed the limitation of being able to add only 1000 tables to a single Oracle publication.
Security fix	Updated the version of the zlib library provided with the Replication Server Windows installer to version 1.2.13 to address a security vulnerability identified in CVE-2022-37434. This vulnerability is not present in the Linux distributions of Replication Server.
Bug fix	Fixed an issue that results in a publication creation failure for trigger-based replication mode if two or more tables have a common name pattern. [Support ticket #83414]
Bug fix	Fixed an issue encountered in an MMR cluster where the update-update conflict resolution might not discard some of the modified column values on the database node whose values are marked to be discarded. [Support ticket #83594]
Bug fix	Fixed an issue that causes the synchronize operation to fail when an Oracle Publication table primary key is based on the NCHAR/NVARCHAR type and the table contains a LOB column.
Bug fix	Fixed an issue encountered in a trigger-based MMR cluster where the history cleanup job is not configured to fire automatically for non-MDN nodes.
Bug fix	Fixed an issue where the synchronize operation fails if the `standard_conforming_strings` is off and the source Oracle column contains a `\'` character combination.
Bug fix	Fixed an issue where the `reloadconf` Replication Server CLI command reports success even when the configuration file is not found.

2.7 Replication Server 7.2.1 release notes

Released: 25 Jul 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.2.1 include the following:

Type	Description
Bug fix	Fixed the issue that caused the Publication creation failure due to a shadow table name conflict. [Support Ticket #83414]
Bug fix	Fixed the issue where the Publication database registration fails for a database that has intarray extension installed on it. [Support Ticket #83759]

2.8 Replication Server 7.2.0 release notes

Released: 24 Jun 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.2.0 include the following:

Type	Description
Bug fix	Fixed the issue where Synchronize replication fails when replicating a CLOB/BLOB column from Oracle to PostgreSQL/EPAS.
Bug fix	Fixed the issue where Synchronize replication fails for a table that contains a CLOB/BLOB column and the primary key is based on a CHAR/VARCHAR column.
Bug fix	Fixed the issue that caused failure of the Create Subscription operation when Oracle is the Publication database and EPAS is set as the Control database.
Bug fix	Fixed an issue that resulted in redundant `INSERT-INSERT` conflicts in a corner case for a MMR cluster.
Bug fix	Fixed the issue of snapshot failure while migrating constraints when the intarray extension is installed.
Bug fix	Fixed the exception seen during snapshot for an Oracle to EPAS cluster configuration.

Type	Description
Bug fix	Fixed the issue that caused a Create Publication operation to fail when the table name contains dot characters.
Bug fix	Fixed the issue where the Create Publication operation with a row-level table Filter fails for money, bit, and “bit varying” data types.
Bug fix	Fixed an issue where the Replication Console hangs when the Conflict Details view is displayed.
Bug fix	Fixed the issue of a Synchronization failure when the user added a table with a case sensitive schema name for an EPAS to SQL Server cluster configuration.
Bug fix	Fixed an issue that caused the Remove Publication operation to halt after a version 7.1.0 upgrade was applied in an Oracle to EPAS cluster configuration.
Bug fix	Fixed the issue where a control function is skipped from removal after the Publication database is unregistered.
Documentation	Updated the documentation to correct certain steps related to the configuration of a SSL certificate.

2.9 Replication Server 7.1.0 release notes

Released: 21 Mar 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.1.0 include the following:

Type	Description
Enhancement	Replication Server is now certified to support SQL Server 2016, 2017, and 2019 as the publication and subscription database.
Enhancement	The `validatepubs` and `validatepub` CLI commands no longer require the `-repgrouptype` option.
Enhancement	The `use-ora-case` option has been added to the Data Validator tool to enforce usage of the same case for the tables specified using the `include-tables` option, irrespective of the default case in the source and target databases. See Performing data validation for more information.
Security Fix	The log4j component version bundled with Replication Server in versions before 7.1.0 has been identified as being affected by CVE-2019-17571. The log4j version has been upgraded to the latest patched version 2.17.0 to address this vulnerability.
Bug fix	Fixed issue so that the replication history is loaded properly in the Replication console for SMR + MMR cluster at publication service startup [Support Ticket #74217].
Bug fix	Fixed the issue that caused a failure when loading the `pgShadowTableQueryMode` configuration parameter [Support Ticket #78117].
Bug fix	Fixed the publication server crash observed in a corner case on Windows when the Sync operation is performed for a WAL-based cluster [Support Ticket #79555].
Bug fix	Fixed the issue that prevented adding PostgreSQL as a publication database when the controller database is EDB Postgres Advanced Server.
Bug fix	Fixed the issue where an update-update conflict with a custom resolution strategy failed to resolve with EDB Postgres Advanced Server/PostgreSQL JDBC driver version 42.3.2 or later.
Bug fix	Fixed the issue where the incorrect publication association is shown for a subscription.
Bug fix	Report an error if the table name is not schema-qualified for the `createpub` CLI command.
Bug fix	Fixed an issue that prevented removal of a publication when a row filter is defined against it.
Bug fix	Fixed the snapshot failure for a SMR cluster with EDB Postgres Advanced Server publication database (created with the `no-redwood-compat` option) and PostgreSQL subscription database.

2.10 Replication Server 7.0.1 release notes

Released: 03 Mar 2022

New features, enhancements, bug fixes, and other changes in Replication Server 7.0.1 include the following:

Type	Description
Security fix	Addresses JDBC vulnerability CVE-2022-21724 that has a CVSS score of 8.5, which is a High severity. The PostgreSQL JDBC driver version 42.3.2 that contains the security fix is bundled with Replication Server. If you have a RPM- or Debian-based Replication Server installation, the EDB JDBC driver for Advanced Server, which incorporates the community version 42.3.2, needs to be updated. For example, on RHEL 7 or CentOS 7 distributions, run: `yum update edb-jdbc` .
Bug fix	Fixed the issue where database registration fails with the error `libpq JNI wrapper library is not available` for WAL-based CDC on Windows.

2.11 Replication Server 7.0.0 release notes

Released: 01 Dec 2021

> **Note**
>
> The 7.0 version is backward compatible with 6.2.x and provides a non-breaking upgrade path for the existing cluster deployments. However, it is strongly recommended to verify it in a staging or nonproduction environment before applying the upgrade in a production environment.

New features, enhancements, bug fixes, and other changes in Replication Server 7.0 include the following:

Type	Description
New feature	The Publication and Subscription Server configuration options can be modified and applied without requiring a server restart. The CLI now exposes a new command `reloadconf` to reload the updated configuration values at runtime.
New feature	Replication Server is enhanced to support replication from EDB Postgres Advanced Server (non-redwood mode) and PostgreSQL to Oracle.
Enhancement	Replication Server is certified to support PostgreSQL v14 and EDB Postgres Advanced Server v14.
Enhancement	Table-level parallel data loading significantly improves initial Snapshot data replication time for large size Publication tables. The benchmarks reveal 3X to 30X performance improvement for a given workload and related options.
Enhancement	Row-level Filtering capability is enhanced to allow inclusion/exclusion of data based on a sub-query [1284035].
Enhancement	The CLI command `addtablesintopub` is enhanced to expose an option to enable row-level filter for the target Subscription (SMR)/Master (MMR) nodes [1321993].
Enhancement	The CLI commands `createpub` and `addtablesintopub` now support a new option to allow the users to add all the tables to a Publication without requiring individual table identification [1273701].
Enhancement	The CLI commands `createpub` and `addtablesintopub` are enhanced to allow the user specify a single custom conflict resolution option applicable for multiple tables [731271].
Enhancement	Table(s) can be added into an existing Publication by skipping schema replication [926058].
Enhancement	The Replication GUI displays additional replication stats on the Real-time Monitor tab to monitor the status of the cluster [795247].
Enhancement	Replication of the HASH partition table is now supported from Oracle to EDB Postgres Advanced Server.
Enhancement	Incremental replication of CLOB, BLOB, and XMLType is now supported from Oracle to EDB Postgres Advanced Server.
Enhancement	Support added for replication of JSON data type from PostgreSQL and EDB Postgres Advanced Server to Oracle.
Bug fix	Fixed an issue where replication from EDB Postgres Advanced Server to SQL Server fails for UUID data type [73835].
Bug fix	The CLI reports an error on Publication removal if one or more Subscriptions are associated with the Publication [72194].
Big fix	Fixed an issue where replication history cleanup job fails when Oracle is the active Controller database [72626].
Big fix	Fixed an issue where the filter clause fails when applied for a table list with more than 10 tables.
Bug fix	Fixed an issue where the add filter command picked the incorrect table index for the filter clause in the WAL mode replication.
Bug fix	In a hybrid replication cluster, the cleanup shadow table routine is subject to remove unprocessed changes.
Bug fix	Fixed an issue where synchronize failed for an SMR cluster during replication from PostgreSQL to Oracle 12.1 when using the JSON datatype.
Bug fix	Fixed an issue where the initial Snapshot fails in a WAL-based MMR cluster while adding the third database.

Type	Description
Bug fix	Fixed an issue where running `yum install ppas-xdb` skips ppas-xdb-libs dependency.
Bug fix	Fixed an issue where the Synchronize operation failed to replicate NULL for a BLOB type for EPAS to Oracle.
Bug fix	Fixed an issue where the Synchronize operation fails for one of the target Subscriptions for EPAS to Oracle replication cluster.

3 Introduction

Replication Server is an asynchronous replication system available for PostgreSQL and for EDB Postgres Advanced Server.

You can use Replication Server to implement replication systems based on either of two different replication models: single-master (primary-to-secondary) replication or multi-master replication. Regardless of the replication model, Replication Server is flexible and easy to use.

For single-master replication, PostgreSQL, EDB Postgres Advanced Server, Oracle, and Microsoft SQL Server are supported in an assortment of configurations (including cascading replication), allowing organizations to use it in multiple use cases with a variety of benefits.

The following are some combinations of cross database replications that Replication Server supports for single-master replication:

- From Oracle to PostgreSQL
- From Oracle to EDB Postgres Advanced Server
- From SQL Server to PostgreSQL
- From SQL Server to EDB Postgres Advanced Server
- From EDB Postgres Advanced Server to Oracle
- From PostgreSQL to SQL Server
- From EDB Postgres Advanced Server to SQL Server
- Between PostgreSQL and EDB Postgres Advanced Server
- From PostgreSQL to Oracle (WAL mode)
- From PostgreSQL to Oracle (trigger mode)

> **Note**
>
> Oracle Real Application Clusters (RAC) and Oracle Exadata aren't supported by Replication Server. These Oracle products aren't evaluated for or certified with Replication Server.

For multi-master replication, Replication Server supports the following configurations:

- Between PostgreSQL database servers
- Between PostgreSQL database servers and EDB Postgres Advanced Servers in PostgreSQL-compatible mode
- Between EDB Postgres Advanced Servers in PostgreSQL compatible mode
- Between EDB Postgres Advanced Servers in Oracle compatible mode

> **Note**
>
> You need basic SQL knowledge and basic Oracle, SQL Server, or PostgreSQL database administration skills (whichever are applicable) to create databases, users, schemas, and tables and assign database object privileges.

3.1 Syntax conventions

Replication Server works with Linux and Windows systems. However, we show directory paths in the Linux format with forward slashes. When working on Windows systems, start the directory path with the drive letter followed by a colon, and substitute back slashes for forward slashes.

Much of this documentation about Replication Server applies to PostgreSQL and EDB Postgres Advanced Server. We use the term Postgres to refer to both PostgreSQL and EDB Postgres Advanced Server. Only when a distinction needs to be made are the full names used.

The installation directory path of the PostgreSQL or EDB Postgres Advanced Server products is referred to as `POSTGRES_INSTALL_HOME`.

- For PostgreSQL Linux installations, this path defaults to `/opt/PostgreSQL/x.x` for version 10 and earlier. For later versions, use the PostgreSQL community packages.
- For PostgreSQL Windows installations, this path defaults to `C:\Program Files\PostgreSQL\x.x`. - For EDB Postgres Advanced Server Linux installations performed using the interactive installer for version 10 and earlier, this path defaults to `/opt/PostgresPlus/x.xAS or /opt/edb/asx.x`.
- For EDB Postgres Advanced Server Linux installations performed using an RPM package, this path defaults to `/usr/ppas-x.x or /usr/edb/asx.x`.
- For EDB Postgres Advanced Server Windows installations, this path defaults to `C:\Program Files\PostgresPlus\x.xAS` or `C:\Program Files\edb\asx.x`. The product version number is represented by `x.x` or by `xx` for version 10 and later.

3.2 Certified and supported product versions

You can use the following database product versions with Replication Server:

- PostgreSQL versions 12, 13, 14, 15, and 16
- EDB Postgres Advanced Server versions 12, 13, 14, 15, and 16
- Oracle 11g Release 2 version 11.2.0.2.0 is explicitly certified. Newer minor versions in the 11.2 line are supported as well.
- Oracle 12c version 12.1.0.2.0 is explicitly certified. Newer minor versions in the 12.1 line are supported as well.
- Oracle 18c version 18.1.0.2.0 is explicitly certified. Newer minor versions in the 18.1 line are supported as well.
- Oracle 19c version 19.1.0.2.0 is explicitly certified. Newer minor versions in the 19.1 line are supported as well.
- SQL Server 2014 version 12.0.5000.0 is explicitly certified. Newer minor versions in the 12.0 line are supported as well.

> **Note**
>
> All PostgreSQL and EDB Postgres Advanced Server versions available as BigAnimal single-node and primary/standby high-availability cluster types are also supported for SMR configurations. See the BigAnimal (EDB's managed database cloud service) documentation for more information about BigAnimal's supported cluster types. See the database version policy documentation for the versions of PostgreSQL and EDB Postgres Advanced Server available in BigAnimal.
>
> EDB Postgres Distributed (PGD) v5.3.0 is explicitly certified as a Publishing database for trigger mode and as Subscription database for both trigger and wal modes.

As of Replication Server 7.1.0:

- SQL Server 2016 version 13.00.5026 is explicitly certified. Newer minor versions in the 13.0 line are supported as well.
- SQL Server 2017 version 14.0.1000.169 is explicitly certified. Newer minor versions in the 14.0 line are supported as well.
- SQL Server 2019 version 15.0.2000.5 is explicitly certified. Newer minor versions in the 15.0 line are supported as well.

Contact your EnterpriseDB Account Manager or sales@enterprisedb.com if you require support for other platforms.

> **Note**
>
> Replication server isn't tested and isn't officially supported for use with Oracle RAC and Exadata, but it might work when connected to a single persistent node. To determine its ability to work with RAC or Exadata, contact your EDB representative.

3.3 Permitted configurations and combinations

Depending on the database products you're using with Replication Server (Oracle, SQL Server, PostgreSQL, or EDB Postgres Advanced Server), along with the compatibility configuration mode if you're using EDB Postgres Advanced Server, certain combinations of a source database server and a target database server aren't permitted for a publication and its associated subscription in a single-master replication system.

Similarly, you can use only certain combinations of database products and EDB Postgres Advanced Server compatibility configuration modes together in a multi-master replication system.

For a single-master replication system, the source refers to the database server of the publication database. The target refers to the database server of the subscription database.

For a multi-master replication system, all of the participating database servers act as both a source and a target for all other participating database servers, so the restrictions pertain to the combinations of database servers and compatibility configuration modes that can be used together in the same multi-master replication system.

> **Note**
>
> A publication or subscription database can't be shared across SMR and MMR clusters. For example, if `inventory` is registered as a subscription database in an SMR cluster, you can't register it as a publication database in the MMR cluster. While Replication Server might not report an error, such a type of deployment architecture isn't supported.

EDB Postgres Advanced Server compatibility configuration modes

EDB Postgres Advanced Server supports two compatibility configuration modes of operation:

- Oracle-compatible configuration mode. Operations are performed using Oracle syntax and semantics for data types, functions, database object creation, and so forth. This mode is useful when your applications are migrated from Oracle or you want your applications built in an Oracle-compatible fashion.
- PostgreSQL-compatible configuration mode. Operations are performed using native PostgreSQL syntax and semantics. This mode is useful when your applications are migrated from PostgreSQL or you want your applications built in a PostgreSQL-compatible fashion.

For more information on features supported in Oracle compatible configuration mode, see Database compatibility for Oracle developers.

Select the compatibility configuration mode when you install EDB Postgees Advanced Server.

Permitted SMR source and target configurations

The following table shows the combinations of source and target database server products and EDB Postgres Advanced Server compatibility configuration modes permitted by Replication Server for single-master replication systems. The left-hand column lists the possible source database server products, including the possible EDB Postgres Advanced Server compatibility configuration modes. The top row lists the same set of possible target database server products and EDB Postgres Advanced Server compatibility configuration modes.

`Yes` at the intersection of a source and target indicates that Replication Server permits replication using that combination of database server configurations for a publication and its associated subscription. `No` indicates replication isn't permitted for that combination.

Source/target	Oracle	Microsoft SQL Server	PostgreSQL	EDB Postgres Advanced Server (Oracle compatible)	EDB Postgres Advanced Server (PostgreSQL compatible)
Oracle	No	No	Yes	Yes	Yes
Microsoft SQL Server	No	No	Yes	Yes	Yes
PostgreSQL	Yes	Yes	Yes	Yes	Yes
EDB Postgres Advanced Server (Oracle compatible)	Yes	Yes	No	Yes	No
EDB Postgres Advanced Server (PostgreSQL compatible)	Yes	Yes	Yes	Yes	Yes

Permitted MMR database server configurations

For multi-master replication systems, each primary node acts as both a source for all primary nodes and a target for all primary nodes. Thus, the permitted database servers comprising a particular multi-master replication system or cluster is determined by the overall composition of the cluster, which is initially established when selecting the database type of the primary definition node (see Step 3 in Adding the primary definition node).

The two basic cluster types can be characterized as follows:

- PostgreSQL-compatible cluster. All primary nodes must consist of PostgreSQL database servers or EDB Postgres Advanced Servers installed in PostgreSQL-compatible configuration mode.
- EDB Postgres Advanced Server Oracle-compatible cluster. All primary nodes must consist of EDB Postgres Advanced Servers installed in Oracle-compatible configuration mode.

The following table summarizes the permitted database server configurations allowed in the two cluster types. The left-hand column lists the possible database server products, including the possible EDB Postgres Advanced Server compatibility configuration modes. The top row lists the supported cluster types.

`Yes` at the intersection of a database server and cluster type indicates that Replication Server permits the database server and the specified configuration mode in the cluster type. `No` indicates the database server and the specified configuration mode can't be used in the cluster type.

Database server/cluster type	PostgreSQL-compatible cluster	EDB Postgres Advanced Server Oracle-compatible cluster
PostgreSQL	Yes	No
EDB Postgres Advanced Server (PostgreSQL compatible)	Yes	No
EDB Postgres Advanced Server (Oracle compatible)	No	Yes

4 Overview

To design a replication system, you need to know basic replication terms and concepts and understand the components and architecture of Replication Server. Design guidelines and directions for implementing a replication system using Replication Server provide you with tools you need to accomplish your task.

4.1 Why use replication?

You can use data replication in a variety of use cases in organizations where it's important to use the same data in multiple settings. This replication of data allows users to work with real data that yields real results that are reliable in more than one setting. Support of both single-master and multi-master replication gives Replication Server a broad range of supported use cases.

Offloading reporting and business intelligence queries

In this use case, users take all or a subset of data from a production OLTP system and replicate it to another database whose sole purpose is to support reporting queries. This approach can have multiple benefits:

- Reporting loads are removed from the OLTP system, improving transaction processing performance.
- Query performance improves without being subordinated to transactions on the system.
- In Oracle installations, the reporting server duties can be handled by a product like EDB Postgres Advanced Server, reducing licensing costs for a reporting server.

Using warm standby servers

When organizations want to improve the availability of their data, one cost-effective solution is to use warm standby servers. These are database servers kept up to date with the online system through replication. You can bring these servers online quickly when a failure occurs in the production system. You can also use warm standby servers for regular maintenance by switching over to the standby server so that you can take the production server offline for regular maintenance.

Testing systems in parallel

Upgrading or moving to a new database system sometimes requires that the old and new systems be up and running in parallel to allow for testing and comparing results in real time. You can use replication in this use case. It's frequently used in development and testing environments.

Migrating data

Similar to running in parallel is the situation in which you migrate data from one system to another in a sort of*seeding* operation. Replication can be very effective in this situation by quickly copying data.

Write availability

In single-master replication, only the primary database is available for writes. The secondary databases are read-only for applications. If the replicated target databases must be available for write access as well, you can use multi-master replication for the same use cases as single-master replication but with the additional advantage of write access to the secondary.

Write scalability

In write-intensive applications, multi-master replication allows you to use multiple database servers on separate hosts to process write transactions independently of each other on their own primary databases. You can then reconcile changes across primary databases on your own schedule.

Localized data access

In a geographically dispersed application, you can provide local access to the database to regions of clients. Having the database servers physically close to clients can reduce latency with the database. Multi-master replication allows you to use a WAN-connected network of primary databases that can be geographically close to groups of clients yet maintain data consistency across primary databases.

4.2 Replication concepts and definitions

Replication Server is a software product that lets you implement a *replication system*. A replication system is software and hardware whose purpose is to make a copy of data from one location to another and to ensure the copied data is the same as the original over time.

Replication Server applies the replication system concept to tables of Oracle, SQL Server, PostgreSQL, and EDB Postgres Advanced Server database management systems.

4.2.1 Comparison of single-master and multi-master replication

Replication Server supports are two models of replication systems:

- **Single-master replication (SMR).** Changes (inserts, updates, and deletions) to table rows are allowed to occur in a designated primary database. These changes are replicated to tables in one or more secondary databases. The replicated tables in the secondary database aren't permitted to accept any changes except from its designated primary database. (This is also known as primary-to-secondary replication.)
- **Multi-master replication (MMR).** Two or more databases are designated in which tables with the same table definitions and initial row sets are created. Changes (inserts, updates, and deletions) to table rows are allowed in any database. Changes to table rows in a database are replicated to their counterpart tables in every other database.

For a single-master replication system, a variety of configurations are supported including:

- Replication between PostgreSQL and EDB Postgres Advanced Server databases (between products in either direction)
- Replication in either direction between Oracle and EDB Postgres Advanced Server
- Replication in either direction between SQL Server and PostgreSQL
- Replication in either direction between SQL Server and EDB Postgres Advanced Server
- Replication in either direction between PostgreSQL and Oracle

For multi-master replication, the participating database servers in a given multi-master replication system must be of the same type:

- PostgreSQL database servers
- PostgreSQL database servers and EDB Postgres Advanced Servers operating in PostgreSQL compatible mode
- EDB Postgres Advanced Servers operating in PostgreSQL compatible mode
- EDB Postgres Advanced Servers operating in Oracle compatible mode

> **Note**
>
> A given database can't simultaneously participate in both a single-master replication system and a multi-master replication system.

4.2.2 Publications and subscriptions

Replication Server uses an architecture called *publish* and *subscribe*. The data made available for copying by a replication system is defined as a publication. To get a copy of that data, you must "subscribe" to that publication. The manner in which you subscribe is slightly different for single-master and multi-master replication systems.

In Replication Server, a publication is defined as a named set of tables and views in a database. The database that contains the publication is called the *publication database* of that publication.

In a single-master replication system, to get a copy of a Replication Server publication, you must create a *subscription*. A Replication Server subscription is a named association of a publication to a database to which to copy the publication. This database is called the *subscription database*.

Similar to a single-master replication system, when creating a multi-master replication system, you first define a publication in the publication database. You then add one or more databases that you want to participate in this multi-master replication system. As you add each database, it is associated with this replication system. You don't create an explicit, named subscription in a multi-master replication system.

In a single-master replication system, replication occurs when Replication Server starts and completes either of the following processes:

- Applies changes that were made to rows in the publication since the last replication occurred to rows in tables of the subscription database (called synchronization).
- Copies rows of the publication to empty tables of the subscription database (called a snapshot). See Snapshot and synchronization overview.

The subscription tables are the tables in the subscription database created from corresponding tables or views in the publication.

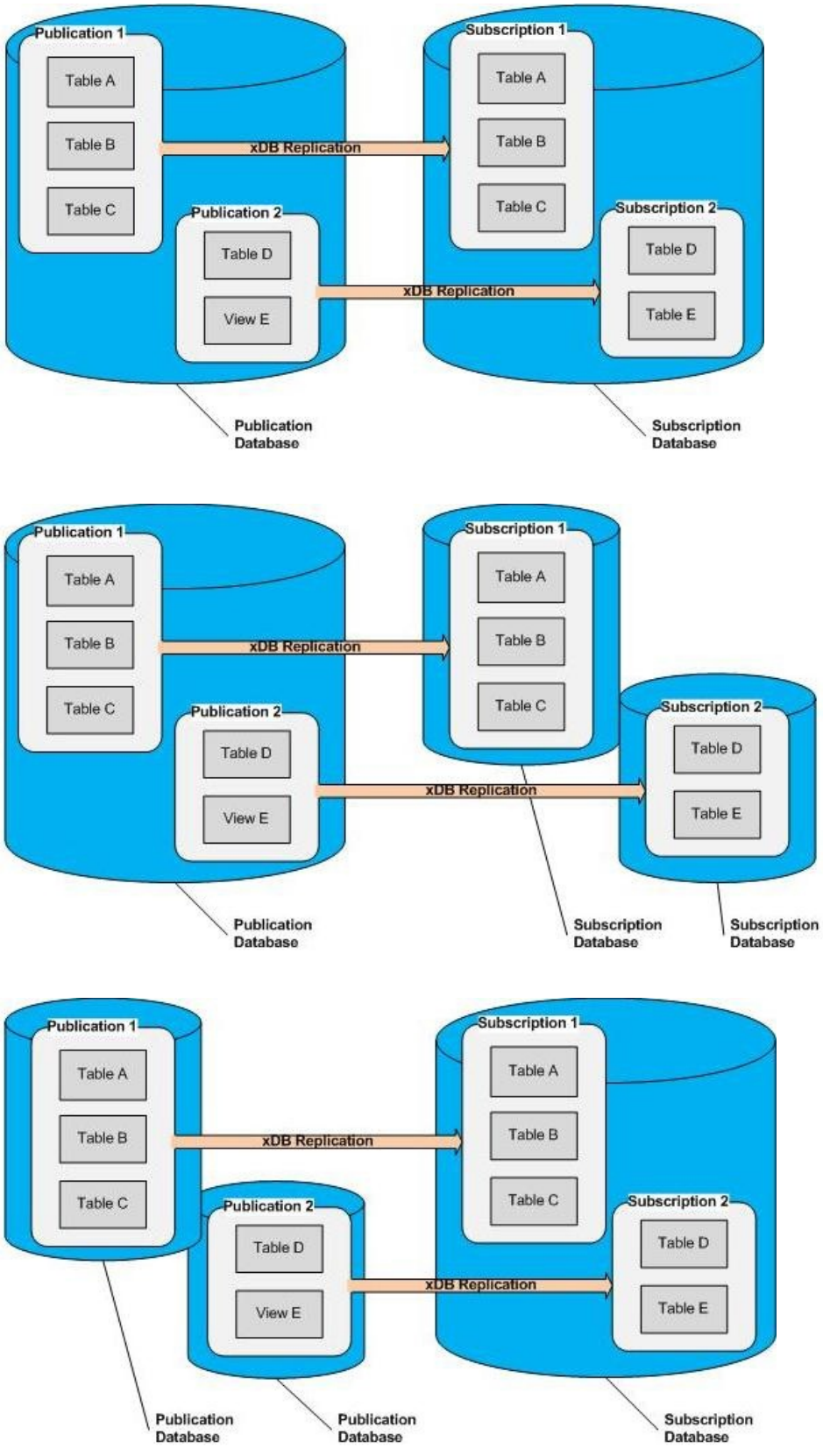
> **Note**

> In a single-master replication system, Replication Server creates a table in the subscription database for each view contained in the publication.

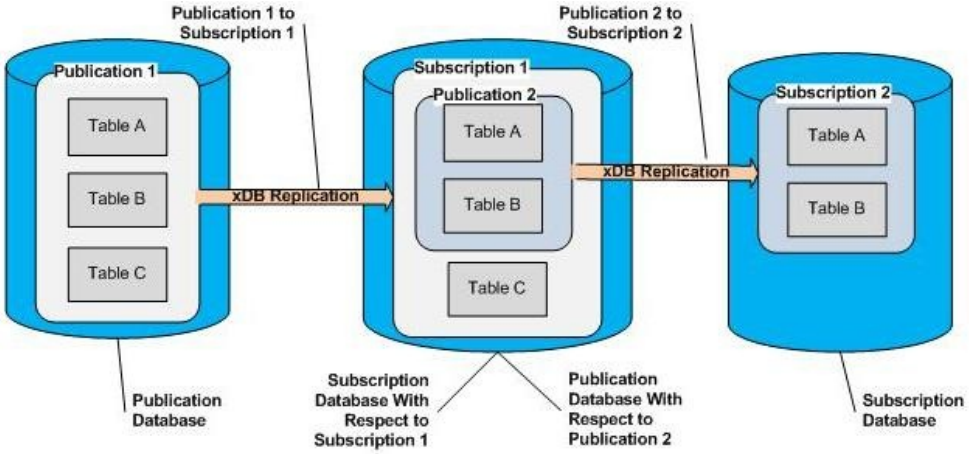
In a multi-master replication system, the concept and definition of replication is nearly identical to a single-master replication system with the following modifications:

- Synchronization can occur between any pair of databases (referred to as primary nodes) participating in the replication system.
- A snapshot can occur from the publication database designated as the primary definition node to any of the other primary nodes.

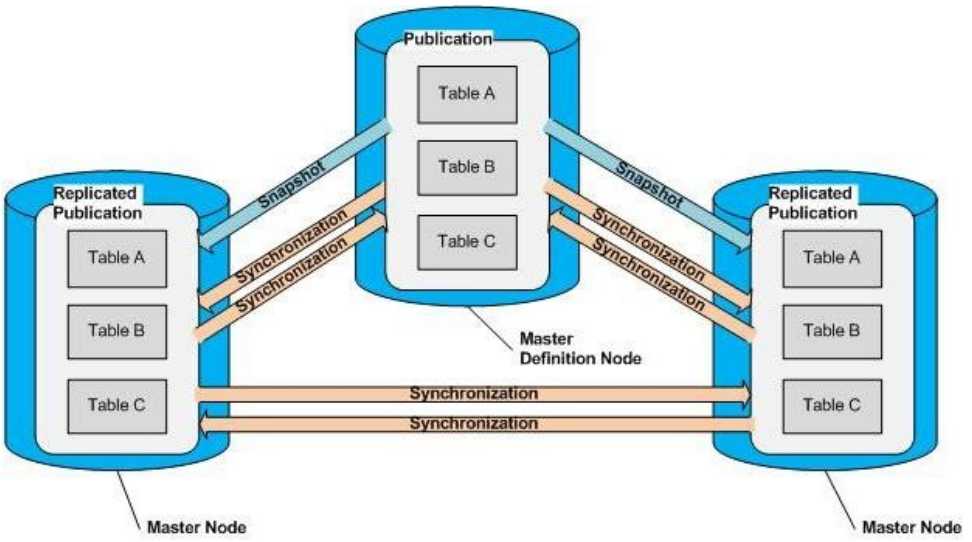
The following diagram shows a multi-master replication system with three primary nodes.



This diagram shows that a table that was created as a member of a subscription can be used in a publication replicating to another subscription. This scenario is called *cascading replication*.

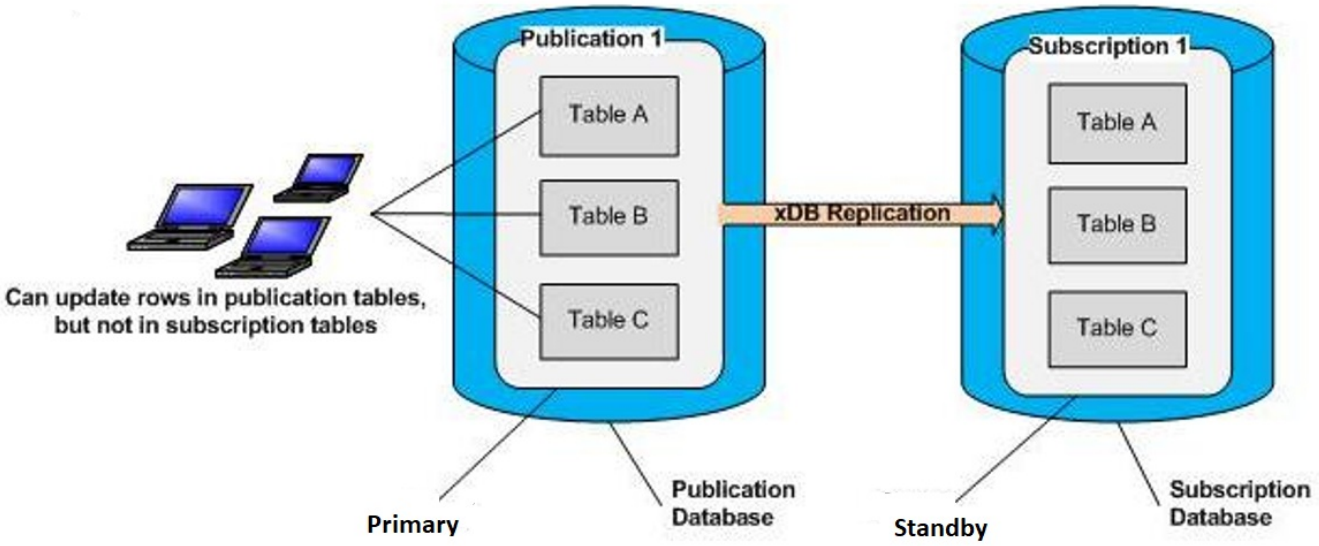


The following diagram shows a multi-master replication system with three primary nodes.



4.2.3 Single-master (primary-to-secondary) replication

Replication Server performs primary-to-secondary replication when a single-master replication system is implemented. The publication is the master and the subscription is the secondary. In a primary-to-secondary relationship, changes are propagated in one direction only, from the master to the secondary.



Generally, don't make changes to the definitions of the publication tables or the subscription tables. If such changes are made to the publication tables, they aren't propagated to the subscription and vice versa unless the DDL change replication feature is used as described in Replicating DDL changes. If changes are made to the table definitions without using the DDL change replication feature, there's a risk that future replication attempts will fail.

Don't make changes to the rows of the subscription tables. If such changes are made, they aren't propagated back to the publication. If you make changes to the subscription table rows, it's likely that the rows will no longer match their publication counterparts. There's also a risk that future replication attempts will fail.

4.2.4 Multi-master replication

As an alternative to the single-master (primary-to-secondary) replication model, Replication Server supports multi-master replication.

The following definitions are used when referring to multi-master replication systems.

A primary node is a database participating in a multi-master replication system.

The database (primary node) in which the publication is initially defined is specially designated as the *primary definition node (MDN)*. There can be only one primary definition node at a time. However, you can change which primary node is the primary definition node. When it's important to make a distinction between the primary definition node and all other primary nodes, we refer to the latter as `non-MDN nodes`.

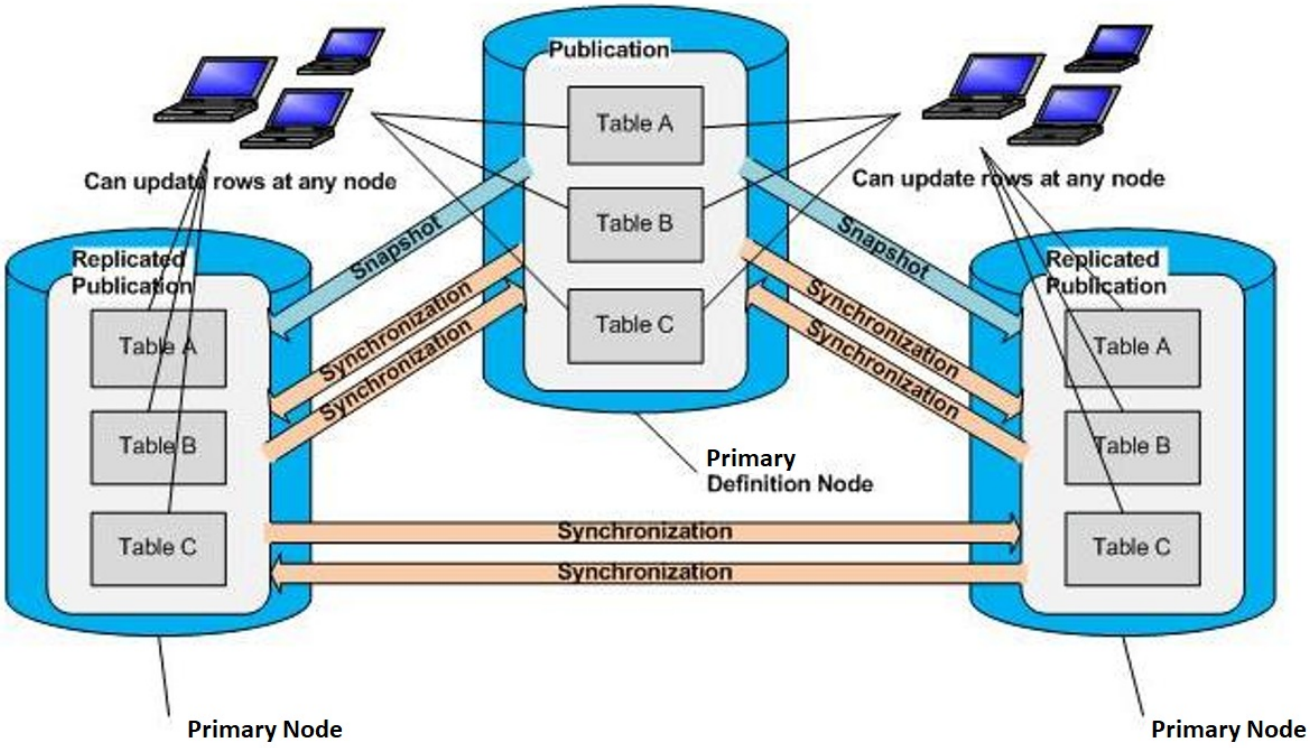
The primary definition node has the following significance:

- The publication is first created in the primary definition node, and the tables comprising the publication must exist in the database to be designated as the primary definition node at the time the publication is defined.
- The publication can be initially replicated to other primary nodes by means of a snapshot from the primary definition node.
- Each subsequent primary node added to the replication system must either:
    - Contain no tables with the same schema-qualified names as the publication tables in the primary definition node
    - Contain all publication table definitions as they exist in the primary definition node with the same schema-qualified names In the first case, when you add the primary node, you select the option to replicate the publication schema from the primary definition node. In the second case, you don't select this option.

- The table rows in a primary node can be reloaded from the primary definition node. The primary node tables are truncated and the rows reloaded by a snapshot from the primary definition node.

Once the multi-master replication system is defined, changes (inserts, updates, and deletions) to rows of the publication tables on any primary node are synchronized to all other primary nodes on either an on-demand or scheduled basis.

Generally, don't make changes to the table definitions in any of the primary nodes, including the primary definition node. If such changes are made, they don't propagate to other nodes in the multi-master replication system unless they're made using the DDL change replication feature described in Replicating DDL changes. If changes are made to tables without using the DDL change replication feature, there's a risk that future replication attempts will fail.



4.2.5 Asynchronous

Replication Server performs replications asynchronously. The systems hosting the databases don't always have to be running continuously for successful replication to occur. If one system goes offline, replication resumes when it comes back online if there is still pending data to replicate.

In addition, you can create a schedule for your replication system. Replication Server starts and performs replications regularly according to the assigned schedule. Creating a schedule allows you to run the replication system unattended. See Creating a schedule.

4.2.6 Snapshot and synchronization overview

Replication Server performs two different types of replications. These two main types are called *snapshot replication* and *synchronization replication*.

In either method, the source tables refer to the tables from which the replication data is originating. In a single-master replication system, it's the publication. In a multi-master replication system it's the primary node whose changes are being replicated to another primary node.

The target tables are the tables that are receiving the replication data from the source tables. In a single-master replication system, it's the subscription tables. In a multi-master replication system, it's the primary node receiving changes from another primary node.

In snapshot replication, all existing rows in the target tables are deleted using the database system's `TRUNCATE` command. The tables are then completely reloaded from the source tables of the publication.

In synchronization replication, only the changes (inserts, updates, and deletions) to the rows in the source tables since the last replication are applied to

the target tables.

> **Note**
>
> Deleting all rows in a source table executed by the `SQL TRUNCATE` command results in replication to the target tables only if the log-based method of synchronization replication is used. If the trigger-based method of synchronization replication is used, executing the `TRUNCATE` command on a source table doesn't replicate the effect to the target tables. You must perform a snapshot from the source table to the target tables if you use the trigger-based method.

Synchronization replication is implemented using two different methods: the trigger-based method and the log-based method.

In the trigger-based method, changes to rows in the source tables result in row-based triggers executing. These triggers record the changes in shadow tables. The changes recorded in the shadow tables are then periodically extracted from the shadow tables, converted to an in-memory data structure, and applied to the target tables by means of SQL statements executed using JDBC. See Synchronization replication with the trigger-based method for information on the trigger-based method.

In the log-based method, changes to rows in the source tables are extracted from the write-ahead log segments (`WAL` files) using asynchronous streaming replication implemented by the logical decoding feature available in Postgres database servers. The extracted changes are converted to an in-memory data structure and applied to the target tables by means of SQL statements executed using JDBC. See Synchronization replication with the log-based method for information on the log-based method.

In a multi-master replication system, the manner in which changes accumulated on all primary nodes are replicated to all other primary nodes is conceptually done in groups identified by the source primary node with the changes to be replicated. See Multi-master parallel replication for information on this process and the improvement for the log-based method with parallel replication.

In a single-master replication system, always do the first replication to a newly created subscription with a snapshot. You can perform subsequent replications by snapshot or by synchronization, provided that the publication isn't defined as a snapshot-only publication, as discussed in Snapshot-only publications.

In a multi-master replication system, you must do the first replication from the primary definition node to a newly added primary node with a snapshot. Subsequent replications between primary nodes occur by synchronization. However, it is possible to perform subsequent snapshots from the primary definition node to any other primary node.

4.2.7 Snapshot-only publications

When a publication is created in a single-master replication system, you can define the publication as a snapshot-only publication. Replication from a snapshot-only publication can be done only by using the snapshot replication method. Synchronization replication isn't permitted on a snapshot-only publication.

You can't create a snapshot-only publication in a multi-master replication system.

See Performance considerations for a discussion of the advantages of using a snapshot-only publication.

4.2.8 Snapshot replication

In snapshot replication, the target tables are completely reloaded from the source tables. The database system's truncate operation is used to delete all rows from the target tables.

For Oracle and SQL Server only: Oracle and SQL Server target tables are loaded using JDBC batches of `INSERT` statements.

For Postgres only: In general, Postgres target tables are loaded using the `JDBC COPY` command, since using truncation and `COPY` is generally faster than executing an `SQL DELETE` statement against the entire table and then adding the rows using JDBC batches of `INSERT` statements. If the `COPY` command fails, the publication server retries the snapshot using JDBC batches of `INSERT` statements.

If the target table (regardless of database type) contains a large object data type such as `BYTEA`, `BLOB`, or `CLOB`, then rows are loaded one at a time per batch using an `INSERT` statement. This approach avoids a heap space error resulting from potentially large rows. Loading time can be decreased by allowing multiple inserts per batch, which you can do by adjusting the configuration option `lobBatchSize` described in Optimizing snapshot replication.

> **Note**
>
> EDB Postgres Advanced Server supports a number of aliases for data types. Such aliases that translate to `BYTEA` are treated as large object data types. See SQL reference for a listing of EDB Postgres Advanced Server data types. (See Database compatibility for Oracle developers for EDB Postgres Advanced Server version 9.5 or earlier versions.)

Under certain circumstances, the corresponding Postgres target table created for certain types of Oracle partitioned tables is a set of inherited tables. In these cases, the `SQL DELETE` statement is used on the inherited child tables instead of truncation. See Replicating Oracle partitioned tables.

A server configuration option is available that forces the snapshot replication process to use the Oracle database link utility instead of `JDBC COPY` to populate the Postgres target tables from an Oracle publication. Oracle database link provides a performance improvement over `JDBC COPY`. See Optimizing snapshot replication for information on using the Oracle database link option as well as various configuration options to optimize snapshot replication.

4.2.9 Synchronization replication with the trigger-based method

If a publication in a single-master replication system is created for use in synchronization replications with the trigger-based method, the publication server installs an insert trigger, an update trigger, and a delete trigger on each publication table. In a multi-master replication system, each replicated table in each primary node using the trigger-based method has an insert trigger, an update trigger, and a delete trigger.

Oracle and SQL Server databases can only be used as publication databases with the trigger-based method. Additionally, Oracle and SQL Server databases can be used as subscription databases with the WAL-based replication method.

The publication server also creates a shadow table for each source table on which triggers were created. A shadow table is a table used by Replication Server to record the changes (inserts, updates, and deletions) made to a given source table. A shadow table records three types of record images:

- For each row inserted into the source table, the shadow table records the image of the inserted row.
- For each existing row that is updated in the source table, the shadow table records the after image of the updated row.
- For each row deleted from the source table, the shadow table records the identity columns (primary key or unique columns) value of the deleted row.

> **Note**
>
> In a multi-master replication system, the before image of an updated row is also stored in the shadow table to perform update conflict detection. See Conflict resolution for information on conflict detection in a multi-master replication system.

After each change on the source table, one of the insert, update, or delete triggers executes. These are row triggers, so for each row affected by the change, the trigger executes. Each execution of the trigger records a row of the appropriate type (insert, update, or deletion) in the shadow table of the corresponding source table.

Though changes made to the source tables since the last replication occurred are applied to the target tables using `SQL INSERT` `UPDATE`, and `DELETE` statements, the actual SQL statements run against the target tables aren't the same SQL statements that were run against the source tables.

When synchronization replication occurs, the publication server executes JDBC batches of SQL statements (also referred to as transaction sets) against the target tables. The batches contain:

- An `INSERT` statement for each shadow table row recording an insert operation
- An `UPDATE` statement for each shadow table row recording an update operation

- A `DELETE` statement for each shadow table row recording a delete operation.

Each batch is executed in one transaction.

You can view shadow table rows that were applied to target tables as shadow table history in the Replication Server console (see Shadow table history).

> **Note**
>
> A single SQL statement executed against a source table can result in many rows recorded in a shadow table and, therefore, many SQL statements executed against the target table. For example, if a single `UPDATE` statement affects 10 rows in the source table, 10 rows are inserted into the shadow table, one for each row in the source table that was updated. When the publication server applies the changes to the target table, 10 `UPDATE` statements execute.

> **Note**
>
> For greater efficiency, when changes to the source tables consist of SQL statements that each affect a large number of rows, the publication server can use prepared SQL statements. See Optimizing synchronization replication for directions on how to control the use of prepared SQL statements as well as information on various other configuration options to optimize synchronization replication.

4.2.10 Synchronization replication with the log-based method

In PostgreSQL 9.4, a feature was introduced called logical decoding (also called logical replication or changeset extraction). Logical decoding lets you extract data manipulation language (DML) changes from the write-ahead log segments (`WAL` files) in a readable format.

For information on logical decoding see the PostgreSQL core documentation.

With logical decoding, you can capture data changes to the publication tables without impacting the online transaction processing rate against these tables that occurs when using the trigger-based method. The trigger-based method results in firing row-level triggers whenever data changes occur and then inserting these data changes into shadow tables for temporary storage. It then applies the changes to the target databases.

Thus, extracting data changes using logical decoding can help improve database server throughput and replication latency.

However, the logical decoding interface streams changes for all tables in a given database, which can have a performance overhead associated with it. For example, if a database contains 100 tables, and you are interested in replicating only a small subset of these tables (for example, only 20 tables in a single publication), the logical decoding protocol streams changes for all 100 tables to the publication server. The publication server eventually filters out the changes for the irrelevant 80 tables. However, this results in network overhead caused by the additional changeset load that isn't required by the replication system.

Using logical decoding to extract changes from a publication database during EPRS synchronization replication is referred to as the log-based method.

4.2.10.1 Requirements and restrictions

The following are the general requirements and restrictions when using the log-based method for any database of a single-master or multi-master replication system:

- The selection of either the trigger-based method or the log-based method is a characteristic that applies only to the publication database. The choice is made when defining the primary database of a single-master replication system (see Adding a publication database) or the primary definition node of a multi-master replication system (see Adding the primary definition node).
- The logical decoding feature, and hence the log-based method, is supported beginning with PostgreSQL version 9.4. Therefore, to use the log-based method for a publication database, that publication database must be running under PostgreSQL version 9.4 or later or under EDB Postgres Advanced Server version 9.4 or later.
- In a single-master replication system, whether the primary database uses the trigger-based method or the log-based method has no added impact on the rules for choosing the subscription database. For example, even if you choose the log-based method for the primary database, the

subscription database can run on Postgres version 9.4 and any supported, earlier version of Postgres, as well as Oracle or SQL Server.
- In a single-master replication system, the primary database can contain one or more publications (that is, named sets of tables for replication). This rule applies to a primary database using either the trigger-based method or the log-based method.
- You can have multiple, single-master replication systems running under a publication server. Some primary databases can use the trigger-based method while others use the log-based method.
- In a multi-master replication system, selecting either the trigger-based method or the log-based method on the primary definition node determines the method for all other primary nodes. In other words, if you choose the trigger-based method for the primary definition node, then all other primary nodes use the trigger-based method. If you choose the log-based method for the primary definition node, then all other primary nodes use the log-based method.
- As a consequence this restriction, to use the log-based method for a multi-master replication system, all of the primary nodes of the system must be running under Postgres version 9.4 or later. All such Postgres database clusters must be configured to use logical decoding for the log-based method.

Selecting the log-based method for any database impacts the configuration of the Postgres database cluster containing that database.

If you plan to use the log-based method with any publication database running under a Postgres database server, the following configuration parameter settings are required in the configuration file `postgresql.conf` of that Postgres database server:

- `wal_level`. Set to logical.
- `max_wal_senders`. Specifies the maximum number of concurrent connections (that is, the maximum number of simultaneously running WAL sender processes). Set, at minimum, to the total number of primary databases of single-master replication systems and primary nodes of multi-master replication systems on this database server that uses the log-based method.
- `max_replication_slots`. Specifies the maximum number of replication slots. If the database server supports both single-master replication systems and multi-master replication systems, then set max_replication_slots, at minimum, to the sum of the requirements for both replication systems. For support of SMR systems, the minimum requirement is the total number of primary databases of the single-master replication systems that use the log-based method. For support of MMR systems, the minimum requirement is the total number of primary nodes in the multi-master replication system multiplied by the number of primary nodes residing on this database server. For information, see Replication origin.
- `track_commit_timestamp`. Set to `on`. This configuration parameter applies only to Postgres database servers version 9.5 and later. See Configuration parameter and table setting requirements for more information.

Also, see Enabling synchronization replication with the log-based method for setting these parameters for a single-master replication system. See Enabling synchronization replication with the log-based method for a multi-master replication system.

In addition, the `pg_hba.conf` configuration file of the Postgres database server must contain an entry permitting `REPLICATION` access for each database using the log-based method running on the database server. Give this access to the publication database user specified when creating the publication database definition using the Replication Server console for a single-master replication system, for a multi-master replication system, or the Replication Server command line interface (CLI). For more information, see:

- Adding a publication database
- Adding the primary definition node
- Adding a publication database (addpubdb)

See Postgres server authentication for setting `REPLICATION` access for a single-master replication system. See Verifying host accessibility for a multi-master replication system.

For configuration options in the publication server configuration file that specifically apply to the log-based method seeLog-based method of synchronization options.

4.2.10.2 Logical replication slots

When using the log-based method on a publication database, the underlying logical decoding framework exposes the data changes (the changeset stream) by means of a logical replication slot.

A logical replication slot represents a changeset stream and applies to a single database. Replication Server assigns a unique identifier, called the slot name, to each logical replication slot it creates in the form `xdb_<dboid>_<pubid>` where `<dboid>` is the publication database object identifier (OID) and `<pubid>` is the publication ID assigned by Replication Server. All slot names are unique in a Postgres database cluster.

Thus, for each single-master replication system using the log-based method, a replication slot is required for the publication database of each such

system.

For a multi-master replication system using the log-based method, each primary node requires a replication slot.

The maximum number of replication slots permitted for a database server is controlled by the `max_replication_slots` configuration parameter in the `postgresql.conf` file. Therefore, you must set this configuration parameter to a value large enough to account for:

- All publication databases defined with the log-based method of single-master replication systems running on the database server.
- All primary nodes of a multi-master replication system defined with the log-based method running on the database server.

Additional replication slots are required to support the use of replication origin (see Replication origin).

For more information on configuration parameters for single-master replication systems, see Enabling synchronization replication with the log-based method.

For multi-master replication systems, see Enabling synchronization replication with the log-based method.

4.2.10.3 Streaming replication with the WAL sender process

The changeset stream is accessible to the Replication Server publication server by the *WAL sender process* (`walsender`) using the streaming replication protocol.

The Replication Server publication server connects using the `walsender` interface through which changes are streamed on a continual basis. The continuous streaming eliminates the need for explicitly polling for changes.

The following are the basic synchronization steps using the log-based method:

1. A streaming replication connection to the database server opens using `libpq` to establish a `walsender` communication channel.
2. A separate thread monitors data changes streamed through the `walsender` interface.
3. As the data changes become available, they transform to populate an in-memory cache.
4. On the next scheduled interval, the in-memory cached data changes apply to each of the target databases in JDBC batches of SQL statements (referred to as transaction sets) in the same manner as described in Synchronization replication with the trigger-based method for the trigger-based method. If one or more target database servers aren't accessible, the data changes save in a local file on the host running the publication server. See In-memory caching and persistence for information on in-memory caching and data persistence.
5. The value of the `WAL` segment's log sequence number (`LSN`) identifying the last set of applied changes based on the last replicated transaction updates. The update is confirmed to the database server.
6. The applied data changes clear from the in-memory cache.
7. Steps 3 through 6 repeat.

> **Note**
>
> A single SQL statement executed against a source table can result in many rows modified and returned in the changeset stream and, therefore, many SQL statements executed against the target table. For example, if a single `UPDATE` statement affects 10 rows in the source table, 10 rows are returned in the changeset stream, one for each row in the source table that was updated. When the publication server applies the changes to the target table, 10 `UPDATE` statements are executed.

4.2.10.4 Replication origin

Starting with Postgres version 9.5, replication origin was introduced to the logical decoding framework. Replication origin allows an application to identify, label, and mark certain aspects of a logical decoding session.

For information on replication origin, see the PostgreSQL core documentation.

For the log-based method of synchronization replication, replication origin provides performance improvement if the primary nodes are running under Postgres version 9.5 or later.

The log-based method uses the `WAL` files to obtain the changes applied to the publication tables. After the changes are retrieved through the `walsender` interface, the publication server applies the set of changes to the other primary nodes using transaction sets consisting of JDBC batches of SQL statements. When these changes are applied to the tables in the other target primary nodes, the same changes are also recorded in the `WAL` files of each database server hosting the target primary nodes.

These redundant or *replayed* changes are included in the changeset stream received by the publication server. Ignore and don't apply these replayed changes since they are duplicates of all changes that were already applied to the target tables through the JDBC batches.

The replayed changes result in performance overhead, as all such changes are transmitted over the network from the database server to the publication server. Then the publication server must discard such redundant changes.

With replication origin, the publication server can set up the logical decoding sessions so that these replayed changes aren't included in the changeset stream transmitted over the network to the publication server. This configuration eliminates the performance overhead.

The following are the conditions under which replication origin is used:

- Replication origin applies to multi-master replication systems only, not to single-master replication systems.
- Replication origin eliminates streaming of replayed changes only from Postgres versions 9.5 or later. Replayed changes are still included in the changeset stream from Postgres version 9.4 but are discarded by the publication server. Thus, multi-master replication systems consisting of both Postgres versions 9.4 and 9.5 use the replication origin advantage on the 9.5 database servers.
- You must set the `max_replication_slots` configuration parameter at a certain minimal level to ensure that the publication server can create the additional replication slots for replication origin.

For each primary node database, in addition to the replication slot used for the changeset stream, more replication slots are required. One added slot corresponds to every other primary node to support the replication origin use. Thus, for each primary node, the total number of replication slots required is equal to the total number of primary nodes in the entire MMR system.

Therefore, for a given database server (that is, a Postgres database cluster containing primary node databases), the total number of replication slots required is equal to the total number of primary nodes in the entire MMR system multiplied by the number of primary node databases residing in the given database cluster.

For example, assume the use of a six-node multi-master replication system using three database clusters as follows:

- Database cluster #1 contains three primary node databases.
- Database cluster #2 contains two primary node databases.
- Database cluster #3 contains one primary node database.

The total number of primary nodes is six. Multiply the number of primary node databases in each database cluster by six to give the required minimum setting for `max_replication_slots` for that database cluster.

The following table shows the required minimum settings for `max_replication_slots` as well as `max_wal_senders`.

Postgres database server	max_wal_senders	max_replication_slots
Cluster #1 (3 primary nodes)	3	18
Cluster #2 (2 primary nodes)	2	12
Cluster #3 (1 primary node)	1	6

If the `max_replication_slots` parameter isn't set to a high enough value, synchronization replication still succeeds but without the replication origin performance advantage.

The publication server log file contains the following warning in such cases:

```
WARNING: Failed to setup replication origin ``xdb_MMRnode_c_emp_pub_6``. Reason: ERROR: could not find
```

```
free replication state slot for replication origin with OID 4
  Hint: Increase max_replication_slots and try again.
```

The following example shows some of the replication slot information for a three-primary node system running on a single database cluster.

The following shows the maximum allowable number of replication slots:

```
SHOW max_replication_slots;
```

```
max_replication_slots
----------------------
 9
(1 row)
```

Use a number greater than the number of replication slots and replication origins currently allocated.

The following displays the replication slots:

```
SELECT slot_name, slot_type, database, active FROM pg_replication_slots ORDER BY
1;
```

```
  slot_name  | slot_type | database  | active
-------------+-----------+-----------+--------
 xdb_47877_5 | logical   | MMRnode_a | t
 xdb_47878_5 | logical   | MMRnode_b | t
 xdb_47879_5 | logical   | MMRnode_c | t
(3 rows)
```

The following shows the replication origins.

```
SELECT * FROM pg_replication_origin ORDER BY
2;
```

```
 roident |          roname
---------+--------------------------
       5 | xdb_MMRnode_a_emp_pub_39
       2 | xdb_MMRnode_a_emp_pub_6
       1 | xdb_MMRnode_b_emp_pub_1
       6 | xdb_MMRnode_b_emp_pub_39
       3 | xdb_MMRnode_c_emp_pub_1
       4 | xdb_MMRnode_c_emp_pub_6
(6 rows)
```

The replication origin name is assigned in the format `xdb_<srcdbname>_<pubname>_<remotedbid>`, where `<srcdbname>` is the source database name, `<pubname>` is the publication name, and `<remotedbid>` is the publication database ID of a remote database.

The replication slots are in the active state when the publication server is running. The replication slots are deactivated when the publication server is shut down.

The replication slots and replication origin sessions are deleted from the database cluster when their corresponding primary nodes are removed from the multi-master replication system using the Replication Server Console or the Replication Server CLI.

If the replication slots aren't properly deleted when required, see Dropping replication slots for log-based synchronization replication to learn how to delete them manually.

4.2.10.5 In-memory caching and persistence

The data changes are fetched and stored in memory buffers to optimize the data replication process. This technique avoids the overhead associated with repeatedly fetching the same set of changes from the database server with multiple target databases.

This approach is sufficient as long as all of the target databases are accessible during a replication event and the data fits in the available cache.

However, if one or more of the target databases is unavailable due to network connectivity problems, server down time, etc., the in-memory data changes must be persisted for later retrieval when the target databases becomes available for synchronization with the source database.

The Replication Server architecture uses *Java object serialization* to persist the in-memory state of the data. Object serialization converts object data and other relevant information to a sequence of bytes that can then be stored in a file.

The following are examples that can result in evicting in-memory data to persistent storage:

- Before the next replication event occurs, the in-memory cache is filled with the data changes and needs to be evicted to accommodate a new set of changes.
- The replication system has multiple target databases. During a synchronization event, all of the changes available in the cache are applied successfully to some of the target databases. However, one or more of the other target databases can't be accessed. All of the applied changes held in memory must be persisted and retained so that these changes can be reloaded and applied when the inaccessible databases becomes available.

The cache size corresponds to the heap size configured for the publication server by the `-Xmxnnnm` setting of the `JAVA_HEAP_SIZE` parameter in the Replication Server configuration file. See Replication configuration file for information on the Replication Server configuration file.

You can minimize the persistence I/O overhead by increasing the heap size value and defining a more frequent synchronization interval, such as for every few seconds. See Creating a schedule for information on setting a replication schedule.

The data changes are persisted in a local file on the host running the publication server. The file is stored in the directory `XDB_HOME/xdata`.

Each time persistence occurs, a new file is created. After the files are processed, they are periodically removed from disk.

4.2.11 Multi-master parallel replication

For a multi-master replication system, you can replicate transactions from one primary node to another by using either the trigger-based method (see Synchronization replication with the trigger-based method) or the log-based method (see Synchronization replication with the log-based method).

For a single replication event to be considered finished and complete, transactions that occurred on all primary nodes since the previous replication event must be successfully replicated to all other primary nodes by the configured synchronization method.

This replication consists of a series of multiple replication sets, each identified by a primary node acting as the source primary node, which contains the transactions that need to be replicated to all other primary nodes acting as the target primary nodes. So for a multi-master replication system consisting of n number of primary nodes, there are n such replication sets, each with a different primary node acting as the source.

Since the initial support of multi-master replication systems in Replication Server version 5.0, such a series of multiple replication sets always started in a strictly serial manner. That is, the transaction replication from a source primary node to all target primary nodes must be completed before the start of the transaction replication from the next primary node to all other target primary nodes, and so on.

For example, consider a three-primary node system consisting of primary node A, primary node B, and primary node C.

If applications have applied transactions to tables in all three primary nodes and then a synchronization replication event starts either on demand by the Replication Server console, a Replication Server CLI command, or by a scheduled replication, the transactions are replicated in the following manner:

1. Transactions that were made on primary node A are replicated to primary node B and primary node C.
2. When Step 1 completes, transactions that were made on primary node B are replicated to primary node A and primary node C.

3. When Step 2 completes, transactions that were made on primary node C are replicated to primary node A and primary node B.

The time to complete the entire replication event, referred to as the latency time, is basically the sum of the replication times, where each primary node acts as the source (that is, the sum of the times for steps 1, 2, and 3).

For the log-based method, this latency time is reduced by implementing parallel replication. Each replication set from a given primary node acting as the source executes and runs simultaneously with all other replication sets where the other primary nodes act as the source.

Thus, a replication set from a primary node isn't waiting for others to complete before it can start. Steps 1, 2, and 3 all run simultaneously instead of one after the other.

> **Note**
>
> The parallel replication applies only to the log-based method, not to the trigger-based method.

You don't need to enable the use of of parallel replication for the log-based MMR system in the configuration settings.

> **Note**
>
> In addition to parallel replication, optimizing replicating from a given primary node to all other primary nodes (that is, in the context of a single replication set) was implemented with the use of multiple threads. This is referred to as *parallel synchronization*. Parallel synchronization applies to both the trigger-based and log-based methods. See Parallel synchronization.

4.2.12 Table filters

Table filters specify the selection criteria for rows in publication tables or views to include during replications to subscriptions from the publication database in a single-master replication system. These table filters also apply between primary nodes in a multi-master replication system. Rows that don't satisfy the selection criteria are excluded from replications to subscriptions or primary nodes on which these table filters were enabled.

Implementing table filters

Implementing table filters is a two-part process. First, you must define a set of available table filters. You can define this set while creating the publication by defining specific, named rules that apply to selected publication tables or views expressed in the form of `SQL WHERE` clauses.

While simple conditions with comparison and logical operators (=, >, <, LIKE, etc.) are allowed as filters for all use cases, in one use case you can use a more complex condition as a filter. You can use a subquery to establish an inclusion/exclusion list of IDs if a single column is returned from the subquery. This technique is supported only for trigger-based SMR use cases. Here are the patterns for this type of query:

```
PUBLISHED_TABLE_COLUMN_NAME IN (SELECT COLUMN_NAME FROM
TABLE_NAME)

PUBLISHED_TABLE_COLUMN_NAME NOT IN (SELECT COLUMN_NAME FROM
TABLE_NAME)
```

For example, for tables `dept` and `dept_filter_lookup` created with:

```
CREATE TABLE public.dept
(

deptno NUMERIC(2) PRIMARY KEY,

dname VARCHAR(14) UNIQUE,

loc VARCHAR(13)
```

```
);

CREATE TABLE public.dept_filter_lookup
(

id NUMERIC(2) PRIMARY KEY

);
```

You can use the following filters:

```
deptno IN (SELECT id FROM
public.dept_filter_lookup)
```

or

```
deptno NOT IN (SELECT id FROM
public.dept_filter_lookup)
```

Use these filters to create a publication with the following RepCLI commands:

```
$ java -jar edb-repcli.jar -createpub  pub -repsvrfile ~/pubsvrfile -pubdbid 1 -tables public.dept -
reptype T \

-tablesfilterclause "public.dept:lookup_filter:deptno IN (SELECT id FROM public.dept_filter_lookup)"
```

or

```
$ java -jar edb-repcli.jar -createpub  pub -repsvrfile ~/pubsvrfile -pubdbid 1 -tables public.dept -
reptype T \

-tablesfilterclause "public.dept:lookup_filter:deptno NOT IN (SELECT id FROM public.dept_filter_lookup)"
```

After you define a set of available table filters, you must enable them only on those subscription tables of either:

- A single-master replication system
- Primary node tables of a multi-master replication system where filtering is to occur during replication to those particular target tables.

No filtering occurs during replication to a target subscription table or primary node table if you didn't enable filters specifically on that table in the subscription or primary node.

We strongly recommend that you perform a snapshot replication to the subscriptions or primary nodes that contain tables on which the filtering criteria changed by adding filter rules, removing filter rules, or modifying existing filter rules.

A snapshot ensures that the content of the subscription tables or primary node tables is consistent with the updated filtering criteria.

> **Note**
>
> When using table filters in a multi-master replication system, the primary definition node, which provides the source of the table content for a snapshot, must contain a superset of all the data contained in the other primary nodes of the multi-master replication system. This configuration ensures that the target of a snapshot receives all of the data that satisfies any filtering criteria enabled on the other primary nodes.

Conversely, if the primary definition node contains only a subset of all the data contained in the other primary nodes, then a snapshot to another primary node might not result in the complete set of data that is required for that target primary node.

Effects of table filtering

A filter enabled on a table affects only the results from snapshot or synchronization replications targeted to that table by Replication Server. Filtering has no effect on changes made directly on the target table by external user applications such as an SQL command line utility.

Filtering has the following effects on a targeted, filtered table.

> **Note**
>
> In the following discussion, a result set refers to the set of rows in a table satisfying the selection criteria of an `UPDATE` or `DELETE` statement executed on that table.

In a snapshot replication, if the row satisfies the filtering criteria, a row from the source table of the snapshot is inserted into the target table. Otherwise the row is excluded from insertion into the target table.

When an `INSERT` statement is executed on a source table followed by a synchronization replication, if the row satisfies the filtering criteria, the row is inserted into the target table of the synchronization. Otherwise the row is excluded from insertion into the target table.

When an `UPDATE` or `DELETE` statement executes on a source table followed by a synchronization replication, values of the identity columns are used in action on the target table of the synchronization. For non-PostgreSQL/EDB Postgres Advanced Server databases, the primary key columns are the identity columns. For PostgreSQL/EDB Postgres Advanced Server databases, the primary key or unique columns are the identity columns (see Design considerations for details).

When an `UPDATE` statement executes on a source table followed by a synchronization replication, the `UPDATE` result set of the source table determines the action on the target table of the synchronization as follows:

- If a row in the result set has no corresponding row in the target table with the same identity columns values and the updated row in the result set satisfies the filtering criteria, then the row is inserted into the target table. A row that previously didn't exist in the target table is added because the updated row in the source table now satisfies the filtering criteria.
- If a row in the result set has a corresponding row in the target table with the same identity columns values and the updated row in the result set satisfies the filtering criteria, then the row in the target table is updated accordingly. The update is applied to an existing, matching row in the target table that still satisfies the filtering criteria after the update.
- If a row in the result set has a corresponding row in the target table with the same identity columns values and the updated row in the result set no longer satisfies the filtering criteria, then the corresponding row in the target table is deleted. An existing, matching row in the target table no longer satisfies the filtering criteria after the update, so the row is removed from the target table.

When a `DELETE` statement executes on a source table followed by a synchronization replication, the `DELETE` result set of the source table determines the action on the target table of the synchronization as follows:

- If a row in the result set has a corresponding row in the target table with the same identity columns values, then the row with that primary key value is deleted from the target table. An existing, matching row in the target table is removed.
- If a row in the result set has no corresponding row in the target table with the same identity columns values, then no action is taken on the target table for that row. Because there's no existing, matching row in the target table, there's no row to remove from the target table.

Thus, regardless of whether the transaction on the source table is an `INSERT`, `UPDATE`, or `DELETE` statement, the goal of a table filter is to ensure that all rows in the target table satisfy the filter rule.

Table settings and restrictions for table filters

For table filters, you can apply specific table settings and restrictions.

REPLICA IDENTITY setting for filtering in a log-based replication system

For replication systems using the log-based method of synchronization replication, a publication table on which you define a filter must have the

`REPLICA IDENTITY` option set to `FULL`.

> **Note**
>
> This `REPLICA IDENTITY FULL` setting isn't required for tables in single-master, snapshot-only publications, See Snapshot-only publications.

Use the `ALTER TABLE` command for this setting:

```
ALTER TABLE schema.table_name REPLICA IDENTITY FULL
```

For additional information see the ALTER TABLE SQL command in thePostgreSQL Core documentation.

For example, for a publication table named `edb.dept`, use the following `ALTER TABLE` command:

```
ALTER TABLE edb.dept REPLICA IDENTITY FULL;
```

You can see the `REPLICA IDENTITY` setting by using the `\d+` command in the `PSQL` utility:

```
edb=# \d+ edb.dept

                                Table "edb.dept"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 deptno | numeric(2,0)          | not null  | main     |              |
 dname  | character varying(14) |           | extended |              |
 loc    | character varying(13) |           | extended |              |
Indexes:
    "dept_pk" PRIMARY KEY, btree (deptno)
    "dept_dname_uq" UNIQUE CONSTRAINT, btree (dname)
Referenced by:
    TABLE "emp" CONSTRAINT "emp_ref_dept_fk" FOREIGN KEY (deptno) REFERENCES dept(deptno)
    TABLE "jobhist" CONSTRAINT "jobhist_ref_dept_fk" FOREIGN KEY (deptno) REFERENCES dept(deptno) ON DELETE
SET NULL
Replica Identity: FULL
```

The `REPLICA IDENTITY FULL` setting is required on tables in the following databases of a log-based replication system:

- In a single-master replication system, you define table filters in the primary database. Thus, the publication tables in the primary database requiring filter definitions must be altered to a `REPLICA IDENTITY FULL` setting but only if the publication is not a snapshot-only publication. See Snapshot-only publications.
- In a multi-master replication system, table filters are defined in the primary definition node. Thus, publication tables in the primary definition node requiring filter definitions must be altered to a `REPLICA IDENTITY FULL` setting.
- In a multi-master replication system, don't set non-MDN nodes tables `REPLICA IDENTITY` options to `FULL` unless transactions are expected to be targeted on those non-MDN nodes. In addition, the transactions will be filtered when they're replicated to the other primary nodes.

The `REPLICA IDENTITY FULL` setting on a source table ensures that certain types of transactions on the source table result in the proper updates to the target tables on which filters are enabled.

> **Note**
>
> In addition to table filtering requirements, you might need the `REPLICA IDENTITY FULL` on publication tables for other reasons in Replication Server. See Configuration parameter and table setting requirements for additional requirements.

Filtering restrictions on data types

Table filters aren't supported on binary data type columns. A binary data type is the Postgres data type `BYTEA`. In addition, table filters aren't supported on EDB Postgres Advanced Server columns with data types `BINARY`, `VARBINARY`, `BLOB`, `LONG RAW`, and `RAW`, as these are alias names for the `BYTEA` data type.

Filtering restrictions on operators

Replication Server supports modulus operator (denoted by %) to define a filter clause. However, the following restrictions apply:

- You can have only a single filter condition using the modulus operator.
- You can't use it with any other conditions using `AND` or `OR` operators.

Replication Server supports the modulus filter in the following formats:

```
deptno%3=0
```

```
@deptno%3=0
```

Roadmap for further instructions

The specific details on implementing table filtering depend upon whether you're using a single-master replication system or a multi-master replication system. Use this roadmap to find relevant information for each type of replication system.

For using table filters in a single-master replication system, see:

- Adding a publication for information on defining the initial set of table filters available for selective enablement on subscriptions
- Adding a subscription for information on enabling available table filters on a newly created subscription
- Updating the set of available table filters in a publication for information on adding, removing, or modifying rules comprising the set of available table filters
- Enabling/disabling table filters on a subscription for information on changing the table filters enabled on an existing subscription

For using table filters in a multi-master replication system see:

- Adding a publication for information on defining the initial set of table filters available for selective enablement on primary nodes
- Creating additional primary nodes for information on enabling available table filters on a newly created primary node
- Updating the set of available table filters in a publication for information on adding, removing, or modifying rules comprising the set of available table filters
- Enabling/disabling table filters on a subscription for information on changing table filters enabled on an existing primary node

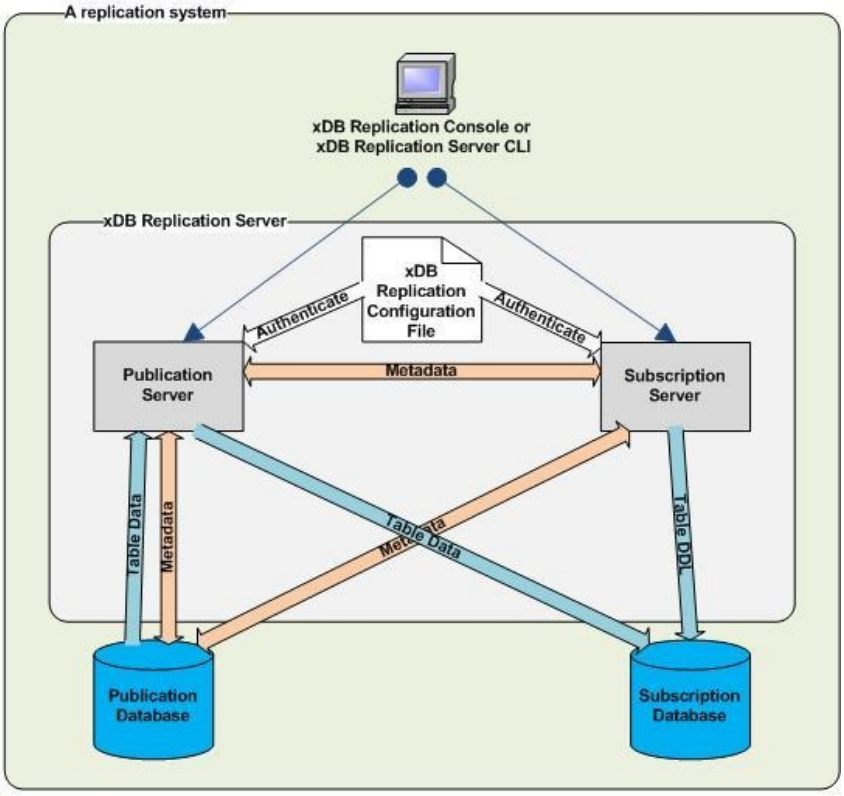
4.3 Replication Server components and architecture

The components and architecture of Replication Server include physical components and logical components.

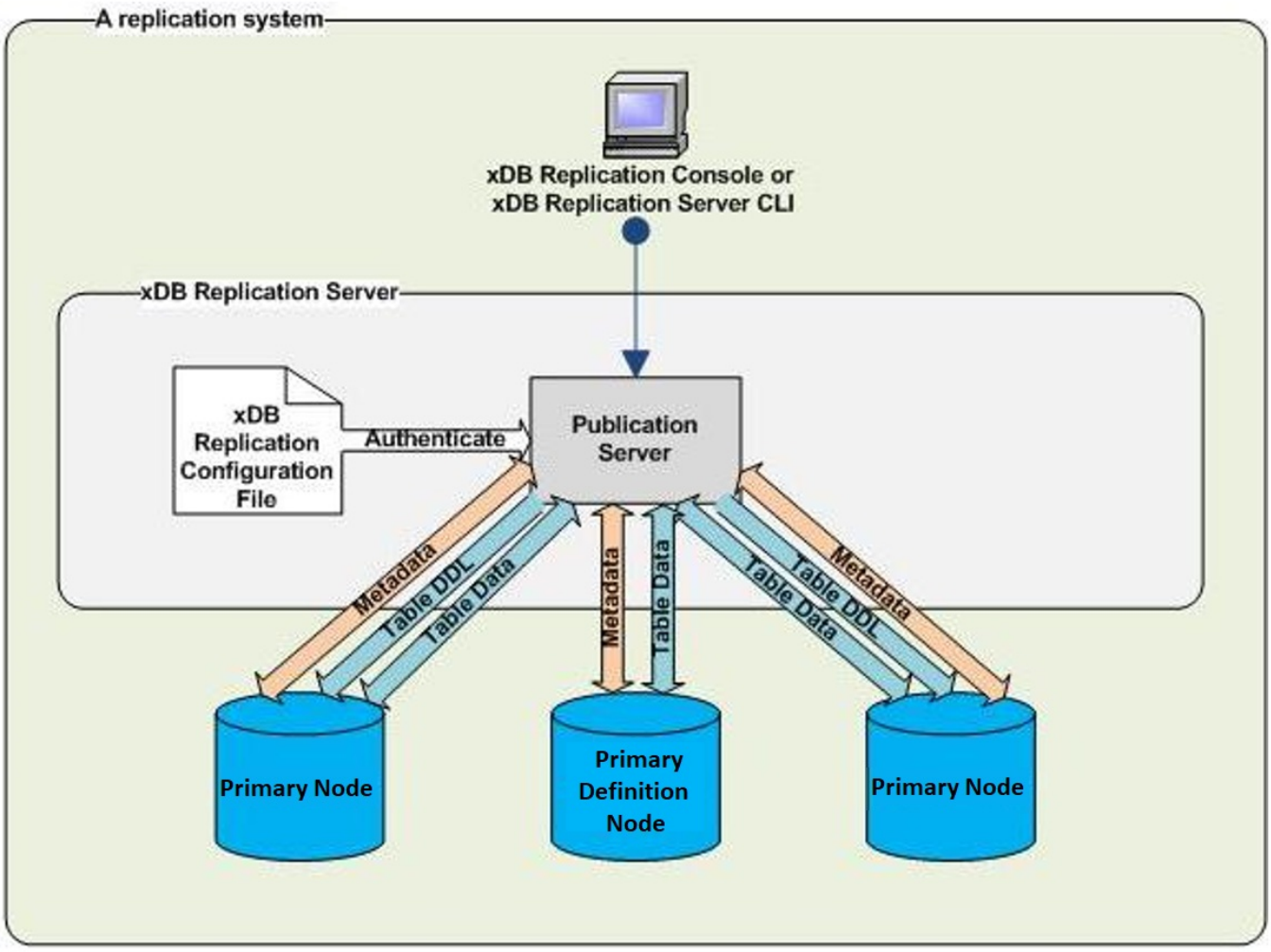
4.3.1 Physical components

Replication Server isn't a single, executable program but rather a set of programs along with data stores containing configuration information and metadata. These components work together to form a replication system.

The following diagram shows the components of Replication Server and how they're used to form a complete, basic, single-master replication system.



The following diagram shows the components of Replication Server and how they're used to form a complete, basic, multi-master replication system.



The minimal configuration of Replication Server for a basic replication system consists of the following software components:

- **Publication server** — The program that configures the publication database and primary nodes for replication and performs replication.
- **Subscription server** — The program that configures the subscription database for replication and initiates replication. The subscription server is used only in single-master replication systems.
- **Replication configuration file** — Text file containing connection and authentication information used by the publication server and subscription server upon startup to connect to a publication database designated as the controller database. Also used to authenticate registration of the publication server and subscription server from the user interface when creating a replication system.
- **Startup configuration file** — Text file containing installation and configuration information used for the Java Runtime Environment when the publication server and subscription server start.

The entire replication system is completed with the addition of the following components:

- User interfaces for configuring and maintaining the replication system
- One or more publication databases for a single-master replication system
- One or more subscription databases for a single-master replication system
- One primary definition node for a multi-master replication system
- One or more additional primary nodes for a multi-master replication system

The user interface, publication server, subscription server, publication database, subscription database, and primary nodes can all run on the same host or on separate, networked hosts.

You can use any number of user interfaces to access any number of publication servers and subscription servers on the network as long as you know the network locations, user names, and passwords of the publication and subscription servers.

Any number of publication and subscription databases can participate in a single-master replication system.

Any number of primary nodes can participate in a multi-master replication system.

Publication server

The publication server creates and manages the metadata for publications. When a publication is created, the publication server creates database objects in the control schema of the publication database to record metadata about the publication.

Whenever a primary node is added to a multi-master replication system, the publication server creates database objects in the control schema of the primary node for recording metadata. For non-MDN nodes, the publication server also calls Migration Toolkit to create the publication table definitions if you choose this option when creating the primary node.

> **Note**
>
> See Control schema and control schema objects for information on the control schema.

The publication server is also responsible for performing a replication. For snapshot replications, the publication server calls Migration Toolkit to perform the snapshot.

For single-master synchronization replications, the publication server uses the Java Database Connectivity (JDBC) interface to apply changes to the subscription table rows based on changes recorded in either of two ways:

- If the publication database is running under Postgres version 9.4 or later and the logical decoding option was chosen when creating the publication, changes are obtained from the Postgres `WAL` files using a logical replication slot.
- In all other circumstances, changes are recorded in metadata tables (called shadow tables) in the publication database by row-based triggers that activate upon any insert, update, or deletion to the publication table rows.

For multi-master synchronization replications, the publication server performs the same process as for single-master synchronizations but does so for each primary node pair combination in the multi-master replication system.

The publication server can run on the same host as the other Replication Server components, or it can run on a separate, networked host.

Subscription server

> **Note**
>
> The subscription server is required only for single-master replication systems. The subscription server doesn't need to be running or installed if only multi-master replication systems are in use.

The subscription server creates and manages the metadata for subscriptions. When a subscription is created, the subscription server creates database objects in the control schema of the publication database to record metadata about the subscription.

When a subscription is created, the subscription server calls Migration Toolkit to create the subscription table definitions in the subscription database. The rows in the subscription tables aren't populated until a replication occurs. Rows are populated by actions of the publication server.

The subscription server is also responsible for initiating a replication as a result of manual user action through the user interface or a schedule created for the subscription. The subscription server initiates a call to the publication server that manages the associated publication. The publication server then performs the replication.

The subscription server can run on the same host as the other Replication Server components, or it can run on a separate, networked host.

When the subscription server starts, it uses the information in the replication configuration file found on its host to connect to the designated controller database.

Replication configuration file

The Replication Server replication configuration file contains the connection and authentication information used by any publication server or subscription server running on the host containing the file.

Specifically, the replication configuration file is accessed in the following circumstances:

- When a publication server or subscription server is started on the host.
- When a publication server or subscription server is registered during the process of creating a replication system. Register a publication server or subscription server using the Replication Server console or command line interface.

The following table contains a brief description of the parameters in the replication configuration file.

Parameter	Description
`admin_user`	Replication Server administrator user name (the admin user name) for registering a publication server or a subscription server on this host containing the replication configuration file
`admin_password`	Encrypted password of the admin user
`database`	Database name of the controller database
`user`	Database user name of the controller database
`password`	Encrypted password of the controller database user
`port`	Port number on which the database server of the controller database listens for requests
`host`	IP address of the host running the database server of the controller database
`type`	Database type of the controller database such as oracle or enterprisedb

Replication Server creates the content of this file as follows:

- The replication configuration file and some of its initial content are created when you install a publication server or subscription server on a host during the Replication Server installation process.
- Parameters `admin_user` and `admin_password` are determined during the Replication Server installation process. See Installation and uninstallation for how the content of these parameters are determined.
- Parameters `database`, `user`, `password`, `port`, `host`, and `type` are set with the connection and authentication information of the first publication database definition you create with the Replication Server console or CLI. This database is designated as the controller database (see Controller database). See Adding a publication database for creating a publication database definition for a single-master replication system. See Adding the primary definition node for creating the publication database definition for a multi-master replication system.

The following is an example of the content of an EPRS Replication Configuration file:

```
#xDB Replication Server Configuration
Properties
#Tue May 26 13:45:37 GMT-05:00
2015
port=1521
admin_password=ygJ9AxoJEX854elcVIJPTw\=\=
user=pubuser
admin_user=admin
type=oracle
password=ygJ9AxoJEX854elcVIJPTw\=\=
database=xe
host=192.168.2.23
```

> **Note**
>
> The passwords for the admin user name and the controller database user name are encrypted. If you change either of these passwords, you must modify the corresponding password parameters in the replication configuration file to contain the encrypted form of the new password. See Encrypting the password in the replication configuration file for directions on how to generate the encrypted form of a password.

See Installation details for the file system location of the EPRS Replication Configuration file.

Replication Server startup configuration file

The Replication Server startup configuration file contains installation and configuration information primarily used by the Java Runtime Environment (JRE) when any publication server or subscription server is started up on the host containing the file.

The content of the file is created by the Replication Server installer when you install Replication Server.

The following is an example of the content of an Replication Server startup configuration file:

```
#!/bin/sh
JAVA_EXECUTABLE_PATH="/usr/bin/java"
JAVA_MINIMUM_VERSION=1.8
JAVA_BITNESS_REQUIRED=64
JAVA_HEAP_SIZE="-Xms2048m -
Xmx4096m"
PUBPORT=9051
SUBPORT=9052
```

The following table contains a brief description of the parameters in the Replication Server startup configuration file.

Parameter	Description
`JAVA_EXECUTABLE_PATH`	Directory path to the Java runtime program used to start and run the publication and subscription servers.
`JAVA_MINIMUM_VERSION`	The earliest JRE version that can be used by the publication and subscription servers.
`JAVA_BITNESS_REQUIRED`	The bitness of the Java virtual machine required by the installed publication and subscription servers.
`JAVA_HEAP_SIZE`	In `-Xmsnnnm` `nnn` specifies the minimum Java heap size in megabytes. In `-Xmxnnnm` `nnn` specifies the maximum Java heap size in megabytes
`PUBPORT`	Port number on which the publication server listens for requests.
`SUBPORT`	Port number on which the subscription server listens for requests.

The `JAVA_EXECUTABLE_PATH` parameter specifies the location of the Java runtime program as identified by the Replication Server installer during the installation process. You can change the setting of this parameter to a different JRE installation if you want.

`The JAVA_MINIMUM_VERSION` parameter specifies the earliest version of the Java Runtime Environment that can be used with Replication Server. Don't change this setting.

Don't change the `JAVA_BITNESS_REQUIRED` parameter. If you change the installed value or if it doesn't match the bitness of the Java virtual machine as identified by `JAVA_EXECUTABLE_PATH`, errors can occur. These errors include failure of the publication and subscription servers to start and registration failure of the Replication Server product.

See Setting heap memory size for the publication and subscription servers for information on setting the `JAVA_HEAP_SIZE` parameter.

See Firewalls and access to ports for information on the `PUBPORT` and `SUBPORT` parameters.

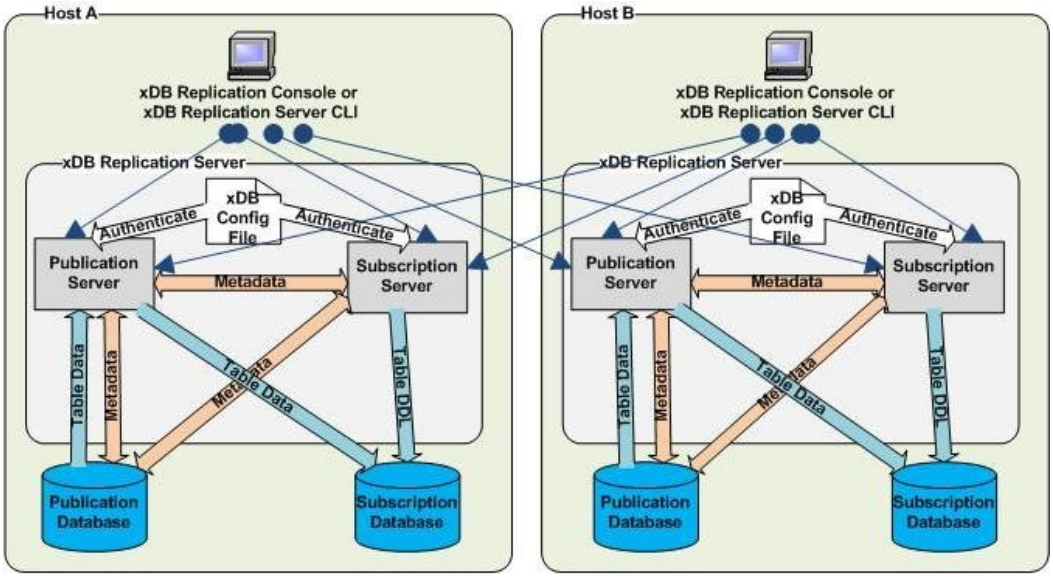
After making any modifications to the Replication Server startup configuration file, restart the publication and subscription servers.

See Installation details for the file system location of the Replication Server configuration file.

Replication Server console

The Replication Server console is the user interface you can use to create and control all aspects of a replication system.

Through this console, you can configure and operate a replication system running on the same host on which the Replication Server console is installed. Or, you can configure and operate replication systems where the Replication Server components are distributed on different hosts in a networked environment.



In the figure, two Postgres installations are running on two networked hosts, each with its own Replication Server installation. Each host is running a publication server and a subscription server.

The Replication Server console on each host can access and manage the replication systems on the other host if given the network IP address, port number, user name, and password with which the publication server and subscription server were installed on the remote host. See Introduction to the Replication Server console for information on the console user interface.

Replication Server command line interface

Replication Server command line interface (CLI) is a command-line-driven alternative to the Replication Server console user interface. The CLI provides equivalent functionality for creating and controlling all aspects of a replication system.

You can automate replication system operations by embedding Replication Server CLI commands in scripts such as Bash for Linux.

Replication Server CLI is installed whenever you install the Replication Server console.

Replication Server command line interface provides instructions for using Replication Server CLI.

Publication database

The publication database contains the tables and views used in a publication. The publication database can run on the same host or on a different host from where the publication server is running as long as the hosts can access each other by a network.

Each publication database also contains a control schema, which is a collection of database objects containing metadata on all replication systems, both single-master and multi-master, controlled by the publication server connected to this publication database. See Control schema and control schema objects for information on the control schema.

In a multi-master replication system, all primary nodes are considered publication databases.

A database plays the roles of both a publication database and a subscription database if it contains publications and subscriptions.

Subscription database

> **Note**
>
> The subscription database applies only to single-master replication systems.

The subscription database contains the tables created from a subscription. The subscription database can run on the same host or on a different host from where the subscription server is running as long as the hosts can access each other by a network.

A subscription database can also serve as a publication source for replicating to a third server if you want. This configuration is referred to as cascading replication.

A database plays the roles of both a publication database and a subscription database if it contains publications and subscriptions such as in the cascaded replication scenario.

Primary node

In a multi-master replication system, the databases containing the set of tables (the publication) for which row changes are to be replicated are called primary nodes. The primary nodes can run on the same host or on different hosts from where the publication server is running as long as the hosts can access each other by a network.

Each primary node also contains a control schema, which is a collection of database objects containing metadata on all replication systems, both single-master and multi-master, controlled by the publication server connected to this primary node. See Control schema and control schema objects for information on the control schema. The primary nodes can run under the same or under multiple database server instances (Postgres database clusters).

Primary definition node

The first node added to create a multi-master replication system is initially designated the primary definition node. This node must contain the table definitions (and optionally, the initial set of rows) to include in the publication.

As databases are added as primary nodes to the replication system, the table definitions and initial row sets can optionally be propagated from the primary definition node to the newly added primary nodes.

After the multi-master replication system is defined, it is possible to reassign the role of the primary definition node to another primary node in the multi-master replication system. The significance of this reassignment is that you can take snapshots from the newly appointed primary definition node to other primary nodes. This might help if the data in the old primary definition node becomes corrupt or out of sync with the other primary nodes and needs to be completely refreshed by a snapshot from another primary node.

As with all primary nodes, the primary definition node contains a control schema, which is a collection of database objects containing metadata on all replication systems, both single-master and multi-master, controlled by the publication server connected to this primary node.

Control schema and control schema objects

The control schema is a conceptual term referring to the collection of metadata database objects that define the logical and physical structure of and

enable the operation and maintenance of Replication Server single-master and multi-master replication systems.

These metadata database objects, referred to as control schema objects, consist of tables, sequences, functions, procedures, triggers, packages, and so on.

The control schema objects store metadata such as:

- Type of replication system (single-master or multi-master)
- Network location
- Database type
- Connection and authentication information for publication databases
- Subscription databases and the primary nodes, names of publications and the tables and views they contain,
- Names of subscriptions and the publications to which they are subscribed
- Replication transaction status
- Replication scheduling
- Replication history cleanup scheduling
- Replication history

Each publication database in a trigger-based, single-master replication system also contains control schema objects with the changes that were made to rows in the publication and the status of whether those changes were applied to the subscription tables.

Similarly, for a multi-master replication system, each trigger-based primary node contains control schema objects with the changes made to rows in the publication residing on that primary node. They also contain the statuses of whether those changes were applied to the other primary nodes in the multi-master replication system.

> **Note**
>
> For log-based single-master and multi-master replication systems, changes are extracted from the database server `WAL` files instead of being stored in control schema objects. See Synchronization replication with the log-based method for information on the log-based method.

The actual, physical database schemas implementing the control schema to which the control schema objects belong varies depending on the database type (Oracle, SQL Server, or Postgres) and how the database was first configured for use by Replication Server.

Note the followoing about the control schema:

- The control schema and its control schema objects are created in every publication database of both single-master and multi-master replication systems. This includes all master (publication) databases of single-master replication systems and all primary nodes of multi-master replication systems.
- When a new primary database is added for a single-master replication system or a new primary node for a multi-master replication system, a snapshot operation is used to replicate the control schema to the newly added publication database, assuming there is an existing controller database. See Controller database for information about the controller database.
- Updates to the configuration of a single-master replication system or a multi-master replication system made by the Replication Server console or the Replication Server command line interface are synchronized between the control schemas on all publication databases to ensure that the metadata is consistent across all publication databases.
- The secondary (subscription) database of single-master replication systems contains one, single table as its metadata database object. The term subscription metadata object specifically refers to this database object in the subscription database. The general terms control schema and control schema objects refer to the database objects in the publication databases.
- The control schema objects in all databases controlled by the same given publication server generally contain the same information. This configuration allows any such database to provide the information needed by Replication Server to control all single-master and multi-master replication systems running under that publication server.
- If a certain publication database of a replication system goes offline due to database server problems, network connectivity issues, and so on, the other replication systems running under the same publication server are still functional. The other publication databases can provide the control schema information needed to run all replication systems.

Controller database

In the Replication Server configuration file, the connection and authentication information for one publication database is included and, as such, designated as the controller database.

As with all publication databases, the controller database contains the control schema with the replication system information for all single-master and

multi-master replication systems run by the publication server that accesses that Replication Server configuration file.

The controller database serves as the primary provider of the replication system information to the publication server and the subscription server. Thus, upon initial startup, the publication server and subscription server try to connect to the designated controller database. This controller database then provides the metadata information for all replication systems.

If the initial connection to the controller database fails, you can manually edit the Replication Server configuration file to provide the connection and authentication information for another publication database. Then, upon startup of the publication server and subscription server, the control schema of this alternate publication database is used to provide the replication system information.

The initial controller database is determined by the first publication database definition created by the Replication Server console or the Replication Server CLI either for a single-master or multi-master replication system. The publication server records the connection and authentication information in the Replication Server configuration file.

If you want to delete the publication database definition of the current controller database, you must first designate another publication database, defined under the same publication server, as the controller using the Replication Server console. See Switching the controller database for information on switching the controller to another publication database.

The following are some points regarding the controller database:

- The database server running the controller database must be running and accessible before starting the publication server and subscription server.
- For a single-master replication system, consider:
    - The publication server under which the publication database and publication are defined
    - The subscription server under which the subscription database and the subscription related to the publication are defined

  Both must connect to the same the controller database. This configuration gives the publication server and the subscription server access to the same control schema.
- When changes are made to the metadata maintained by the control schema in the controller database, these changes are replicated by the publication server to the control schemas of all other publication databases. This setup ensures that the metadata of all single-master and multi-master systems are complete and consistent in the control schemas of all publication databases. This setup also allows you to switch the controller database later. See Switching the controller database.

> **Note**
>
> If the controller database is an Oracle or a SQL Server publication database, then you can't add a second Oracle or SQL Server publication database to create a second single-master replication system. For Replication Server to run more than one single-master replication system consisting of Oracle or SQL Server publication databases, you must designate a Postgres publication database as the controller database.

Once you have multiple Oracle or SQL Server publication databases set up in single-master replication systems with a Postgres controller database, don't switch the controller database to an Oracle or SQL Server publication database.

4.3.2 Logical components

Logical components are created when you build a replication system using the Replication Server console or the Replication Server CLI. The logical components are stored as part of the replication system metadata in the control schema of the publication databases.

To create a replication system, you must:

- Register a publication server.
- Create a publication database definition.
- Create a publication.

For a single-master replication system, you then must:

- Register a subscription server.
- Create a subscription database definition.
- Create a subscription.

For a multi-master replication system, you create additional primary nodes by creating additional publication database definitions.

Each of these steps creates a logical component that is represented by a node in the replication tree of the Replication Server console. SeeIntroduction to the Replication Server Console for a detailed description of the Replication Server console.

Publication server

The first step in creating a publication is to identify the publication server to use to manage the publication. This process is called registering the publication server.

Using the Replication Server console or the Replication Server CLI, register a publication server by giving the IP address and port number of the host on which the publication server is running. Also give the admin user name and password stored in the Replication Server configuration file located on the host running the publication server. (This information is determined during the publication server installation process.)

When viewed in the Replication Server console, a registered publication server appears under the top level Replication Servers node in the replication tree. All publication-related logical components are created subordinate to a registered publication server and appear underneath it in the replication tree.

Registering a publication server gives instructions for registering a publication server for a single-master replication system. See Registering a publication server for a multi-master replication system.

Replication system type (SMR/MMR)

Subordinate to a registered publication server, two nodes representing the replication system type appear. One is identified by the label SMR for single-master replication and the other has the label MMR for multi-master replication.

If you're creating a single-master replication system, you proceed to add logical components under the SMR type node.

If you're creating a multi-master replication system, you proceed to add logical components under the MMR type node.

Publication database definition

You can create one or more publication database definitions subordinate to one of the Replication System Type nodes under a registered publication server.

A publication database definition identifies a database whose tables and views to use in a publication. The identifying information consists of the database server IP address, port number, a database user name and password, and the database identifier.

The publication server uses this information to connect to the publication database to create the replication system control schema in the publication database and perform the replications.

Though the process of creating a publication database definition is similar for single-master and multi-master replication systems, their usage in the replication system is somewhat different.

In a single-master replication system, a publication database definition identifies the storage area of one or more publications, each of which is eventually associated with its own subscription in a primary-to-secondary relationship.

In a multi-master replication system, each publication database definition subordinate to the MMR type node of a given publication server identifies a primary node in a single, multi-master replication system.

> **Note**
>
> Currently, you can have only one multi-master replication system per publication server.

Adding a publication catabase discusses creating a publication database definition for a single-master replication system. See Adding the primary definition node and Creating additional primary nodes for a multi-master replication system.

Publication

You can define one or more publicaitons subordinate to a publication database definition in a single-master replication system. A publication contains a list of tables and views to replicate to a subscription database.

In a single-master replication system, the database user name specified in the publication database definition of the publication's parent, as viewed in the replication tree, must have the `SELECT` object privilege on any table or view to include in the publication.

You can define only one publication subordinate to a publication database definition in a multi-master replication system. The publication contains the list of tables to replicate and keep synchronized in the primary nodes of the multi-master replication system.

In a multi-master replication system, the database user name specified in the publication database definition of the publication's parent, as viewed in the replication tree, must have superuser privileges and be the owner of all tables to be included in the publication.

Adding a publication discusses creating a publication for a single-master replication system. See Adding a publication for a multi-master replication system.

Subscription server

> **Note**
>
> The subscription server applies only to single-master replication systems. You don't register a subscription server when creating a multi-master replication system.

The first step in creating a subscription is to identify the subscription server to use to manage the subscription. This process is called registering the subscription server.

Using the Replication Server console or the Replication Server CLI, register a subscription server by giving the IP address and port number of the host on which the subscription server is running. Also give the admin user name and password stored in the Replication Server configuration file located on the host running the subscription server. This information is determined during the subscription server installation process.

When viewed in the Replication Server console, a registered subscription server appears under the top-level Replication Servers node in the replication tree. All subscription-related logical components are created subordinate to a registered subscription server and appear underneath it in the replication tree. Registering a subscription server gives detailed instructions.

Subscription database definition

> **Note**
>
> The subscription database definition applies only to single-master replication systems. You don't create a subscription database definition when

> creating a multi-master replication system.

Create one or more subscription database definitions subordinate to a registered subscription server.

A subscription database definition identifies a database to which to replicate a publication's tables and views. The identify information consists of:

- The database server IP address
- Port number
- A database user name and password
- the database identifier

The subscription server uses this information to connect to the subscription database to create the table definitions.

The publication server also uses this information to connect to the subscription database when it performs replications.

Adding a subscription database discusses creating a subscription database definition.

Subscription

> **Note**
>
> The subscription applies only to single-master replication systems. You don't create a subscription when creating a multi-master replication system.

You can define one or more subscriptions subordinate to a subscription database definition. A subscription associates a publication to a subscription database to which to replicate the publication's tables and views.

You can associate each subscription with only one publication.

For details, see Adding a subscription.

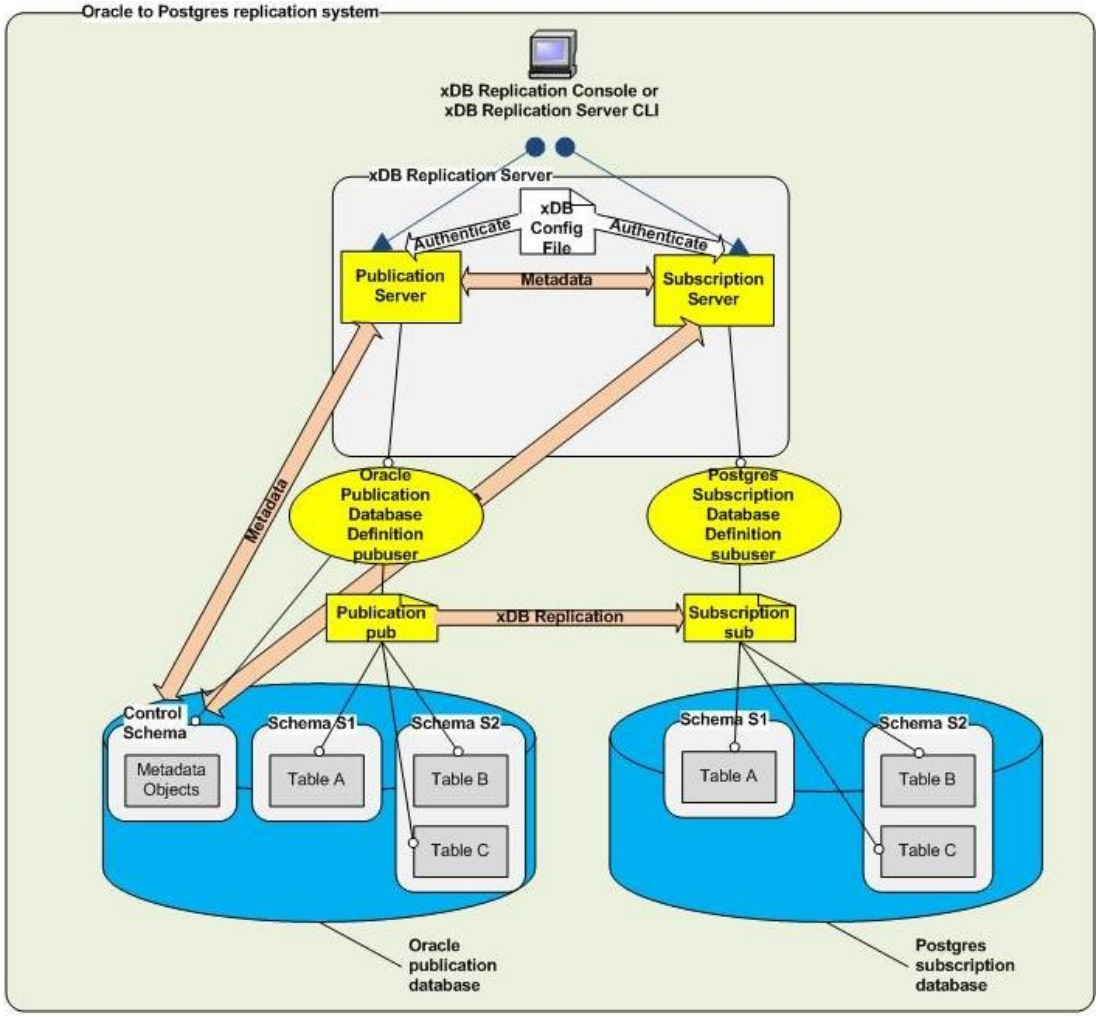
4.3.3 Replication Server system examples

In the diagrams that follow, the logical components, represented by nodes in the replication tree of the Replication Server console, are superimposed on physical component diagrams. The logical components are shaded yellow to help to identify them in the diagrams.

See Introduction to the Replication Server console for more information about the Replication Server console.

Oracle to PostgreSQL or EDB Postgres Advanced Server replication

The following shows a basic Oracle to PostgreSQL or EDB Postgres Advanced Server single-master replication system. A single publication in Oracle contains tables from two schemas that are replicated to a database residing in either PostgreSQL or EDB Postgres Advanced Server.

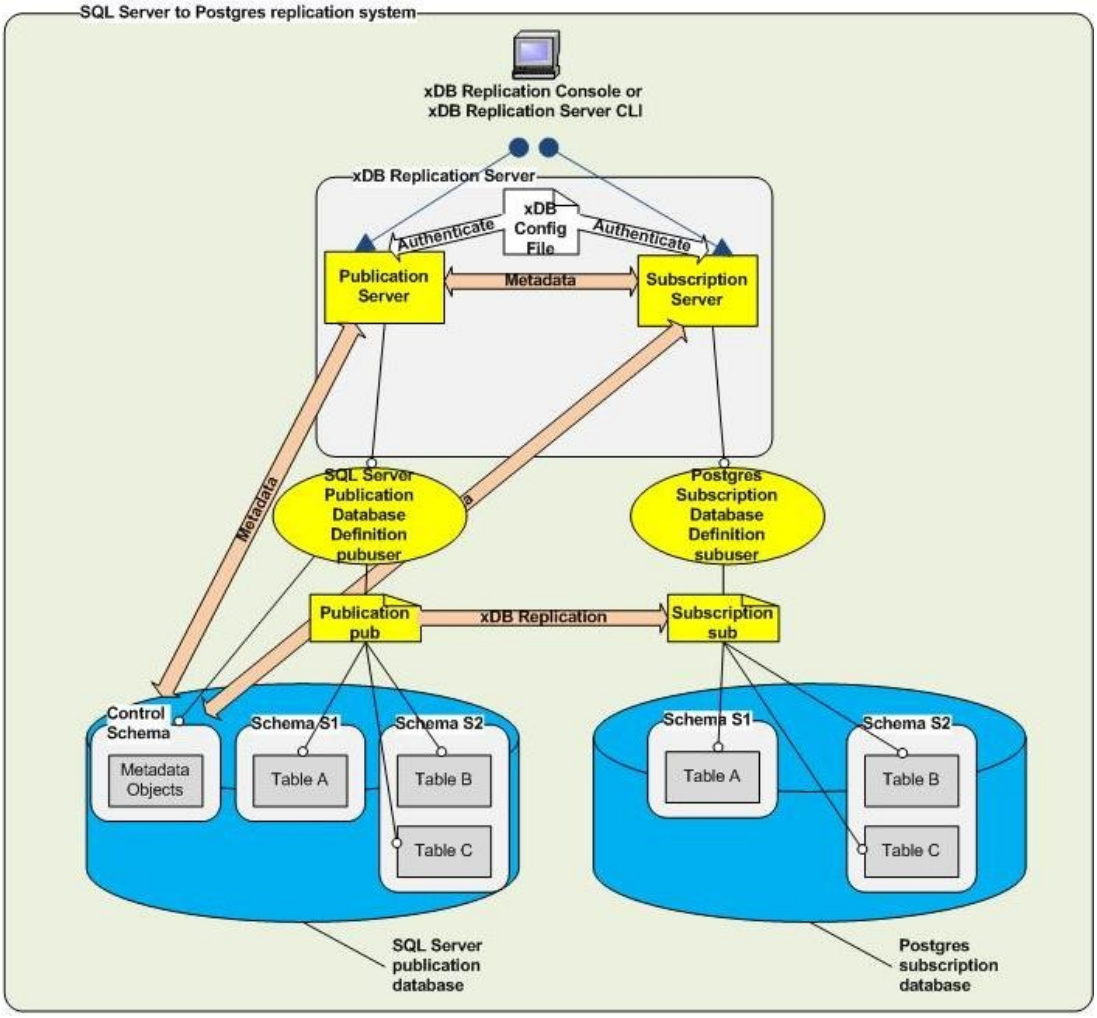


The following describes the logical components in the preceding diagram: * The publication server to use is identified by registering its network location, user name, and password.

- A publication database definition is created subordinate to the SMR type node under the publication server. The Oracle database user name pubuser is specified in the definition along with the database network location and database identifier. When you create a user named pubuser in Oracle, a schema named `pubuser` is automatically created by Oracle at the same time. The publication server creates the control schema objects in the `pubuser` control schema for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2` .
- The subscription server to use is identified by registering its network location, user name, and password.
- A subscription database definition is created subordinate to the subscription server. The Postgres database user name subuser is specified in the definition along with the database network location and database identifier.
- A subscription named `sub` is created subordinate to the subscription database definition. When the subscription is created, the subscription server creates schemas named `S1` and `S2` in the subscription database. The table definitions for tables `A, B` , and `C` are also created at this time. When replication occurs, the publication server populates these tables with rows from the publication.

SQL Server to PostgreSQL or EDB Postgres Advanced Server replication

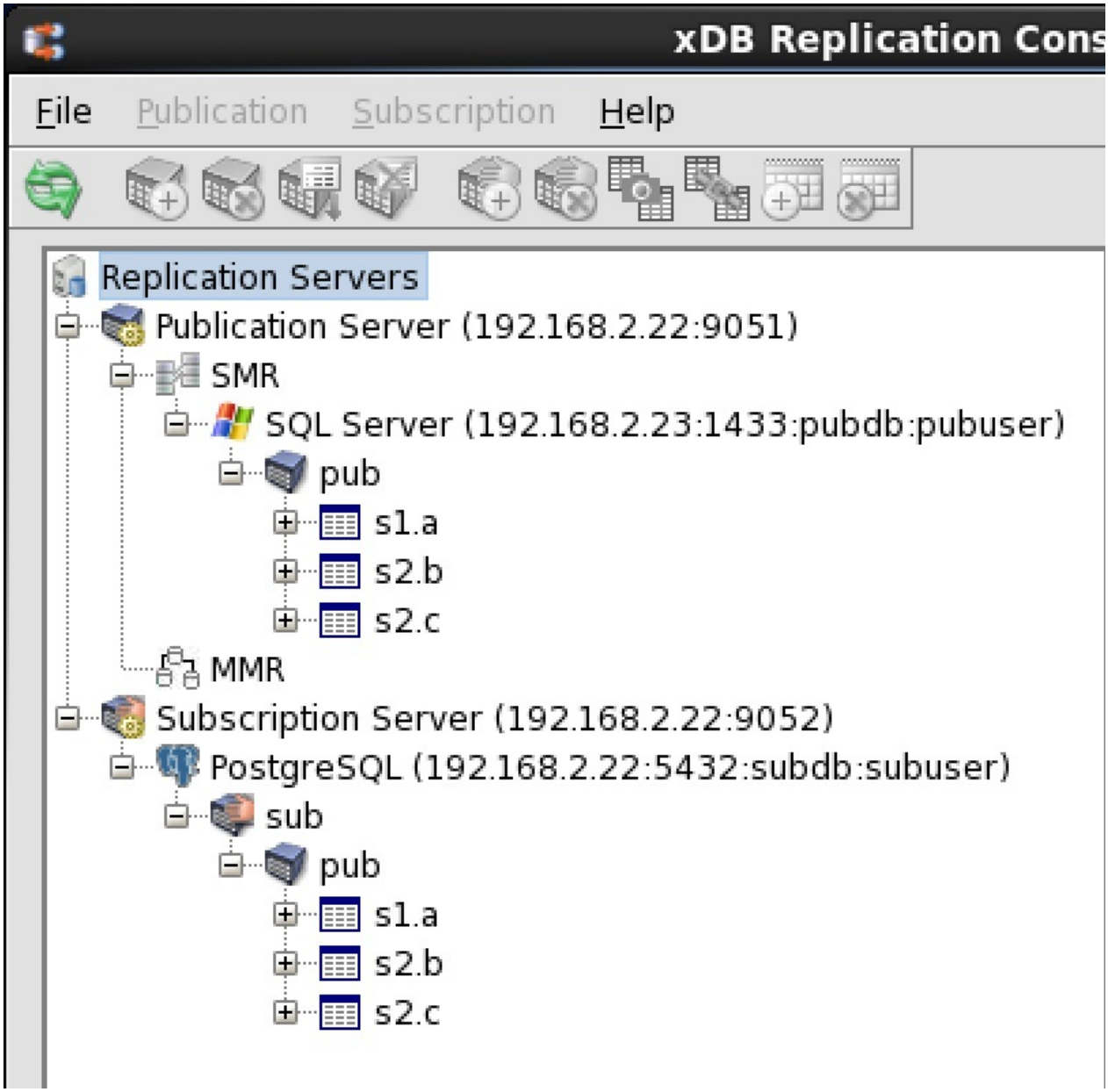
The following shows a basic SQL Server to PostgreSQL or EDB Postgres Advanced Server single-master replication system. A single publication in SQL Server contains tables from two schemas that are replicated to a database residing in either PostgreSQL or EDB Postgres Advanced Server.



The following describes the logical components in the diagram:

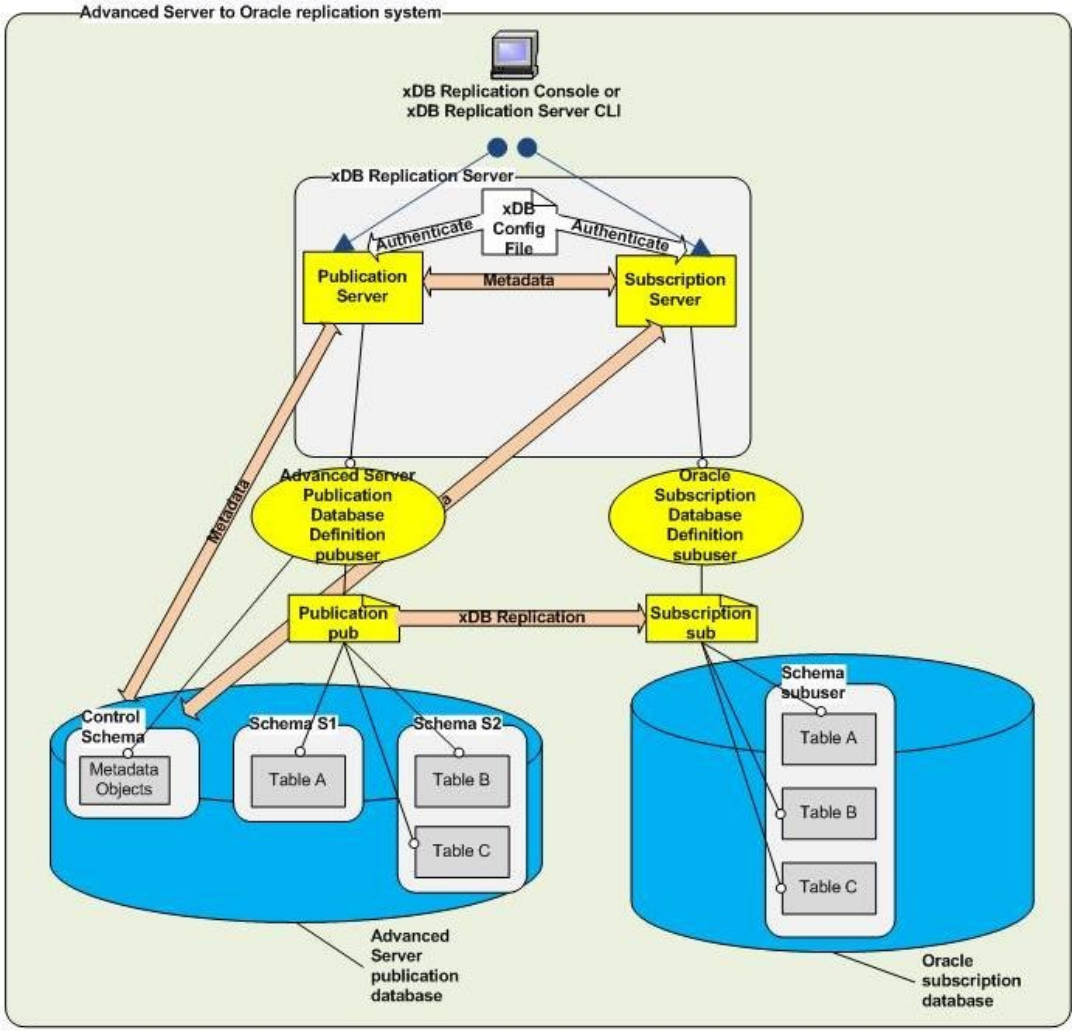
- The publication server to use is identified by registering its network location, user name, and password.
- A publication database definition is created subordinate to the SMR type node under the publication server. The SQL Server login pubuser is specified in the definition along with the database network location and database identifier. The schema `pubuser` was created during the publication database preparation step as described in SQL Server publication database. The `pubuser` schema along with the control schema consisting of three physical schemas `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler` are populated with the control schema objects for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2`.
- The subscription server to use is identified by registering its network location, user name, and password.
- A subscription database definition is created subordinate to the subscription server. The Postgres database user name `subuser` is specified in the definition along with the database network location and database identifier.
- A subscription named sub is created subordinate to the subscription database definition. When the subscription is created, the subscription server creates schemas named `S1` and `S2` in the subscription database. The table definitions for tables `A, B`, and `C` are also created at this time. When replication occurs, the publication server populates these tables with rows from the publication.

The following image shows how the logical components of this replication system appear in the Replication Server console replication tree.



EDB Postgres Advanced Server to Oracle replication

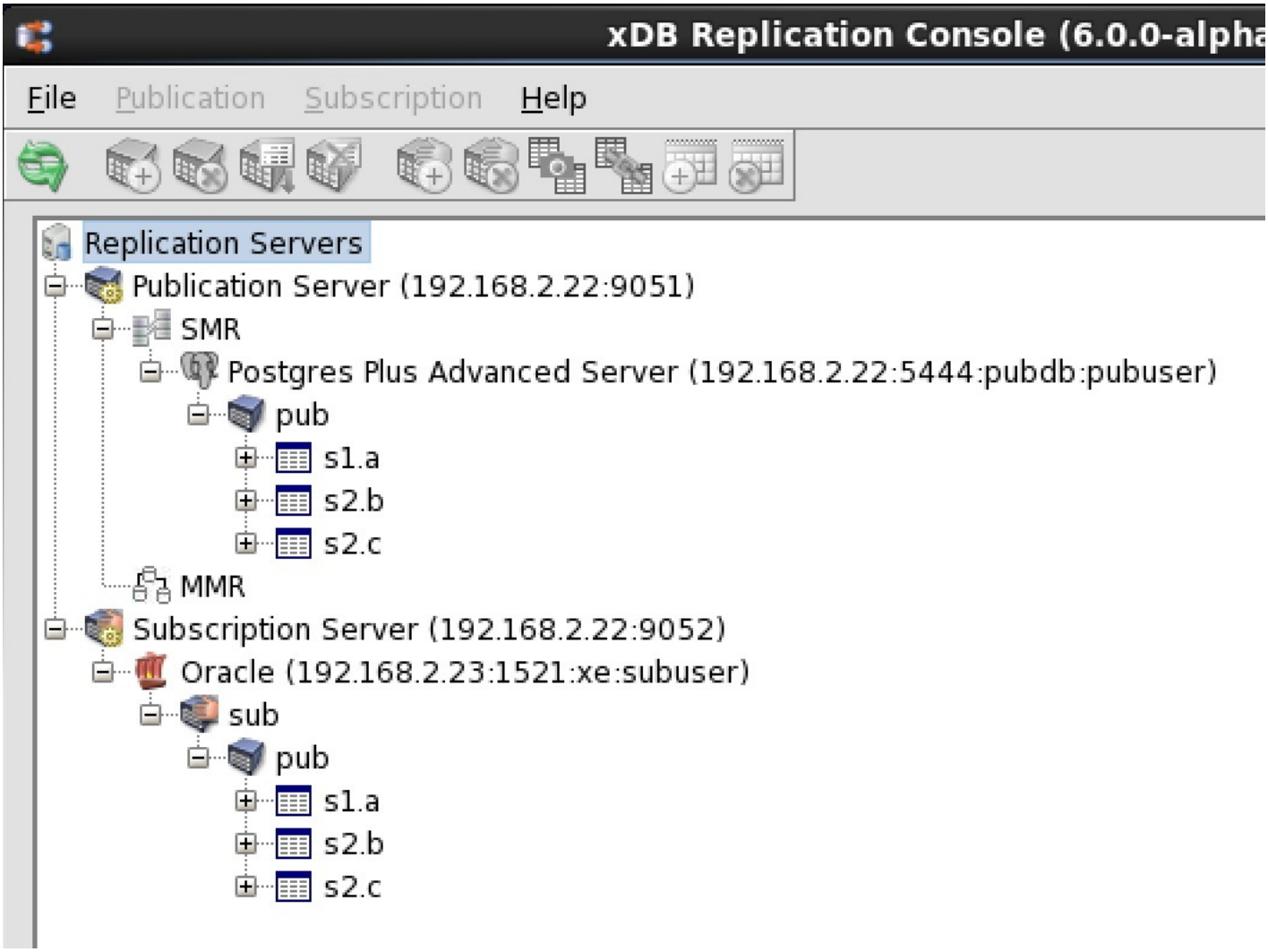
The following shows a basic EDB Postgres Advanced Server to Oracle single-master replication system. A single publication in a EDB Postgres Advanced Server database contains tables from two schema that are replicated to an Oracle database.



The following describes the logical components in the diagram:

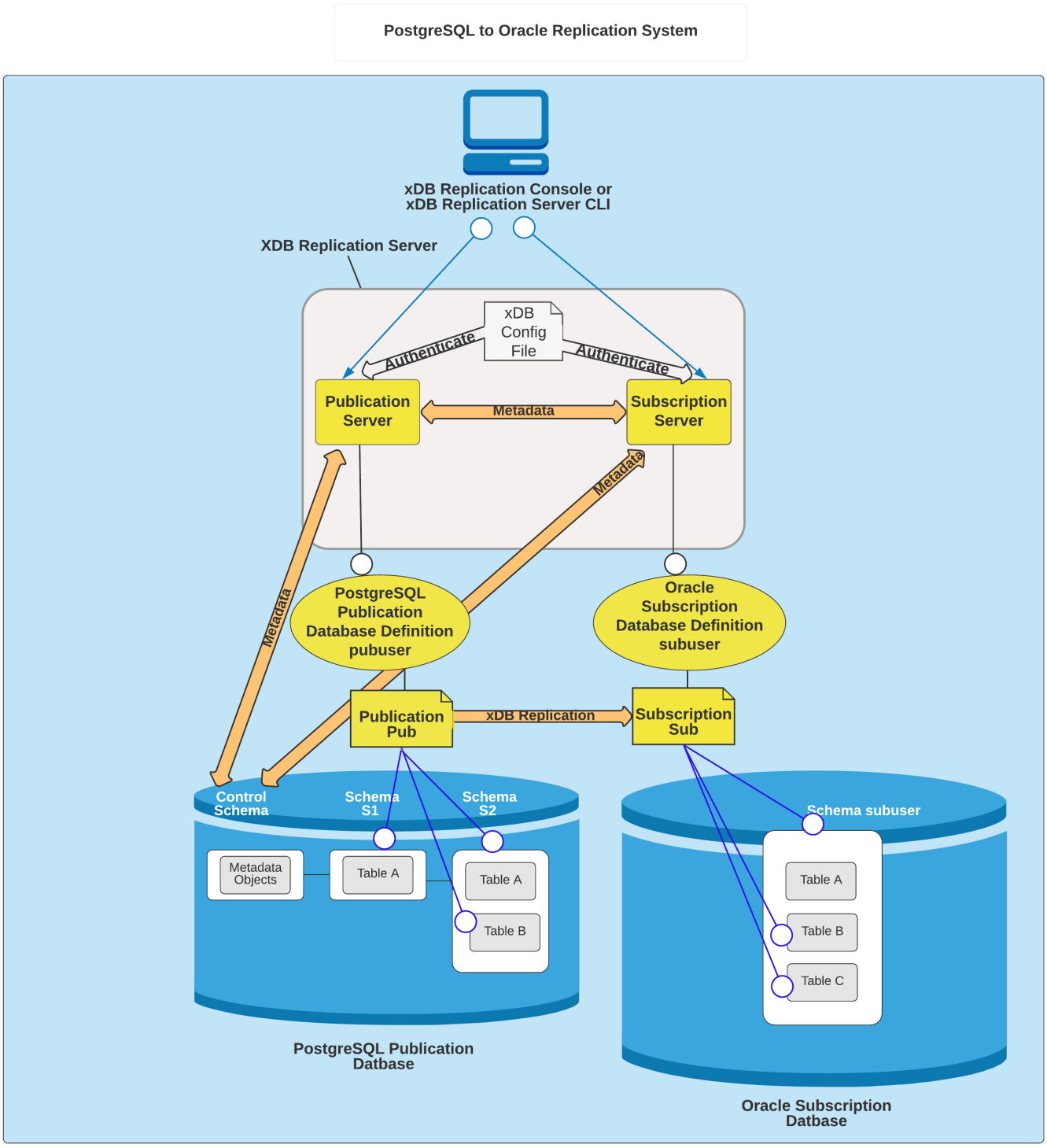
- The publication server to use is identified by registering its network location, user name, and password.
- A publication database definition is created subordinate to the SMR type node under the publication server. The Postgres database user name pubuser is specified in the definition along with the database network location and database identifier. The publication server creates the control schema consisting of three physical schemas `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler` and populates them with the control schema objects for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2`.
- The subscription server to use is identified by registering its network location, user name, and password.
- A subscription database definition is created subordinate to the subscription server. The Oracle database user name `subuser` is specified in the definition along with the database network location and database identifier.
- A subscription named sub is created subordinate to the subscription database definition. When you create a user named subuser in Oracle, a schema named `subuser` is automatically created by Oracle at the same time. The table definitions for tables `A`, `B`, and `C` are created in schema `subuser` when you create subscription sub. When replication occurs, the publication server populates these tables with rows from the publication.

The following image shows how the logical components of this replication system appear in the Replication Server console replication tree.



PostgreSQL to Oracle replication

The following shows a basic PostgreSQL to Oracle single-master replication system. A single publication in a PostgreSQL database contains tables from two schemas that are replicated to an Oracle database. WAL-based method as well as trigger-based method is supported in this type of replication.

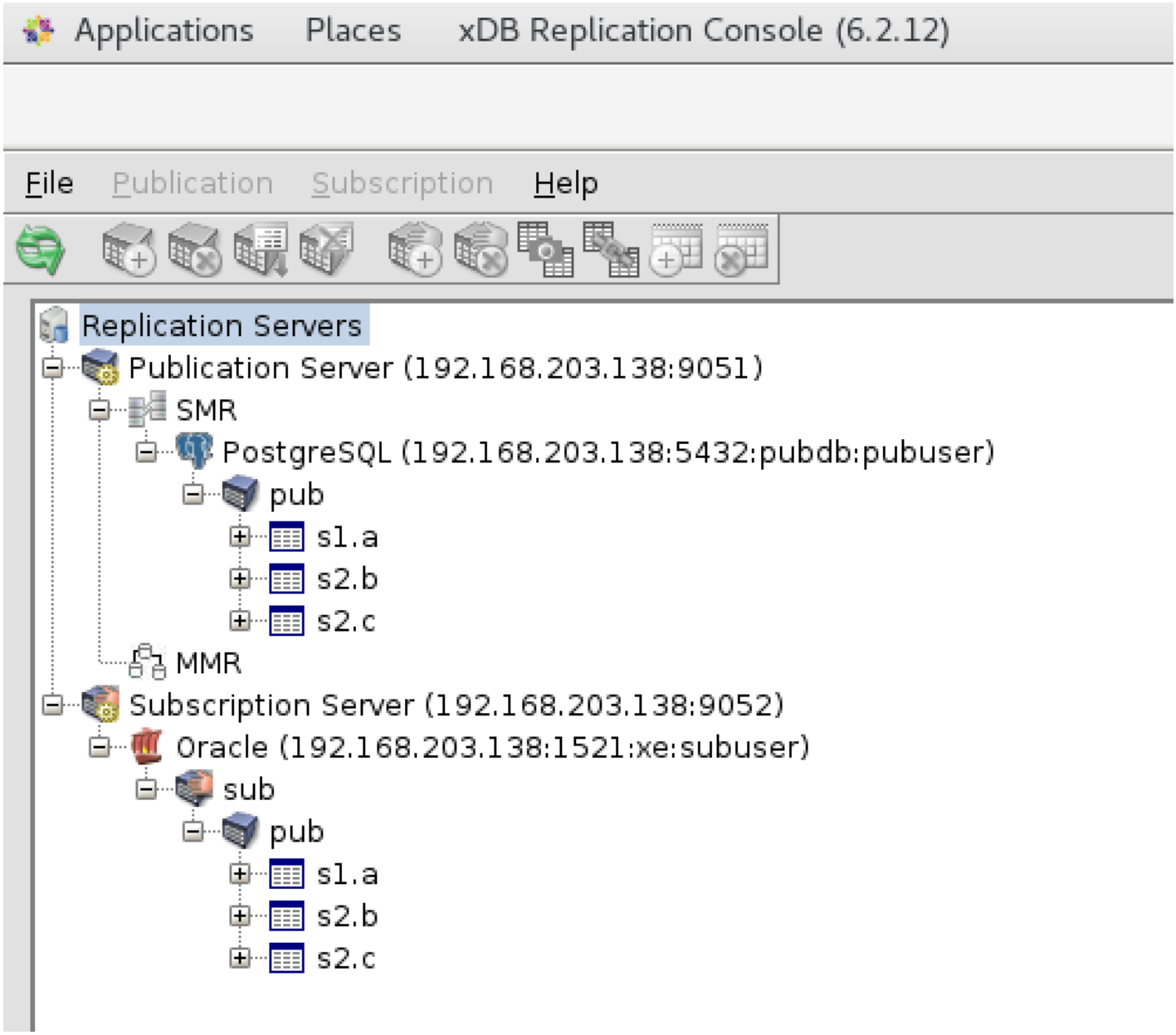


The following describes the logical components in the diagram:

- The publication server to use is identified by registering its network location, user name, and password.
- A publication database definition is created subordinate to the SMR type node under the publication server. The Postgres database user name pubuser is specified in the definition along with the database network location and database identifier. The publication server creates the control schema consisting of three physical schemas _edb_replicator_pub, _edb_replicator_sub, and _edb_scheduler and populates them with the control schema objects for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2`.
- The subscription server to use is identified by registering its network location, user name, and password.
- A subscription database definition is created subordinate to the subscription server. The Oracle database user name subuser is specified in the definition along with the database network location and database identifier.
- A subscription named sub is created subordinate to the subscription database definition. When you create a user named subuser in Oracle, a schema named `subuser` is automatically created by Oracle at the same time. The table definitions for tables `A`, `B`, and `C` are created in schema

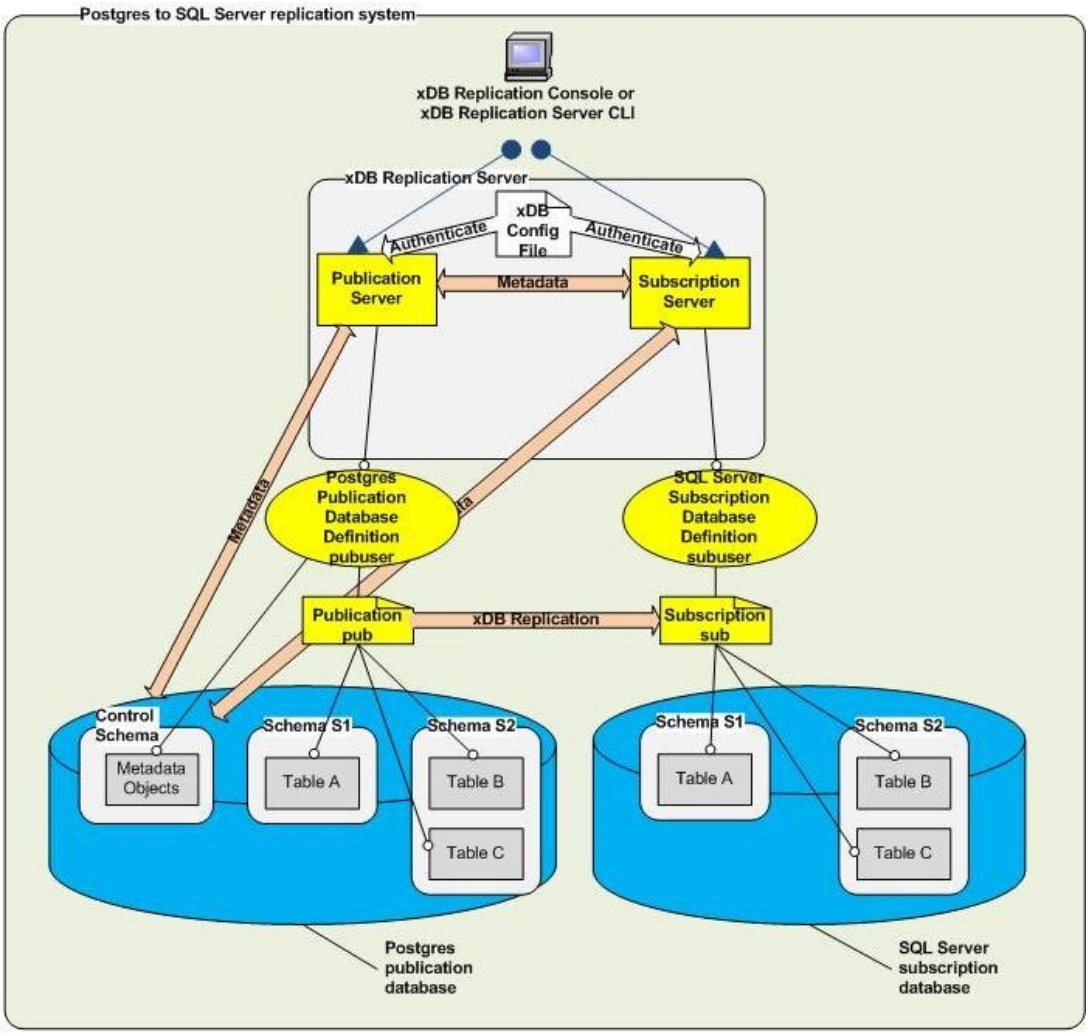
`subuser` when you create subscription sub. When replication occurs, the publication server populates these tables with rows from the publication.

The following image shows how the logical components of this replication system appear in the Replication Server console replication tree.



PostgreSQL or EDB Postgres Advanced Server to SQL Server replication

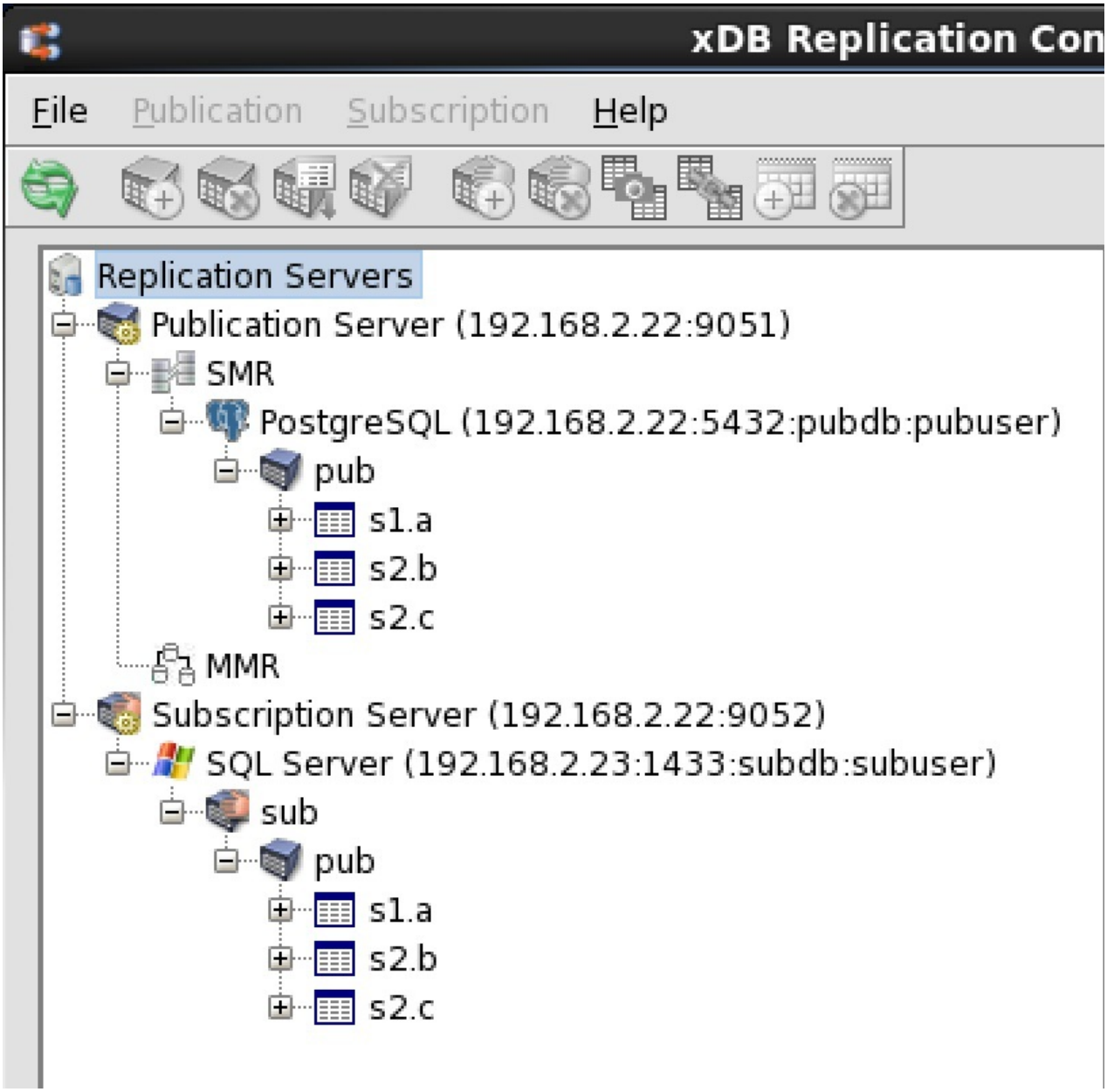
The following shows a basic PostgreSQL or EDB Postgres Advanced Server to SQL Server single-master replication system. A single publication in a PostgreSQL or EDB Postgres Advanced Server database contains tables from two schemas that are replicated to a SQL Server database.



The following describes the logical components in the diagram:

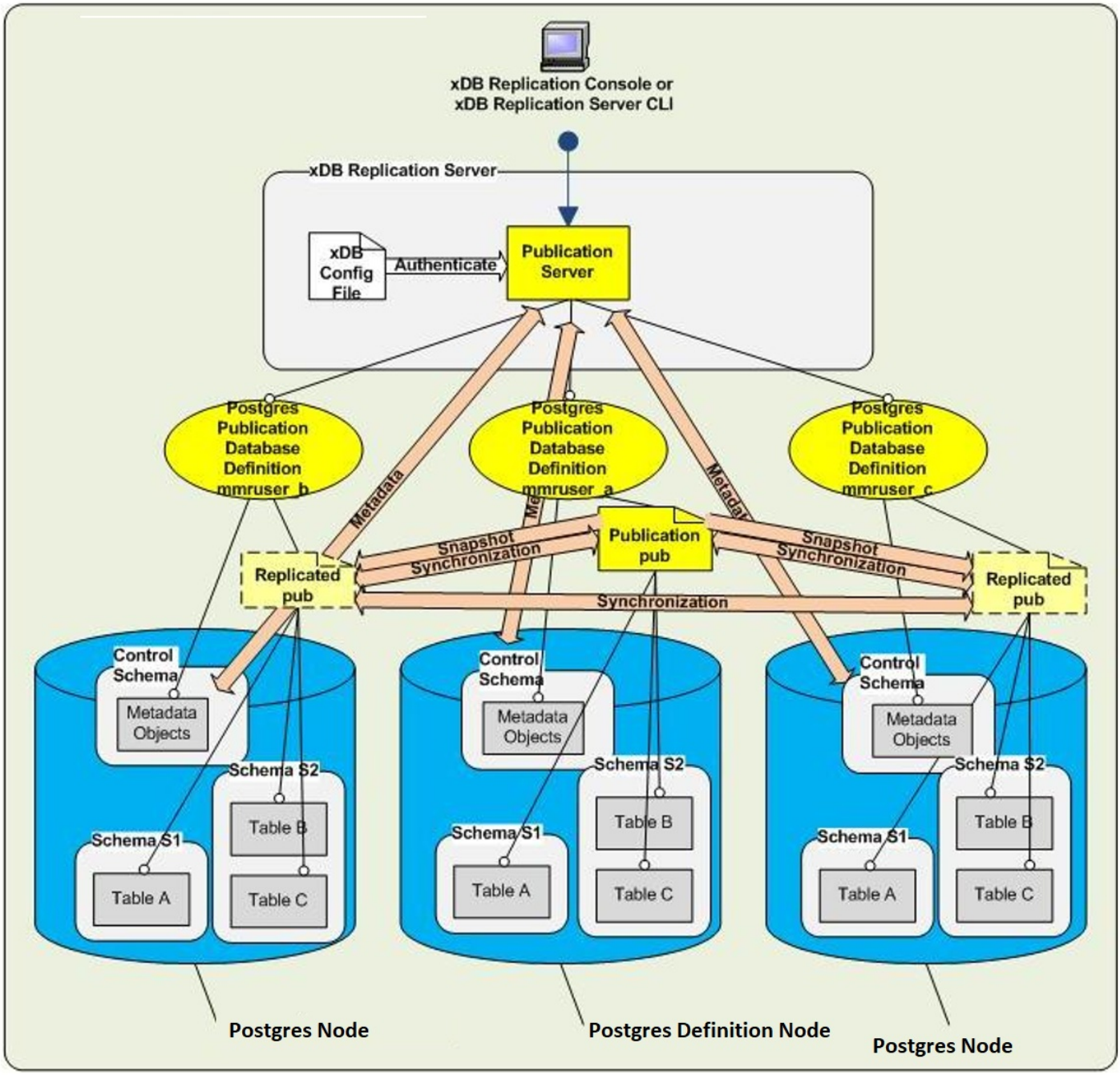
- The publication server to use is identified by registering its network location, user name, and password.
- A publication database definition is created subordinate to the SMR type node under the publication server. The Postgres database user name pubuser is specified in the definition along with the database network location and database identifier. The publication server creates the control schema consisting of three physical schemas `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler` and populates them with the control schema objects for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2`.
- The subscription server to use is identified by registering its network location, user name, and password.
- A subscription database definition is created subordinate to the subscription server. The SQL Server login `subuser` is specified in the definition along with the database network location and database identifier.
- A subscription named sub is created subordinate to the subscription database definition. When the subscription is created, the subscription server creates schemas named `S1` and `S2` in the subscription database. The table definitions for tables `A, B`, and `C` are also created at this time. When replication occurs, the publication server populates these tables with rows from the publication.

The following image shows how the logical components of this replication system appear in the Replication Server console replication tree.



Postgres multi-master replication

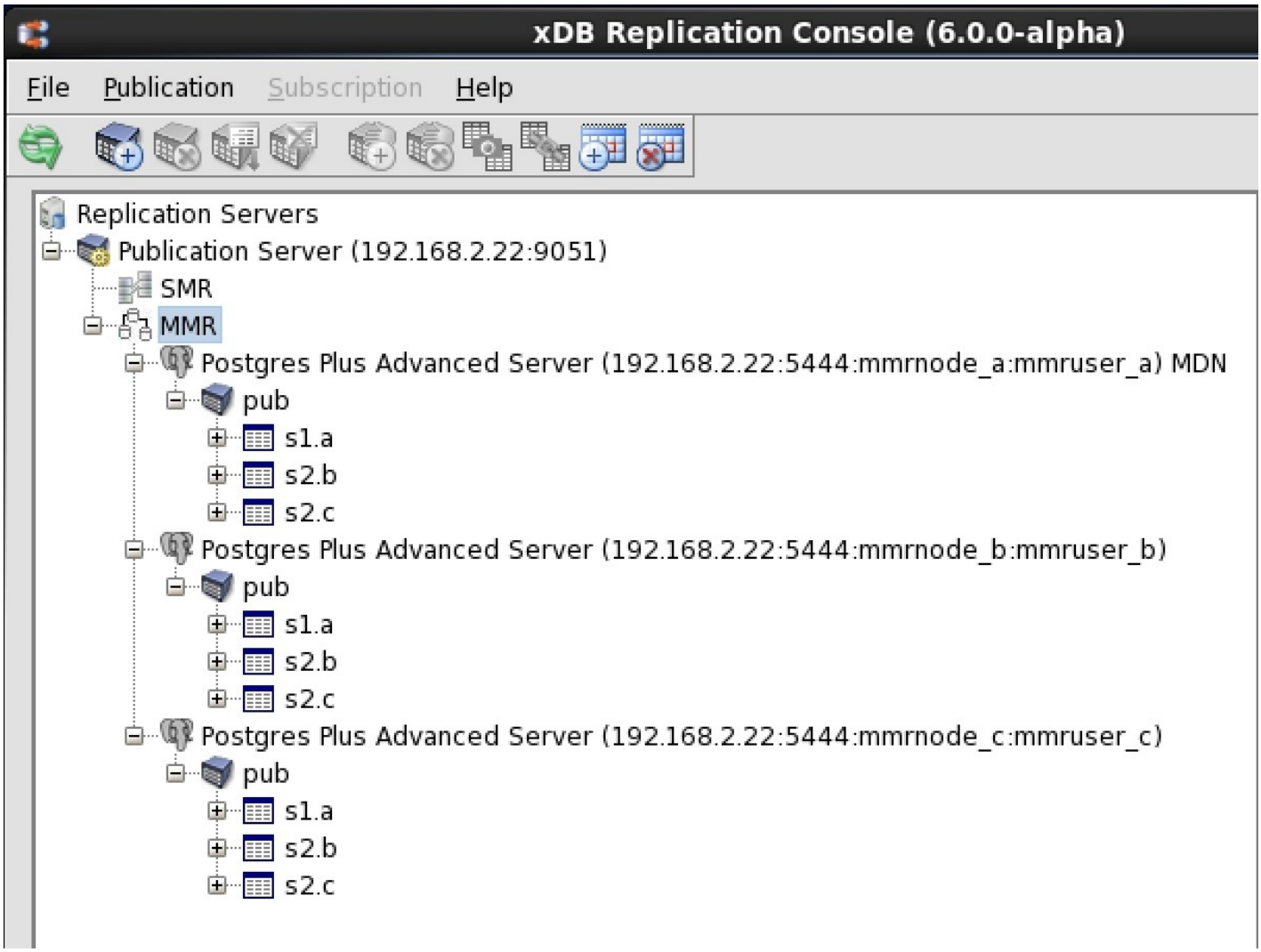
The following shows a basic Postgres multi-master replication system. A publication in a Postgres primary definition node contains tables from two schemas that are initially replicated to two other Postgres primary nodes. The tables in all three primary nodes can then be updated and synchronized with each other.



The following describes the logical components in the diagram:

- The publication server to use is identified by registering its network location, user name, and password.
- A publication database definition is created subordinate to the MMR type node under the publication server. This first publication database definition identifies the primary definition node. The Postgres database user name MMRuser_a is specified in the definition along with the database network location and database identifier. The publication server creates the control schema consisting of three physical schemas `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler` and populates them with the control schema objects for the replication system's metadata when you create the publication database definition.
- A publication named pub is created subordinate to the publication database definition. The publication consists of table `A` in schema `S1` and tables `B` and `C` in schema `S2`.
- A second primary node is added by creating another publication database definition subordinate to the MMR type node of the publication server under which the primary definition node resides. The Postgres database user name MMRuser_b is specified in the definition along with the database network location and database identifier to create the second primary node.
- When you add the second primary node, you can choose to have the publication server create schemas `S1` and `S2` and the table definitions for `A, B`, and `C` for you, or you can manually create the schemas and table definitions beforehand. The publication server creates the control schema consisting of three physical schemas `_edb_replicator_pub`, _edb_replicator_sub, and _edb_scheduler under which it creates the control schema objects to store the primary node's metadata. When defining the primary node, you can choose to have the publication server populate these tables with rows from the publication at that time, or you can defer table loading to later.
- A third primary node is added in a similar manner using the Postgres database user name MMRuser_c.

The following image shows how the logical components of this replication system appear in the Replication Server console replication tree.



4.4 Designing a replication system

Understand the general steps, design considerations, and best practices for designing a replication system before you begin the actual implementation.

4.4.1 General steps for implementing a replication system

When implementing a replication system, follow these general guidelines.

1. Determine if Replication Server is the right solution for your requirements and you have chosen the best solution for your particular needs. You can use Replication Server to implement single-master or multi-master replication systems. For single-master replication systems, the distinguishing characteristic of Replication Server is its ability to replicate:

    - From an Oracle database to a PostgreSQL or EDB Postgres Advanced Server database
    - From a SQL Server database to a PostgreSQL or EDB Postgres Advanced Server database
    - From an EDB Postgres Advanced Server database to an Oracle database

- From a PostgreSQL or EDB Postgres Advanced Server database to a SQL Server database

2. Plan the general strategy of how you will use Replication Server. Will the single-master or multi-master model best suit your needs? (SeeWhy use replication? for use case examples of single-master and multi-master replication systems.) Will you be replicating from Oracle to Postgres, from SQL Server to Postgres, from EDB Postgres Advanced Server to Oracle, or from Postgres to SQL Server? Will you be replicating between PostgreSQL and EDB Postgres Advanced Server databases? How often will you need to replicate the data? Will replication be done on an ad hoc basis or does it need to occur regularly according to a schedule?

3. Plan the logistics of your replication system. How many tables do you expect to replicate, and what are their sizes in total number of bytes and number of rows? What percentage of rows do you expect to change on each table between each replication? Are your database servers required to run on dedicated machines?

4. Design your replication system. Determine whether your replication system will be distributed or will run on a single host. Determine the publications and subscriptions you'll need and their tables and views. Make sure your publication tables meet the requirements for a Replication Server publication. See Restrictions on replicated database objects for details.

5. Implement and test your replication system in a test environment. Try out your replication system on a subset of your publication data to ensure the replication process works as expected. Make sure you can use the resulting replicated tables as expected in your application. Establish preliminary metrics on how long the replication process will take in your full production environment.

6. Implement and test your replication system in your production environment.

4.4.2 Design considerations

Keep the following points in mind when designing a replication system:

- Multi-master replication is supported only on PostgreSQL databases. In addition, EDB Postgres Advanced Server databases must run in the same compatibility mode—either all Oracle or all PostgreSQL.

- An Oracle table can be a member of at most one publication if all publications are subordinate to one publication database definition. However, an Oracle table can be a member of multiple publications if each publication is subordinate to a different publication database definition.

- A Postgres table can be a member of at most one publication.

- Each non-PostgreSQL/EDB Postgres Advanced Server database table used in a publication must have a primary key, with the exception of tables in snapshot-only publications. These tables don't require a primary key.

- Each PostgreSQL/EDB Postgres Advanced Server database table used in a publication must have identity columns (primary key or unique columns), with the exception of tables in snapshot-only publications. These tables don't require identity columns. For example:

```
CREATE TABLE dept_pk
(
   deptno NUMERIC(2) PRIMARY
KEY,
   dname VARCHAR(14)
   loc
VARCHAR(13)
);
```

If a table doesn't have a primary key, then it should have a column with `UNIQUE NOT NULL` constraints. For example:

```
CREATE TABLE dept_uk2
(
   deptno
NUMERIC(2),
   dname VARCHAR(14) NOT NULL,
```

```
   loc VARCHAR(13) NOT
NULL,
   CONSTRAINT dept_uk2_dname_loc_key UNIQUE (dname, loc)
);
```

- Make sure table definitions are well established before creating publications. Unless you use the DDL change replication feature, as described in Replicating DDL changes, if a table definition changes, you must delete and re-create any publication containing the table along with its associated subscription. Otherwise replication might fail. The same applies for the table definitions in a primary definition node and its associated primary nodes. You can see replication failures in the replication history.

- Views can be members of snapshot-only publications. The views are replicated as tables in the subscription databases.

- Materialized views can be members of snapshot-only publications from Oracle to EDB Postgres Advanced Server databases. The materialized views are replicated as tables in the subscription databases. The following items provide additional information pertinent to performing snapshot-only replication of materialized views:

  1. Use the `'tables'` parameter for specifying materialized views while creating a publication via EDB Replication Server CLI. While creating a snapshot-only publication via Replication Console, the view type currently appears as a table.

  2. We recommend that users create the views with auto-refresh in Oracle. If the views are not auto-refresh they are marked as invalid upon updating the base table until refreshed explicitly. EDB Replication Server currently does not support subscription creation or snapshot for invalid views.

     The following error appears during a snapshot where the view is invalid:

     ```
     INFO: MTK-14001: One or more tables are missing from the source Oracle
     database.
     ```

     With this error, the user must refresh the view before taking the snapshot.

     > **Note**
     >
     > Snapshot-only replication for materialized views is currently tested and certified only for Oracle to EDB Postgres Advanced Server databases.

- A publication can have multiple subscriptions.

- A subscription can be associated with at most one publication.

- A database can contain both publications and subscriptions.

- A given publication server can support only one multi-master replication system. All primary nodes created subordinate to a given publication server are assumed to be part of the same multi-master replication system.

- You can use a table that is created as a result of a subscription in another publication. Thus, a publication can replicate data to a subscription which, in turn, you can use in a publication to replicate to another subscription. This scenario creates a cascaded replication architecture.

- For restrictions on the combinations and configurations of database servers that you can use for a publication and its subscription, see EDB Postgres Advanced Server compatibility configuration modes.

- All replication system components must be running for replication to occur or before you configure, operate, or modify the replication system. Use the Replication Server console to configure and modify a replication system. The console doesn't need to be running for replication to occur.

- In general, the order of creating a replication system is as follows:

  1. Create the required physical databases, database user names, tables, and views to use in the replication system.
  2. Define the replication system logical components using the Replication Server console or CLI.
  3. Perform replication.

- In general, the order of removing a single-master replication system is as follows:

1. Remove the replication system logical components using the Replication Server console or CLI, starting with the subscriptions (Subscription nodes) and then their parent components (Subscription Database nodes).
2. Unregister the subscription server if you no longer need it.
3. Repeat the same process for the publications.
4. After all replication system logical components are removed (except for possibly the publication server and subscription server) you can drop any of the physical database objects in Oracle, SQL Server, or Postgres. Don't drop the control schema objects manually, for example by using an SQL command line utility. Doing so can cause the Replication Server console and CLI to stop working. See Deleting the control schema and control schema objects if this problem occurs. Deleting the replication system logical components using the Replication Server console or CLI drops the control schema objects from the physical database.

- The order of removing a multi-master replication system is as follows:

1. Remove the replication system logical components using the Replication Server console or CLI starting with the publication database definitions of the non-MDN nodes.
2. Remove the publication from under the primary definition node.
3. Remove the publication database definition of the primary definition node. After all replication system logical components are removed (except for possibly the publication server) you can drop any of the physical database objects in Postgres. Don't drop the control schema objects manually, for example by using an SQL command line utility. Doing so can cause the Replication Server console and CLI to stop working.

> **Note**
>
> For partition tables, replication is supported only when all the partition tables have a primary key.

4.4.3 Restrictions on replicated database objects

When you create a subscription in a single-master replication system, the table definitions and most database objects and attributes associated with the publication tables are created in the subscription database by the subscription server.

If you choose, the same process can automatically occur when a primary node is added to a multi-master replication system. You can create the table definitions and most database objects and attributes associated with the publication tables in the newly added primary node by the publication server.

The following is a list of database objects and table attributes that are replicated from the publication in either a single-master or multi-master replication system.

- Tables
- Views (for snapshot-only publications) created as tables in the subscription database
- Materialized Views (for Oracle snapshot-only publications) created as tables in the subscription database
- Primary keys
- Not null constraints
- Unique constraints
- Check constraints
- Indexes

> **Note**
>
> Foreign key constraints aren't replicated by the publication or subscription server in a single-master replication system. However, in a multi-master replication system, foreign key constraints are replicated from the primary definition node to other primary nodes.

> **Note**
>
> Sequences (database objects created by the `CREATE SEQUENCE` statement) aren't replicated from the publication database to the subscription databases in a single-master replication system. Sequences also aren't replicated from the primary definition node to other primary nodes in a multi-master replication system.

Replication Server does have some restrictions on the types of tables it can replicate.

Restrictions on Oracle database objects

The following are the restrictions on Oracle database objects:

- You can replicate certain types of Oracle partitioned tables. See Replicating Oracle partitioned tables for details.
- You can't replicate Oracle global temporary tables.
- You can use Oracle tables with the `RAW` data type in snapshot-only publications but not in synchronization replications.
- You can include Oracle materialised views in snapshot-only publications but not in synchronization replications.
- You can't replicate Oracle tables that include the following data types:
  - `BFILE`
  - `BINARY_DOUBLE`
  - `BINARY_FLOAT`
  - `MLSLABEL`
  - `LONG`
  - `LONG RAW`
- Replication Server 7.0 provides support for LOB column streaming from Oracle to PostgreSQL/EDB Postgres Advanced Server. However, you must create LOB columns to allow NULLs on the source Oracle and target PostreSQL/EDB Postgres Advanced Server database server. The previous version of Replication Server didn't stream the LOB columns. Rows of tables containing LOB columns were replicated during streaming, but LOB columns contained NULL. This limitation was removed.

> **Note**
>
> If you upgrade an existing Replication Server 6.x deployment to 7.0 and the published tables contain LOB columns, you must replublish all the tables with LOB columns to avoid data inconsistency.

- Oracle provides different methods to insert and update LOB columns. Replication Server supports LOB streaming only if LOB columns are inserted, updated, and deleted using `INSERT`, `UPDATE`, and `DELETE` statements.
- If you use the command line SQL*Plus tool, the maximum LOB size that you can insert or update is 4K. You can use Java program to insert large values, but you must use an `INSERT` or `UPDATE` statement.
- If JDBC LOB APIs (provided by `java.sql.Clob` or `java.sql.Blob` classes) are used, triggers won't activate, and hence streaming for LOB columns won't work.
- Similarly, if you use functions and procedures provided by the `DBMS_LOB` package, triggers won't activate, and streaming for LOB columns won't work.

Restrictions on SQL Server database pbjects

You can't replicate SQL Server tables that include the following data types:

- `GEOGRAPHY`
- `GEOMETRY`
- `SQL_VARIANT`

> **Note**

> See Replicating the SQL Server SQL_VARIANT data type for a method to replicate tables containing the `SQL_VARIANT` data type under certain conditions.

You can use SQL Server tables with the following data types in snapshot-only publications but not in synchronization replications:

- `BINARY`
- `IMAGE`
- `NTEXT`
- `NVARCHAR(max)`
- `TEXT`
- `TIMESTAMP`
- `VARBINARY`
- `VARBINARY(max)`

Restrictions on Postgres database objects

For replicating Postgres partitioned tables, see Replicating Postgres partitioned tables for details. You can't replicate Postgres tables with the following data types in a column that's part of the identity columns:

- `BLOB`
- `BYTEA`
- `RAW`

This restriction applies to primary key or unique columns. See Design considerations.

You can't replicate PostgreSQL or EDB Postgres Advanced Server database tables to the Oracle database that include the following data type:

- `JSONB`
- Geometry data types
- `tsvector` , `tsquery` , `txid_snapshot` , `pg_lsn`
- `cidr` , `inet` , `mcaddress` , `mcaddress8` , `uuid` , `money`
- `ENUM` , `ARRAY` , range data types (such as `int4range` , `tstzrange` , `numrange` , `daterange`)
- Any user-defined data type (that is, defined as `CREATE TYPE type_name`)

You can't replicate Postgres tables that include `OID` -based large objects. For information on `OID` -based large objects, see `pg_largeobject` in the PostgreSQL core documentation.

You can't replicate Postgres tables to an Oracle subscription database that include any geometric data types such as `POINT` and `POLYGON` .

You can't replicate Postgres tables to a SQL Server subscription database that include the following data types:

- `ABSTIME`
- `ACLITEM`
- `CHKPASS`
- Geometric data types (such as `LINE` , `PATH` , `POLYGON`)
- `CIRCLE`
- `CUBE`
- `JSON`
- `JSONB`
- `ROWID`
- `tsvector` , `tsquery` , `txid_snapshot` , `pg_lsn`
- `SEG`
- `Any ARRAY` data type (that is, defined as `data_type[]`)
- `ENUM` , composite type, range data types (such as `int4range` , `tstzrange` , `numrange` , `daterange`), `mcaddress8`
- Any user-defined data type (that is, defined as `CREATE TYPE <type_name>`)

Restrictions on range data types

Postgres data types called range types were first supported in PostgreSQL version 9.2 and EDB Postgres Advanced Server version 9.2. Built-in range types refer to the following built-in data types: `int4range`, `int8range`, `numrange`, `tsrange`, `tstzrange`, and `daterange`.

You can include Postgres tables containing the built-in range types in the publication of a single-master or multi-master replication system. However, this results in the following restrictions on the subscription databases of a single-master replication system or the additional primary nodes of a multi-master replication system:

- If a publication table of a single-master replication system contains any built-in range types, then a you can add a database as a subscription database only if the database server of the intended subscription database is Postgres version 9.2 or later.
- If a publication table of the primary definition node in a multi-master replication system contains any built-in range types, then you can add a database as a primary node only if the database server of this intended primary node is Postgres version 9.2 or later.

Custom range types constructed with the `CREATE TYPE AS RANGE` command aren't supported in Replication Server.

Limitations during re-snapshot for views

As part of the re-snapshot process, Replication Server drops constraints from the tables included in the snapshot using the `CASCADE` option. As a result, any objects, such as views in certain use cases, are also dropped. The following example illustrates a situation with a view that contains a `GROUP BY` clause that depends on the constraints in the underlying table where if the `CASCADE` option were not used, the constraints could not be dropped for the snapshot operation.

```
 CREATE TABLE t_place_type
(
  id serial NOT
NULL,
  c_name character
varying(100),
  CONSTRAINT pk_t_place_type PRIMARY KEY
(id)
);

CREATE TABLE t_place
(
  id serial NOT
NULL,
  c_name character
varying(50),
  id_place_type integer,
  CONSTRAINT pk_t_place PRIMARY KEY
(id),
  CONSTRAINT fk_t_place_t_place_type FOREIGN KEY (id_place_type)
      REFERENCES t_place_type (id) MATCH
SIMPLE
      ON UPDATE NO ACTION ON DELETE NO
ACTION
);

CREATE OR REPLACE VIEW v_place AS
 SELECT p.id,
    p.c_name,
    pt.c_name AS c_place_type
   FROM t_place
p
     LEFT JOIN t_place_type pt ON pt.id =
p.id_place_type
  GROUP BY p.id, pt.id,
p.c_name;
```

```
ALTER TABLE t_place DROP CONSTRAINT
fk_t_place_t_place_type;
ALTER TABLE t_place DROP CONSTRAINT
pk_t_place;
ALTER TABLE t_place_type DROP CONSTRAINT
pk_t_place_type;

ERROR:  cannot drop constraint pk_t_place_type on table t_place_type because other objects depend on
it
DETAIL:  view v_place depends on constraint pk_t_place_type on table t_place_type
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

Since Replication Server drops the constraints using the `CASCADE` option, the above error would not occur; however, the result of using this option means that the view will also be dropped.

4.4.4 Performance considerations

Keep these general guidelines for performance in mind as you design your replication system.

When to use snapshot or synchronization

Generally, synchronization is the quickest replication method since it applies changes to the target tables only since the last replication occurred.

However, suppose a large percentage of rows change between each replication. At some point it might be faster to completely reload the target tables using a snapshot than to execute individual SQL statements on a large percentage of rows as with synchronization replication. You might need to experiment to determine if, and at what point, a snapshot is faster.

Snapshot replication might be an option for tables with the following characteristics:

- Tables are relatively small
- A large percentage of rows change between replications

Synchronization replication is the better option for tables with the following characteristics:

- Tables are large
- A small percentage of rows change between replications

In a single-master replication system, you might find that one group of tables consistently replicates faster using snapshot replication. In this case, you can make these tables part of a snapshot-only publication while the remaining tables are members of a publication that uses synchronization replication.

When to use on-demand replication

The Replication Server console and CLI both enable you to immediately start a replication. This ability is called on-demand replication.

You can perform on-demand replication at any time regardless of whether there's an existing schedule. An on-demand replication doesn't change the date and time when the next replication will occur according to an existing schedule.

If a publication is a snapshot-only publication, then the only type of on-demand replication that you can perform on the publication is a snapshot.

If a publication is not a snapshot-only publication, you can perform an on-demand replication using either the snapshot method or the synchronization

method.

When you're in the development and testing phases of your replication system, you typically use on-demand replication so that you can immediately force the replication to occur and analyze the results.

When your replication system is ready for production, you typically use a schedule so that replications can occur unattended at regular time intervals. See Creating a schedule.

In other situations, you want to force a replication to take place ahead of its normal schedule. Reasons for performing an on-demand replication include the following:

- The number of changes to the source tables is growing at a faster rate than usual, and you don't want to wait for the regularly scheduled synchronization time to replicate all of the accumulated changes.
- You set up your replication system to perform synchronizations, but an unusually large number of changes were made to the source tables. You want to perform a snapshot of all source tables rather than execute a large number of SQL statements against the target tables.
- Changes were made directly to the rows of the target tables so that they no longer have the same content as their source table counterparts. You can perform an on-demand snapshot replication to reload all rows of the target tables from your current set of source tables.

> **Note**
>
> In a multi-master replication system, you can make on-demand snapshots only from the primary definition node to another primary node.

See On-demand replication for information on performing an on-demand replication for a single-master replication system. See On-demand replication for a multi-master replication system.

4.4.5 Distributed replication

Replication Server provides the flexibility for you to run the replication system's components on separate machines on a network.

Replication Server is designed so that you can set up replication systems where:

- Each of the components (publication server, subscription server, publication database, subscription database, and primary nodes) can all run on the same host
- Each component can run on its own separate host
- Any combination of components can run on any number of hosts

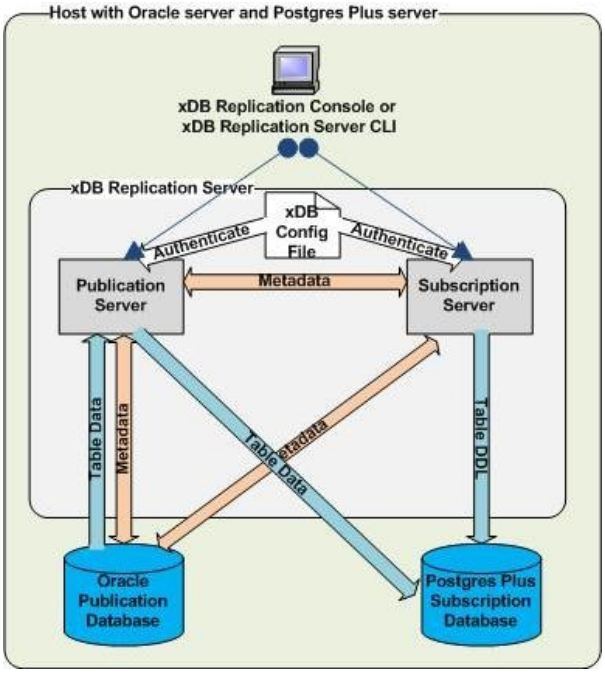
However, for practical purposes, there are two basic scenarios. The simplest case is where all components are on the same host. The other case is where you have the Oracle or SQL Server database server running on a host separate from the rest of the replication system components.

Each scenario has advantages and disadvantages.

Single host

The simplest implementation of a replication system is when all replication components run on a single host. This means that the PostgreSQL or EDB Postgres Advanced Server installation, the complete Replication Server installation (publication server and subscription server), and the Oracle or SQL Server database server reside on the same machine.

The Postgres publication or subscription database can reside in the initial database cluster created when Postgres is installed on the host.



The advantages of a single-host replication system are the following:

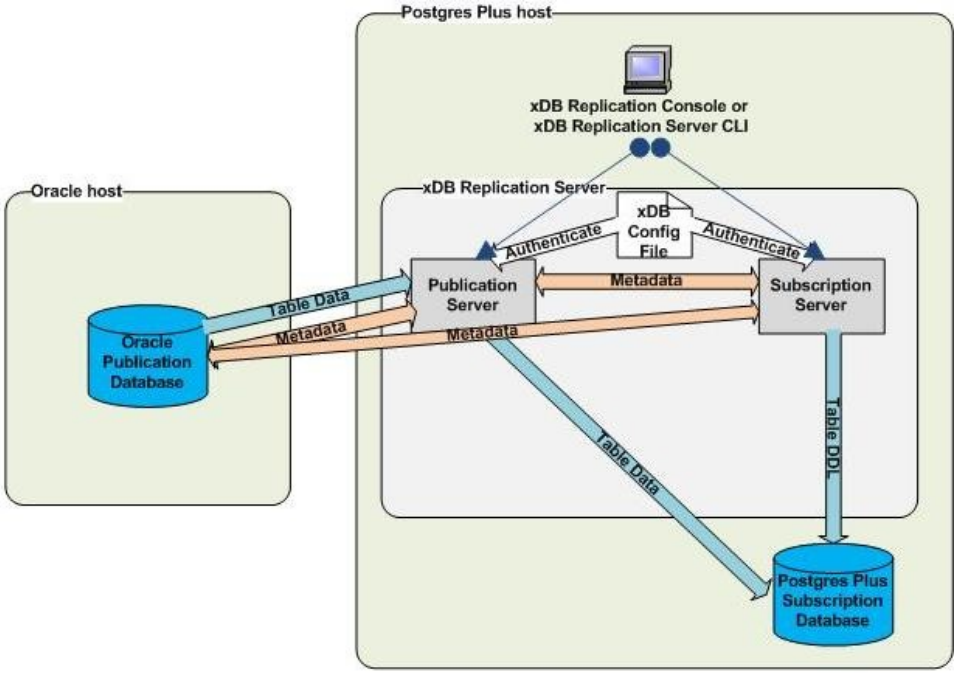
- Performance is better since there's no network over which to push replication data, especially if large snapshots are involved.
- Configuration is much simpler. When creating the replication system logical components, the IP addresses of all components are the same.

The disadvantages of a single host replication system are the following:

- The replication system and the database servers all consume the resources of one machine, which can adversely affect database application performance.
- The publication and subscription databases might be in different geographic locations, thereby requiring multiple networked hosts.
- Your site might require the use of a dedicated host for the Oracle or SQL Server database server, so Replication Server can't reside on the same machine.

Single-master replication distributed hosts

Replication Server allows you to build a replication system with either or both of the publication database and the subscription database on separate hosts, as shown in the figure:



You can use the same remote distribution for the subscription database instead of or in addition to the publication database.

The advantages of a distributed host replication system are the following:

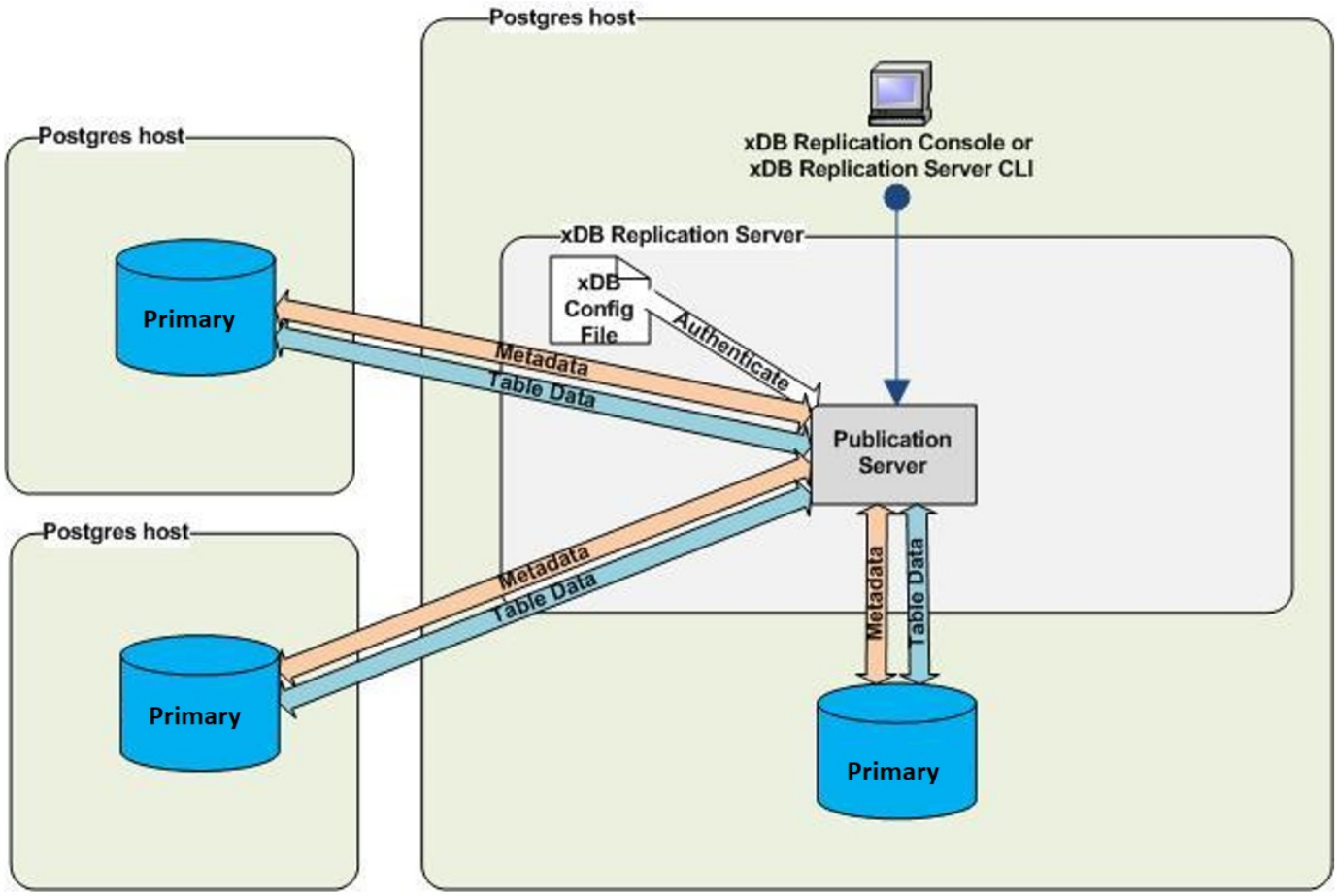
- The replication system and the database servers can each consume the resources of their own machines, which you can individually select and tune.
- The publication and subscription databases can be in different geographic locations.
- You can enforce stronger database security if only the database server is allowed to run on a host.

The disadvantages of a distributed host replication system are the following:

- There might be a performance disadvantage since there is a network over which to push replication data, especially if large snapshots are involved.
- Installation is more complex if the Postgres database must run on a different host than Replication Server. Installation involves installing Postgres on two separate hosts.
- Configuration is more complex. The network and firewalls must be properly configured to allow the distributed components to communicate. When creating the replication system logical components, you must use the correct IP addresses of all components. In addition, you must keep the correct IP addresses up to date in the replication system control schema if they change in the networked environment.

Multi-master replication distributed hosts

In a multi-master replication system, the Postgres database servers running the primary nodes can run on a single or multiple hosts. The following figure shows two primary nodes running on database servers on separate hosts as well as a primary node running on the same database server as the publication server.



5 Supported Java platforms

Replication Server is certified to work with the following Java platforms:

Operating systems	JDK versions
CentOS 7	Red Hat OpenJDK 8
SLES 12	Red Hat OpenJDK 8
SLES 15	OpenJDK 11
RHEL 7 and 8	- Red Hat OpenJDK 8 - Oracle JDK 8
RHEL 9	Red Hat OpenJDK 11
Rocky Linux/AlmaLinux 8	Red Hat OpenJDK 8
Rocky Linux 9 /AlmaLinux 9	Red Hat OpenJDK 11
PPCLE RHEL 8	Red Hat OpenJDK 8
PPCLE RHEL 9	Red Hat OpenJDK 11
Windows Server 2019	Red Hat OpenJDK 8
Windows Server 2022	Oracle JDK 8
Debian 10 and 11	Red Hat OpenJDK 11
Ubuntu 20, 22	OpenJDK 11

See Product Compatibility for more information on operating system support.

6 Installing Replication Server

Select a link to access the applicable installation instructions:

Linux x86-64 (amd64)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8, RHEL 7
- Oracle Linux (OL) 9, Oracle Linux (OL) 8, Oracle Linux (OL) 7
- Rocky Linux 9, Rocky Linux 8
- AlmaLinux 9, AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

Debian and derivatives

- Ubuntu 22.04, Ubuntu 20.04
- Debian 11, Debian 10

Linux IBM Power (ppc64le)

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9, RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15, SLES 12

Windows

- Windows Server 2022, Windows Server 2019

6.1 Installing Replication Server on Linux x86 (amd64)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL) and derivatives

- RHEL 9
- RHEL 8
- RHEL 7
- Oracle Linux (OL) 9
- Oracle Linux (OL) 8
- Oracle Linux (OL) 7
- Rocky Linux 9
- Rocky Linux 8
- AlmaLinux 9
- AlmaLinux 8
- CentOS 7

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

Debian and derivatives

- Ubuntu 22.04
- Ubuntu 20.04
- Debian 11
- Debian 10

6.1.1 Installing Replication Server on RHEL 9 or OL 9 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.2 Installing Replication Server on RHEL 8 or OL 8 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.3 Installing Replication Server on AlmaLinux 9 or Rocky Linux 9 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.4 Installing Replication Server on AlmaLinux 8 or Rocky Linux 8 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.5 Installing Replication Server on RHEL 7 or OL 7 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

- Enable additional repositories to resolve dependencies:

```
subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms"  --enable
"rhel-ha-for-rhel-*-server-rpms"
```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo yum  -y install edb-xdb
```

To install an individual component:

```
sudo yum  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.6 Installing Replication Server on CentOS 7 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

```
dnf repolist | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Install the EPEL repository:

```
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo yum  -y install edb-xdb
```

To install an individual component:

```
sudo yum  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.7 Installing Replication Server on SLES 15 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/15.4/x86_64
  ```

- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo zypper  -n install edb-xdb
```

To install an individual component:

```
sudo zypper  -n install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface

Package name	Component
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.8 Installing Replication Server on SLES 12 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

  ```
  sudo SUSEConnect -p PackageHub/12.5/x86_64
  sudo SUSEConnect -p sle-sdk/12.5/x86_64
  ```

- Refresh the metadata:

  ```
  sudo zypper refresh
  ```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo zypper  -n install edb-xdb
```

To install an individual component:

```
sudo zypper  -n install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.9 Installing Replication Server on Ubuntu 22.04 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.

4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo apt-get  -y install edb-xdb
```

To install an individual component:

```
sudo apt-get  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.10 Installing Replication Server on Ubuntu 20.04 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  `apt-cache search enterprisedb`

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo apt-get  -y install edb-xdb
```

To install an individual component:

```
sudo apt-get  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.11 Installing Replication Server on Debian 11 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

```
apt-cache search enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo apt-get  -y install edb-xdb
```

To install an individual component:

```
sudo apt-get  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.1.12 Installing Replication Server on Debian 10 x86_64

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  apt-cache search enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo apt-get  -y install edb-xdb
```

To install an individual component:

```
sudo apt-get  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.2 Installing Replication Server on Linux IBM Power (ppc64le)

Operating system-specific install instructions are described in the corresponding documentation:

Red Hat Enterprise Linux (RHEL)

- RHEL 9
- RHEL 8

SUSE Linux Enterprise (SLES)

- SLES 15
- SLES 12

6.2.1 Installing Replication Server on RHEL 9 ppc64le

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.2.2 Installing Replication Server on RHEL 8 ppc64le

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  dnf repolist | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.
  4. Follow the instructions for setting up the EDB repository.

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo dnf  -y install edb-xdb
```

To install an individual component:

```
sudo dnf  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.2.3 Installing Replication Server on SLES 15 ppc64le

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

  To determine if your repository exists, enter this command:

  ```
  zypper lr -E | grep enterprisedb
  ```

  If no output is generated, the repository isn't installed.

  To set up the EDB repository:

  1. Go to EDB repositories.
  2. Select the button that provides access to the EDB repository.
  3. Select the platform and software that you want to download.

4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/15.4/ppc64le
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo zypper  -y install edb-xdb
```

To install an individual component:

```
sudo zypper  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.2.4 Installing Replication Server on SLES 12 ppc64le

Prerequisites

Before you begin the installation process:

- Set up the EDB repository.

  Setting up the repository is a one-time task. If you have already set up your repository, you don't need to perform this step.

To determine if your repository exists, enter this command:

```
zypper lr -E | grep enterprisedb
```

If no output is generated, the repository isn't installed.

To set up the EDB repository:

1. Go to EDB repositories.
2. Select the button that provides access to the EDB repository.
3. Select the platform and software that you want to download.
4. Follow the instructions for setting up the EDB repository.

- Activate the required SUSE module:

```
sudo SUSEConnect -p PackageHub/12.5/ppc64le
sudo SUSEConnect -p sle-sdk/12.5/ppc64le
```

- Refresh the metadata:

```
sudo zypper refresh
```

Install the package

You can install all Replication Server components with a single install command, or you may choose to install selected, individual components by installing only those particular packages.

To install all Replication Server components:

```
sudo zypper  -y install edb-xdb
```

To install an individual component:

```
sudo zypper  -y install <package_name>
```

Where `<package_name>` is:

Package name	Component
`edb-xdb-console`	Replication console and the Replication Server command line interface
`edb-xdb-publisher`	Publication server
`edb-xdb-subscriber`	Subscription server

Initial configuration

Before using Replication Server, you must download and install JDBC drivers. See Installing a JDBC driver for details.

6.3 Installing Replication Server on Windows

EDB provides a graphical interactive installer for Windows. You can access it two ways:

- Download the graphical installer from the Downloads page and invoke the installer directly. See Installing directly.
- Use Stack Builder (with PostgreSQL) or StackBuilder Plus (with EDB Postgres Advanced Server) to download the EDB installer package and invoke the graphical installer. See Using Stack Builder or StackBuilder Plus.

Prerequisites

- You must have Java Runtime Environment (JRE) version 1.8 or later installed on the hosts where you intend to install any Replication Server component (Replication Console, publication server, or subscription server). You can use any Java product, such as Oracle Java or OpenJDK. Follow the directions for your host operating system to install the Java runtime environment.
- Be sure the system environment variable, `JAVA_HOME`, is set to the JRE installation directory of the JRE version you want to use with Replication Server. Make sure that the 64-bit version of JRE is installed and `JAVA_HOME` is set before installing Replication Server.

> **Note**
>
> Replication Server doesn't support JRE installations with 32-bit.

Installing directly

After downloading the graphical installer, to start the installation wizard, assume sufficient privileges (superuser or administrator) and double-click the installer icon. If prompted, provide a password.

In some versions of Windows, to invoke the installer with administrator privileges, you need to right-click the installer icon and select **Run as Administrator** from the context menu.

Proceed to Using the graphical installer.

Using Stack Builder or StackBuilder Plus

If you're using PostgreSQL, you can invoke the graphical installer with Stack Builder. SeeUsing Stack Builder.

1. In Stack Builder, follow the prompts until you get to the module selection page.
2. Expand the **Registration-required and trial products** node.
3. Expand the **EnterpriseDB Tools** node and select **Replication Server**.
4. Proceed to Using the graphical installer.

If you're using EDB Postgres Advanced Server, you can invoke the graphical installer with StackBuilder Plus. SeeUsing StackBuilder Plus.

1. In StackBuilder Plus, follow the prompts until you get to the module selection page.
2. Expand the **EnterpriseDB Tools** node and select **Replication Server**.
3. Proceed to Using the graphical installer.

Using the graphical installer

1. Select the installation language and select **OK**.
2. On the Setup Replication Server page, select **Next**.
3. Read the license agreement. If you accept the agreement, select the **I accept the agreement** option and select **Next**.
4. Browse to a directory where you want to install the Replication Server components, or leave the default location. Select **Next**.
5. If you don't want a particular Replication Server component installed, clear the box next to the component name. Select **Next**.
6. Enter information for the Replication Server administrator.

   > **Note**
   >
   > From this point on, we suggest you record the values you enter as you need them during the publication and subscription server registration process.

   Enter values for the following fields:

   - **Admin User** — The Replication Server administrator user name needed to authenticate some Replication Server actions, such as registering a publication server or subscription server running on this host. You can enter any alphanumeric string for the admin user name. The default admin user name is admin.
   - **Admin Password** — Password of your choice for the Replication Server administrator.

   The admin user and the admin password (in encrypted form) are saved to the `XDB_HOME\etc\edb-repl.conf` configuration file. Select **Next**.
7. If a publication server is being installed, enter an available port on which the publication server can run. The default port number is `9051` . Select **Next**.
8. If a subscription server is being installed, enter an available port on which the subscription server can run. The default port number is `9052` . Select **Next**.
9. If you're using EDB Postgres Advanced Server installed in Oracle-compatible configuration mode, enter `postgres` or `enterprisedb` for the operating system account under which the publication server or subscription server runs.
10. On the Ready to Install page, select **Next**.

    An information box shows the installation progress of the selected components. This might take a few minutes.
11. When the installation is complete, select **Finish**.

Successful installation of Replication Server results in the creation of directory structures and files in your host environment as described in Installation

details. Verify that the path to your Java runtime program set in `XDB_HOME\etc\edb-repl.conf` is correct.

6.4 Installing a JDBC driver

Choosing and installing a JDBC driver

The JDBC driver you use depends on the database you're using. If you're using:

- **EDB Postgres Advanced Server**, use the EDB JDBC driver. To download the latest driver, see EDB Connectors on the EDB Downloads page. For installation instructions, see Installing and configuring EDB JDBC Connector.
- **PostgreSQL**, use the PostgreSQL JDBC driver. To download the latest supported driver (Java 8), see the JDBC drivers section on the PostgreSQL Downloads page.
- **Oracle**, use the freely available Oracle JDBC driver.
- **Microsoft SQL Server**, use the freely available Microsoft SQL Server JDBC driver.

> **Note**
>
> For Microsoft SQL Server, if your system uses JDK/JAVA, then we recommend that you use an equal or earlier mssql-jdbc jre version than the JDK/JAVA version on your system. For example, a system with JDK/JAVA version 18 can use `mssql-jdbc-11.2.0.jre18.jar`, `mssql-jdbc-11.2.0.jre17.jar`, and so on.

Configuring the driver

After downloading the driver, create a symlink in the `XDB-install-folder/lib/jdbc` directory that points to the location where you installed the driver.

For Linux, create a symlink for the driver using these naming conventions.

Driver	symlink name
EDB	edb-jdbc18.jar
PostgreSQL	postgresql.jar
Oracle	ojdbc.jar
Microsoft SQL Server	mssql-jdbc.jar
jTDS	jtds.jar

For Windows, copy the drivers to the `XDB-install-folder/lib/jdbc` directory and rename them using the same naming conventions.

6.5 Installation details

Windows details

On Windows systems, the publication server and subscription server run as services named `Publication Service` and `Subscription Service`.

Linux details

On Linux hosts where you installed Replication Server with the graphical installer or from the command line, a publication server daemon and a subscription server daemon are now running on your computer, assuming you chose to install the publication server and subscription server components. If you installed the Replication Server RPM package, you must start the publication server and the subscription server based on the instructions in Registering a publication server for the publication server and Registering a subscription server for the subscription server.

> **Note**
>
> On some Linux systems, you might have to restart the server before you can see the EPRS Replication Console choice in the application menu. If the Replication Console choice is still unavailable in the application menu, you can start it by invoking the script `XDB_HOME/bin/runRepConsole.sh`.

Additional details

The Postgres application menu contains a new item for the EPRS Replication Console.

> **Note**
>
> If Replication Server is installed from a Replication Server RPM package, start the EPRS Replication Console by invoking the script `XDB_HOME/bin/runRepConsole.sh`.

During the configuration process, you might need the following files that are created during installation.

File Name	Location	Description
`edb-repl.conf` (Linux)	`/etc`	EPRS Replication Configuration file
`edb-repl.conf` (Windows)	`XDB_HOME\etc`	EPRS Replication Configuration file
`edb-xdbpubserver` (Linux)	`/etc/init.d`	Start, stop, or restart the publication server
`edb-xdbpubserver.service` (Linux)	`/usr/lib/systemd/system`	Start, stop, or restart the publication server (CentOS 7, RHEL 7, Rocky Linux 8, AlmaLinux 8, RHEL 8)
`edb-xdbsubserver` (Linux)	`/etc/init.d`	Start, stop, or restart the subscription server
`edb-xdbsubserver.service` (Linux)	`/usr/lib/systemd/system`	Start, stop, or restart the subscription server (CentOS 7, RHEL 7, Rocky Linux 8, AlmaLinux 8, RHEL 8)
`xdb_pubserver.conf`	`XDB_HOME/etc`	Publication server configuration file
`xdb_subserver.conf`	`XDB_HOME/etc`	Subscription server configuration file
`xdbReplicationServer-xx.config`	`XDB_HOME/etc/sysconfig`	Replication Server Startup Configuration file
`pubserver.log` (Linux)	`var/log/edb/xdb`	Publication server log file
`pubserver.log` (Windows)	`POSTGRES_HOME\.enterprisedb\xdb\x.x`	Publication server log file
`subserver.log` (Linux)	`var/log/edb/xdb`	Subscription server log file
`subserver.log` (Windows)	`POSTGRES_HOME\.enterprisedb\xdb\x.x`	Subscription server log file

File Name	Location	Description
`edb-xdbpubserver.log` (Linux)	`/var/log/edb/xdb`	Publication services startup log file
`edb-xdbsubserver.log` (Linux)	`/var/log/edb/xdb`	Subscription services startup log file
`servers.xml`	`USER_HOME/.enterprisedb/xdb/x.x`	Server login file

> **Note**
>
> - `XDB_HOME` is the directory where Replication Server is installed.
> - For Linux, Replication Server is installed in the `/usr/edb/xdb` directory.
> - For Windows, Replication Server is installed in the `C:\Program Files\edb\EnterpriseDB-xDBReplicationServer` directory.

> **Note**
>
> `POSTGRES_HOME` is the home directory of the postgres operating system account (enterprisedb for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode).

> **Note**
>
> The publication and subscription services startup log files (`edb-xdbpubserver.log` and `edb-xdbsubserver.log`) aren't generated for Windows and Mac OS X operating systems.

> **Note**
>
> `USER_HOME` is the home directory of the operating system account in use.

> **Note**
>
> The Replication Server version number is represented by `x.x` or by `xx` (for example `7.0` or `70`).

6.6 Upgrading Replication Server

You can install Replication Server 7 when you have existing single-master or multi-master replication systems that are running under Replication Server version 7.

It's assumed that you're installing Replication Server 7.x on the same host machine that's currently running the earlier version of Replication Server you're upgrading from and that you plan to then manage the existing replication systems using Replication Server 7.x.

If you're using a version of Replication Server earlier than 6.2.15, first upgrade to 6.2.15 or a later 6.2.x point version before upgrading to 7.x.

> **Note**
>
> Version 7.x provides a non-breaking upgrade path for existing 6.2.x based cluster deployments. However, we strongly recommended that you verify the upgrade in a staging or nonproduction environment before applying the upgrade in a production environment. There's no downgrade path from version 7.x to version 6.2.x, so it's essential to test the upgrade first before applying it to the production environment.

For more details on upgrading Replication Server, see:

- Updating the publication and subscription server

- Upgrading from a Replication Server 6.2 installation on Linux
- Upgrading from a Replication Server 7.x installation on Linux
- Upgrading with the graphical user interface installer

After upgrading and before using Replication Server, you need to download a JDBC driver and create a symlink to it (for Linux) or rename the driver (for Windows). See Installing a JDBC driver for more information.

6.6.1 Updating the publication and subscription server ports

The newly installed publication server and subscription server of Replication Server 7 are configured to use the default port numbers 9051 and 9052, respectively. These port numbers are set in the Replication Server startup configuration file as described Replication Server configuration file.

If your Replication Server 6.2.x replication systems were running under port numbers other than 9051 and 9052, you must adjust some of your settings in Replication Server 7 to continue to use these existing replication systems.

> **Note**
>
> The following changes regarding port 9052 and the subscription server are needed only if you're running a single-master replication system. If you're using only a multi-master replication system, then only the changes involving port 9051 and the publication server are needed.

You can use either of two methods can correct this:

- To continue to use the old port numbers (other than 9051 and 9052) that were in use for Replication Server 6.2.x, stop the publication and subscription servers. Change the settings of the `PUBPORT` and `SUBPORT` parameters in the Replication Server Startup Configuration file from `9051` and `9052` to the old port numbers used by Replication Server 6.2.x. Restart the publication and subscription servers. Register the publication server and the subscription server with the old Replication Server 6.2.x port numbers along with the admin user and password as described in Registering a publication server and Registering a subscription server.
- To use the default port numbers 9051 and 9052 with the Replication Server 7 replication systems, you must replace the old port numbers with the default port numbers 9051 and 9052. Register the publication server and the subscription server with port numbers 9051 and 9052 along with the admin user and password as described in Registering a publication server and Registering a subscription server. For single-master replication systems only, you then need to change the port numbers stored in the control schema from the old port numbers to 9051 and 9052. First, perform the procedure described in Subscription server network location, and then perform the procedure described in Updating a subscription.

After making the changes, select **Refresh** in the Replication Server console. The replication tree of the Replication Server console displays the complete set of nodes for the replication systems.

6.6.2 Upgrading from a Replication Server 7.x installation on Linux

If you have an existing Replication Server 7.x installation on Linux, you can use yum to upgrade your repository configuration file and update to a more recent product version. To update the `edb.repo` file, assume superuser privileges and enter:

```
yum upgrade edb-repo
```

`yum` updates the `edb.repo` file to enable access to the current EDB repository, configured to connect with the credentials specified in your `edb.repo` file. Then, you can use yum to upgrade any installed packages:

```
yum upgrade edb-xdb*
```

If you're upgrading from a Replication Server 6.2 installation on Linux, see Upgrading from a Replication Server 6.2 installation on Linux for details.

After upgrading and before using Replication Server, you need to download a JDBC driver and create a symlink to it. SeeInstalling a JDBC driver for more information.

6.6.3 Upgrading with the graphical installer

You can upgrade to Replication Server 7 using the graphical installer.

1. Before starting the upgrade process, replicate any pending backlog of transactions on the publication tables.

2. After all pending transactions are replicated to their target databases, stop the Replication Server 6.2.x publication server and subscription server. See Registering a publication server and Registering a subscription server.

3. Ensure the installation user has administrative permissions on the `XDB_HOME/xdata` folder. On Windows, you can do this by opening the Replication Server installation directory in Windows Explorer and selecting the xdata folder. When prompted, select **Continue** to enable the required permission.

4. Install Replication Server 7. See Installation and uninstallation for instructions on installing Replication Server, but note the differences described in the following steps.

5. Following the acceptance of the license agreement, the Select Components screen appears but with the entries disabled. The old Replication Server components are replaced with the new ones in the old Replication Server's directory location. Select **Next**.

6. The Existing Installation screen confirms that an existing Replication Server installation was found. To proceed with the upgrade, select **Next**.

7. On the Ready to Install screen, select **Next**.

   The remaining screens that appear confirm completion of the installation process and allow you to exit from Stack Builder or StackBuilder Plus.

8. After installation completes, the publication server of the new Replication Server product is running, connected to the controller database used by Replication Server 6.2. The subscription server might be running at this point, which is an expected outcome of this process.

9. Complete the publication server and subscription server configuration file setup.

   In the `XDB_HOME/etc` directory, a new set of configuration files for Replication Server version 7 are created. These files are named `xdb_pubserver.conf.new` and `xdb_subserver.conf.new`. The new configuration files contain any new configuration options added for Replication Server 7.

   The old configuration files used by Replication Server version 6.2.x remain unchanged as `xdb_pubserver.conf` and `xdb_subserver.conf`.

   Merge the old and new configuration files so that the resulting active configuration files contain any new Replication Server 7 configuration options as well as any nondefault settings you used with the Replication Server 6.2.x and want to continue to use with Replication Server 7. The final set of active configuration files must be named `xdb_pubserver.conf` and `xdb_subserver.conf`.

   In the `XDB_HOME/etc/sysconfig` directory, make sure the Replication Server startup configuration file `xdbReplicationServer-62.config` contains the parameter settings you want to use with Replication Server 7. See Replication Server startup configuration file for more information.

10. Restart the publication server and the subscription server. See Registering a publication server and Registering a subscription server).

11. Check the publication server and subscription server log files to verify that no errors occurred (see Publication and subscription server startup failures).

12. If necessary, adjust the publication server and subscription server port numbers.

    The Replication Server 7 publication and subscription servers are installed to use the default port numbers `9051` and `9052`, respectively. If the Replication Server 6.2.x replication systems used port numbers other than `9051` and `9052`, then make the changes to correct this inconsistency as described in Updating the publication and subscription server ports.

If you don't need to adjust the port numbers, register the publication server and subscription server with the Replication Server console as described in Registering a publication server and Registering a subscription server. The existing replication systems appear in the replication tree of the Replication Server Console.

After upgrading and before using Replication Server, you need to download a JDBC driver and create a symlink to it (for Linux) or rename the driver (for Windows). See Installing a JDBC driver for more information.

You're now ready to use Replication Server 7 to create new replication systems and manage existing ones.

> **Note**
>
> **For Windows:** If you give a new admin password during an upgrade, it's ignored. After the upgrade, Replication Server picks the old admin user name and password, which is saved in `edb-replconf`.

6.6.4 Upgrading from a Replication Server 6.2 installation on Linux

If you're using Replication Server 6.2.x that was installed using the Replication Server RPM package, you can upgrade to Replication Server 7 from an RPM package.

> **Note**
>
> Be sure the repository configuration file `edb.repo` for Replication Server 7 is set up in the `/etc/yum.repos.d` directory.

1. Before starting the upgrade process, replicate any pending backlog of transactions on the publication tables.

2. After all pending transactions are replicated to their target databases, stop the Replication Server 6.2.x publication server and subscription server. See Registering a publication server and Registering a subscription server).

3. Save a copy of the following configuration files:

   - `/etc/edb-repl.conf`
   - `/usr/edb/xdb/etc/xdb_pubserver.conf`
   - `/usr/edb/xdb/etc/xdb_subserver.conf`
   - `/usr/edb/xdb/etc/sysconfig/xdbReplicationServer-70.config`

   Copies of these files are typically saved by the upgrade process if the files were modified since their original installation. However, it's safest to save copies in case the upgrade process doesn't. Use the saved files as your Replication Server 6.2.x configuration files for the updates described in Step 7.

4. If any Oracle publication or subscription databases are used in existing single-master replication systems, make sure a copy of the Oracle JDBC driver, version ojdbc5 or later, is accessible by the publication server and subscription server where Replication Server 7 will be installed. See Enabling access to Oracle for information.

   > **Note**
   >
   > Two options are available: 1) Copy the Oracle JDBC driver to the `jre/lib/ext` subdirectory of your Java runtime environment. 2) Copy the Oracle JDBC driver to the `lib/jdbc` subdirectory of the Replication Server installation directory.
   >
   > We recommend that you copy the Oracle JDBC driver to the `jre/lib/ext` subdirectory of your Java runtime environment. If you instead copy it to `lib/jdbc`, you must copy the Oracle JDBC driver to the `/usr/edb/xdb/lib/jdbc` directory after you install Replication Server 7.

5. Make sure that the controller database is up and running. The other publication and subscription databases of existing SMR and MMR systems don't need to be up and running.

6. As the root account, invoke the yum update command to begin the upgrade from Replication Server 6.2.x to Replication Server 7 as shown by the following:

```
yum update ppas-xdb*
```

Be sure to include the asterisk character (*) to update all Replication Server components.

The following is an example:

```
 [root@localhost ~]# yum update ppas-xdb*
Loaded plugins: fastestmirror, langpacks
Determining fastest mirrors
Resolving Dependencies
--> Running transaction check
---> Package edb-xdb.x86_64 0:7.0.0-1.rhel7 will be obsoleting
--> Processing Dependency: edb-jdbc-driver >= 42.2.24.1 for package: edb-xdb-7.0.0-1.rhel7.x86_64
--> Processing Dependency: edb-xdb-subscriber for package: edb-xdb-7.0.0-1.rhel7.x86_64
--> Processing Dependency: edb-xdb-publisher for package: edb-xdb-7.0.0-1.rhel7.x86_64
--> Processing Dependency: edb-xdb-libs for package: edb-xdb-7.0.0-1.rhel7.x86_64
--> Processing Dependency: edb-xdb-console for package: edb-xdb-7.0.0-1.rhel7.x86_64
---> Package ppas-xdb.x86_64 0:6.2.15b-1.rhel7 will be obsoleted
---> Package ppas-xdb-console.x86_64 0:6.2.15b-1.rhel7 will be obsoleted
---> Package ppas-xdb-libs.x86_64 0:6.2.15b-1.rhel7 will be obsoleted
---> Package ppas-xdb-publisher.x86_64 0:6.2.15b-1.rhel7 will be obsoleted
---> Package ppas-xdb-subscriber.x86_64 0:6.2.15b-1.rhel7 will be obsoleted
--> Running transaction check
---> Package edb-jdbc.x86_64 0:42.2.19.1-1.rhel7 will be updated
---> Package edb-jdbc.x86_64 0:42.2.24.1-1.rhel7 will be an update
---> Package edb-xdb-console.x86_64 0:7.0.0-1.rhel7 will be installed
---> Package edb-xdb-libs.x86_64 0:7.0.0-1.rhel7 will be installed
---> Package edb-xdb-publisher.x86_64 0:7.0.0-1.rhel7 will be installed
---> Package edb-xdb-subscriber.x86_64 0:7.0.0-1.rhel7 will be installed
---> Finished Dependency Resolution

Dependencies Resolved

================================================================================================================
=================================
Package Arch Version Repository Size
================================================================================================================
=================================
Installing:
edb-xdb x86_64 7.0.0-1.rhel7 edb 9.3 k
replacing ppas-xdb.x86_64 6.2.15b-1.rhel7
replacing ppas-xdb-console.x86_64 6.2.15b-1.rhel7
replacing ppas-xdb-libs.x86_64 6.2.15b-1.rhel7
replacing ppas-xdb-publisher.x86_64 6.2.15b-1.rhel7
replacing ppas-xdb-subscriber.x86_64 6.2.15b-1.rhel7
Installing for dependencies:
edb-xdb-console x86_64 7.0.0-1.rhel7 edb 1.6 M
edb-xdb-libs x86_64 7.0.0-1.rhel7 edb 13 M
edb-xdb-publisher x86_64 7.0.0-1.rhel7 edb 41 k
edb-xdb-subscriber x86_64 7.0.0-1.rhel7 edb 11 k
Updating for dependencies:
edb-jdbc x86_64 42.2.24.1-1.rhel7 edb 1.0 M

Transaction Summary
================================================================================================================
=================================
Install 1 Package (+4 Dependent packages)
```

```
Upgrade ( 1 Dependent package)

Total download size: 15 M
Is this ok [y/d/N]: y
Downloading packages:
No Presto metadata available for edb
(1/6): edb-xdb-7.0.0-1.rhel7.x86_64.rpm | 9.3 kB 00:00:01
(2/6): edb-xdb-console-7.0.0-1.rhel7.x86_64.rpm | 1.6 MB 00:00:09
(3/6): edb-jdbc-42.2.24.1-1.rhel7.x86_64.rpm | 1.0 MB 00:00:11
(4/6): edb-xdb-publisher-7.0.0-1.rhel7.x86_64.rpm | 41 kB 00:00:01
(5/6): edb-xdb-subscriber-7.0.0-1.rhel7.x86_64.rpm | 11 kB 00:00:00
(6/6): edb-xdb-libs-7.0.0-1.rhel7.x86_64.rpm | 13 MB 00:02:14
--------------------------------------------------------------------------------------------------------------
---------------------------------
Total 109 kB/s | 15 MB 00:02:25
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : edb-xdb-libs-7.0.0-1.rhel7.x86_64 1/12
Installing : edb-xdb-publisher-7.0.0-1.rhel7.x86_64 2/12
Installing : edb-xdb-console-7.0.0-1.rhel7.x86_64 3/12
Installing : edb-xdb-subscriber-7.0.0-1.rhel7.x86_64 4/12
Updating : edb-jdbc-42.2.24.1-1.rhel7.x86_64 5/12
Installing : edb-xdb-7.0.0-1.rhel7.x86_64 6/12
Erasing : ppas-xdb-6.2.15b-1.rhel7.x86_64 7/12
Erasing : ppas-xdb-subscriber-6.2.15b-1.rhel7.x86_64 8/12
warning: /usr/ppas-xdb-6.2/etc/xdb_subserver.conf saved as /usr/ppas-xdb-
6.2/etc/xdb_subserver.conf.rpmsave
Erasing : ppas-xdb-console-6.2.15b-1.rhel7.x86_64 9/12
Cleanup : edb-jdbc-42.2.19.1-1.rhel7.x86_64 10/12
Erasing : ppas-xdb-publisher-6.2.15b-1.rhel7.x86_64 11/12
warning: /usr/ppas-xdb-6.2/etc/xdb_pubserver.conf saved as /usr/ppas-xdb-
6.2/etc/xdb_pubserver.conf.rpmsave
Erasing : ppas-xdb-libs-6.2.15b-1.rhel7.x86_64 12/12
Verifying : edb-jdbc-42.2.24.1-1.rhel7.x86_64 1/12
Verifying : edb-xdb-libs-7.0.0-1.rhel7.x86_64 2/12
Verifying : edb-xdb-publisher-7.0.0-1.rhel7.x86_64 3/12
Verifying : edb-xdb-7.0.0-1.rhel7.x86_64 4/12
Verifying : edb-xdb-console-7.0.0-1.rhel7.x86_64 5/12
Verifying : edb-xdb-subscriber-7.0.0-1.rhel7.x86_64 6/12
Verifying : ppas-xdb-subscriber-6.2.15b-1.rhel7.x86_64 7/12
Verifying : ppas-xdb-console-6.2.15b-1.rhel7.x86_64 8/12
Verifying : edb-jdbc-42.2.19.1-1.rhel7.x86_64 9/12
Verifying : ppas-xdb-libs-6.2.15b-1.rhel7.x86_64 10/12
Verifying : ppas-xdb-6.2.15b-1.rhel7.x86_64 11/12
Verifying : ppas-xdb-publisher-6.2.15b-1.rhel7.x86_64 12/12

Installed:
edb-xdb.x86_64 0:7.0.0-1.rhel7

Dependency Installed:
edb-xdb-console.x86_64 0:7.0.0-1.rhel7 edb-xdb-libs.x86_64 0:7.0.0-1.rhel7 edb-xdb-publisher.x86_64
0:7.0.0-1.rhel7
edb-xdb-subscriber.x86_64 0:7.0.0-1.rhel7

Dependency Updated:
edb-jdbc.x86_64 0:42.2.24.1-1.rhel7

Replaced:
ppas-xdb.x86_64 0:6.2.15b-1.rhel7 ppas-xdb-console.x86_64 0:6.2.15b-1.rhel7 ppas-xdb-libs.x86_64
```

```
0:6.2.15b-1.rhel7
ppas-xdb-publisher.x86_64 0:6.2.15b-1.rhel7 ppas-xdb-subscriber.x86_64 0:6.2.15b-1.rhel7

Complete!
```

At this point, the publication server and the subscription server for Replication Server 7 aren't running. The directories now contain the following:

- Replication Server 7 is installed in directory location `/usr/edb/xdb`.
- Replication Server 6.2.x remains in directory location `/usr/ppas-xdb-6.2` but with the files removed from the subdirectories such as `bin` and `lib`.
- In the `etc` subdirectory, there might be the configuration files renamed as `xdb_pubserver.conf.rpmsave` and `xdb_subserver.conf.rpmsave`.
- In the `etc/sysconfig subdirectory`, there might be the configuration file renamed as `xdbReplicationServer-62.config.rpmsave`.
- In the `/etc directory`, there might be either one or two Replication Server configuration files named `edb-repl.conf` and possibly `edb-repl.conf.rpmsave`. The file `edb-repl.conf` contains the connection and authentication information for the controller database used by the Replication Server 6.2.x publication server. The file `edb-repl.conf.rpmsave` contains only the new administrator user parameters `admin_user` and `admin_password`. Before starting the publication server and subscription server, be sure the controller database is up and running and that the `edb-repl.conf` file contains the controller database connection and authentication parameters.

7. Complete the publication server and subscription server configuration file setup.

   In the `/usr/edb/xdb/etc` directory, a new set of configuration files for Replication Server version 7 was created. These files are named `xdb_pubserver.conf` and `xdb_subserver.conf`. The new configuration files contain any new configuration options added for Replication Server 7. The old configuration files used by Replication Server version 6.2.x can be found in the `/usr/ppas-xdb-6.2/etc` directory renamed as `xdb_pubserver.conf.rpmsave` and `xdb_subserver.conf.rpmsave`.

   > **Note**
   >
   > If these files don't exist, use the ones you saved in Step 3.

   Merge the old and new configuration files so that the resulting, active configuration files contain any new Replication Server 7 configuration options as well as any nondefault settings you used with Replication Server 6.2.x and want to continue to use with Replication Server 7.

   The final set of active configuration files must be contained in the directory `/usr/edb/xdb/etc` and named `xdb_pubserver.conf` and `xdb_subserver.conf`. In the `/usr/edb/xdb/etc/sysconfig directory`, make sure the Replication Server startup configuration file `xdbReplicationServer-70.config` contains the parameter settings you want to use with Replication Server 7. See Replication Server configuration file for more information.

8. Restart the publication server and the subscription server (see Registering a publication server and Registering a subscription server).

9. Check the publication server and subscription server log files to verify that no errors occurred (see Replication Server configuration file).

10. Adjust the publication server and subscription server port numbers if necessary.

    The Replication Server 7 publication and subscription servers are installed to use the default port numbers 9051 and 9052, respectively. If the Replication Server 6.2.x replication systems used port numbers other than 9051 and 9052 for the publication and subscription servers, then make the changes to correct this inconsistency as described in Updating the publication and subscription server ports.

    If you don't need to adjust the port numbers, register the publication server and subscription server with the Replication Server console as described in Registering a publication server and Registering a subscription server). The existing replication systems appear in the replication tree of the Replication Server console.

After upgrading and before using Replication Server, you need to download a JDBC driver and create a symlink to it. See Installing a JDBC driver for more information.

You're now ready to use Replication Server 7 to create new replication systems and manage existing ones.

6.7 Uninstalling

Uninstalling the Replication Server results in the removal of the following:

- Publication server
- Subscription server
- EPRS console
- Replication Server command line interface
- EPRS replication configuration file
- Replication Server startup configuration file
- Publication server configuration file
- Subscription server configuration file

Uninstalling Replication Server doesn't remove any databases used as primary nodes, publication databases, or subscription databases.

Use the Replication Server console or the Replication Server command line interface to delete any existing single-master or multi-master replication systems before you uninstall Replication Server. Otherwise the control-schema objects created in the publication databases or primary nodes remain in those databases. You must then delete these control-schema objects manually, such as by using a SQL command line utility.

If you installed Replication Server using the Replication Server installer program invoked from Stack Builder or StackBuilder Plus, uninstall Replication Server by invoking the `uninstall-xdbreplicationserver` script.

Uninstalling an RPM package installation

If you installed Replication Server from the RPM package, uninstall it using the yum package manager.

If you installed Replication Server from the RPM package, you can uninstall a Replication Server component by using the following command:

```
yum remove package_name
```

Where `package_name` is `edb-xdb-console`, `edb-xdb-publisher`, or `edb-xdb-subscriber`.

You can remove all Replication Server components using the following command:

```
yum remove edb-xdb*`
```

For example:

```
[root@localhost yum.repos.d]# yum remove edb-xdb*
```

```
 Loaded plugins: fastestmirror, langpacks
Resolving Dependencies
--> Running transaction check
---> Package edb-xdb.x86_64 0:7.0.0-1.rhel7 will be erased
---> Package edb-xdb-console.x86_64 0:7.0.0-1.rhel7 will be erased
---> Package edb-xdb-libs.x86_64 0:7.0.0-1.rhel7 will be erased
---> Package edb-xdb-publisher.x86_64 0:7.0.0-1.rhel7 will be erased
---> Package edb-xdb-subscriber.x86_64 0:7.0.0-1.rhel7 will be erased
--> Finished Dependency Resolution

Dependencies Resolved
```

```
================================================================================================================
================================================================================
Package                                          Arch                                  Version
Repository                               Size
================================================================================================================
================================================================================
Removing:
edb-xdb                                          x86_64                                7.0.0-1.rhel7
installed                                0.0
edb-xdb-console                                  x86_64                                7.0.0-1.rhel7
installed                                3.3 M
edb-xdb-libs                                     x86_64                                7.0.0-1.rhel7
installed                                 15 M
edb-xdb-publisher                                x86_64                                7.0.0-1.rhel7
installed                                131 k
edb-xdb-subscriber                               x86_64                                7.0.0-1.rhel7
installed                                5.2 k

Transaction Summary
================================================================================================================
================================================================================
Remove  5 Packages

Installed size: 19 M
Is this ok [y/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Erasing    : edb-xdb-7.0.0-1.rhel7.x86_64
1/5
  Erasing    : edb-xdb-subscriber-7.0.0-1.rhel7.x86_64
2/5
  Erasing    : edb-xdb-console-7.0.0-1.rhel7.x86_64
3/5
  Erasing    : edb-xdb-publisher-7.0.0-1.rhel7.x86_64
4/5
  Erasing    : edb-xdb-libs-7.0.0-1.rhel7.x86_64
5/5
  Verifying  : edb-xdb-libs-7.0.0-1.rhel7.x86_64
1/5
  Verifying  : edb-xdb-publisher-7.0.0-1.rhel7.x86_64
2/5
  Verifying  : edb-xdb-7.0.0-1.rhel7.x86_64
3/5
  Verifying  : edb-xdb-console-7.0.0-1.rhel7.x86_64
4/5
  Verifying  : edb-xdb-subscriber-7.0.0-1.rhel7.x86_64
5/5

Removed:
edb-xdb.x86_64 0:7.0.0-1.rhel7              edb-xdb-console.x86_64 0:7.0.0-1.rhel7      edb-xdb-libs.x86_64
0:7.0.0-1.rhel7
edb-xdb-publisher.x86_64 0:7.0.0-1.rhel7    edb-xdb-subscriber.x86_64 0:7.0.0-1.rhel7

Complete!
```

Uninstalling a Windows installation

To uninstall Replication Server from a Windows host:

1. From the Windows Control Panel, select **Uninstall a Program**.
2. From the list of programs to uninstall or change, select the Replication Server product. Select **Uninstall/Change**.
3. Select **Yes** to confirm that you want to unistall Replication Server.
4. The Uninstallation Completed dialog box appears when the process is complete. Select **OK**.

Uninstalling in text or unattended mode

You can also uninstall Replication Server without using the user interface.

The following shows how to uninstall Replication Server in text mode.

```
 $ su root
Password:
$ ./uninstall-xdbreplicationserver --mode text
Do you want to uninstall xDB Replication Server and all of its modules? [Y/n]: y

------------------------------------------------------------------------------
Uninstall Status

 Uninstalling xDB Replication Server
 0% _______________ 50% _______________ 100%
 ########################################

Info: Uninstallation completed
Press [Enter] to continue :
```

The following shows how to uninstall Replication Server in unattended mode.

```
 $ su root
Password:
$ ./uninstall-xdbreplicationserver --mode unattended
```

7 Introduction to the Replication Server console

The Replication Server console is the user interface for configuring and managing the replication system. You can perform the same actions using the Replication Server command line interface (CLI). See Replication Server command line interface for details.

The Replication Server console window consists of the following main areas:

- Menu Bar. Menus for the replication system components
- Tool Bar. Icons for quick access to dialog boxes
- Replication Tree. Replication system components represented as nodes in an inverted tree
- Information Window. Tabbed window with information about a highlighted node in the replication tree

The options that are available on the menu bar and tool bar depend on the node selected in the replication tree. Only those options relevant to the selected node are available in the menu bar and tool bar.

The content of the information window applies to the selected node as well.

Replication Server console tool bar

The operations associated with the tool bar are described in Creating a publication and Creating a subscription for single-master replication. For multi-master replication see Creating a Publication.

> **Note**
>
> The publication server must be running to use tools relevant to publications. The subscription server must be running to use tools relevant to subscriptions.

Refresh

The **Refresh** icon is always active. Select **Refresh** if the replication tree or information window doesn't appear to display the latest information after performing an operation. Selecting **Refresh** ensures that the latest information is shown in the replication tree and in the information window.

Create Publication

The **Create Publication** icon is active when a Publication Database node is selected in the replication tree.

Publication management

The **Remove Publication**, **Add Publication Tables**, and **Remove Publication Tables** icons are active when a Publication node is selected in the replication tree.

Create Subscription

The **Create Subscription** icon is active when a Subscription Database node is selected in the replication tree.

Subscription management

The **Remove Subscription**, **Snapshot**, **Synchronize**, **Configure Schedule**, and **Remove Schedule** icons are active when a Subscription node is selected in the replication tree.

Saving server login information

When you use the EPRS Replication Console to create a replication system, you need to register a publication server and a subscription server. During this process, you can save the server's login information.

The following discussion applies to both publication servers and subscription servers. These are generically referred to as servers.

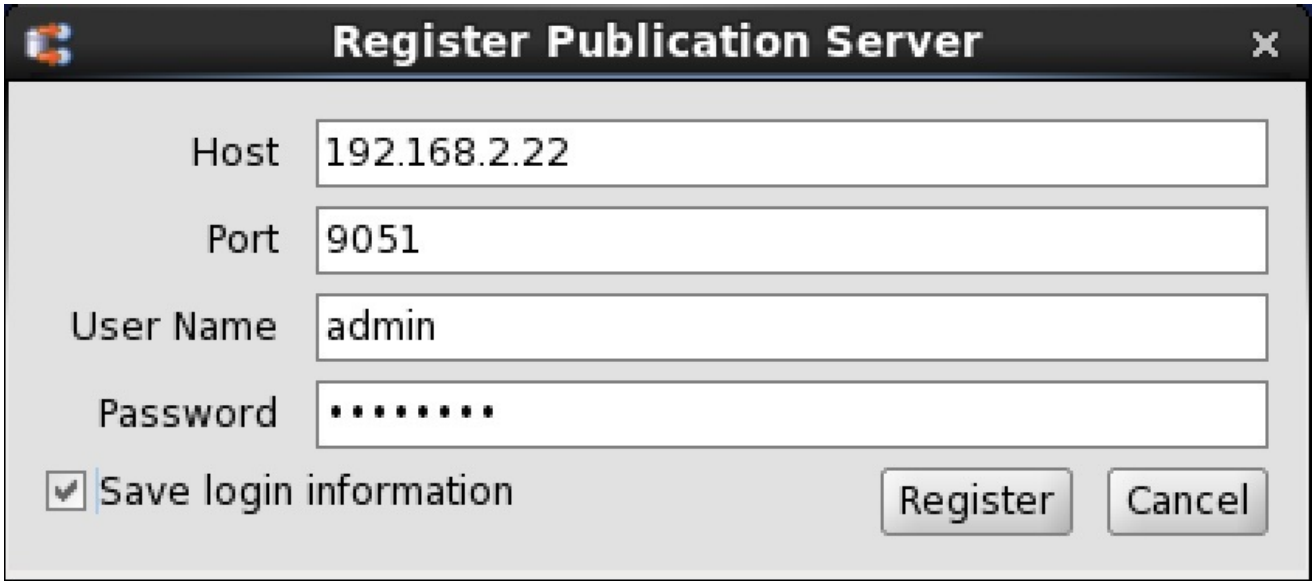
Server login file

If you choose to save the login information, the server's network location (IP address and port number), admin user name, and password are stored in a server login file in a hidden location under the home directory of the operating system account with which you opened the Replication Server console. See Installation details for the location of this file.

The following shows the Register Publication Server dialog box where the option to save login information is presented as a check box. In this example, the following fields are saved in the server login file for this publication server if the admin user name and password validation are successful:

- `192.168.2.22` entered in the **Host** field
- `9051` entered in the **Port** field
- `admin` entered in the **User Name** field
- An encrypted form of the password entered in the **Password** field

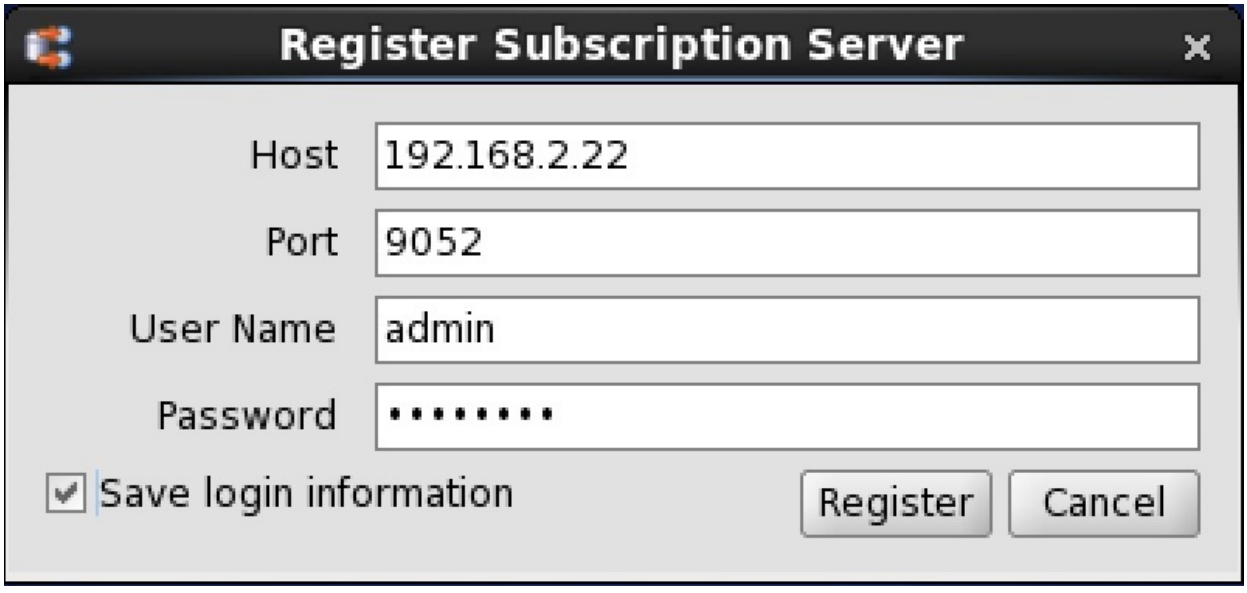
The values for **User Name** and **Password** that you enter are validated against the admin user name and password in the Replication Server configuration file residing on host 192.168.2.22. The admin user name and password must successfully authenticate before you can register the publication server and save the publication server's login information in the server login file. See Replication Server configuration file for more information.



See Registering a publication server for more information on the purpose of these fields and the process of registering a publication server.

The following shows the Register Subscription Server dialog box. In this example, the following fields are saved in the server login file for this subscription server if the admin user name and password validation are successful:

- `192.168.2.22` entered in the **Host** field
- `9052` entered in the **Port** field
- `admin` entered in the **User Name** field
- An encrypted form of the password entered in the **Password**



See Registering a subscription server for more information on the purpose of these fields and the process of registering a subscription server.

Saving server login information gives you the convenience of immediate access to the publication server and any of its subordinate publications or access to the subscription server and any of its subordinate subscriptions. Wen you open the Replication Server console, the Publication Server nodes of saved publication servers immediately appear in the replication tree, allowing you to perform administrative tasks on its subordinate publications.

Similarly, the Subscription Server nodes of saved subscription servers immediately appear in the replication tree, allowing you to perform administrative tasks on its subordinate subscriptions.

If you don't save server login information, the server nodes aren't visible in the replication tree. You must reenter the server's network location, admin user name, and password, registering the server each time you open the Replication Server console.

> **Note**
>
> Each operating system account on a given host has its own server login file. Thus, the servers that are saved and appear in the Replication Server console when opened is independently determined for each operating system account.

Security risks of saved server login information

The security risk associated with saving server login occurs if unauthorized persons gain access to your operating system account. They can then potentially open the Replication Server console on your host using your operating system account.

If the login information of publication servers or subscription servers is saved, the corresponding Publication Server nodes or Subscription Server nodes immediately appear in the Replication Server console with no request for authentication information.

This allows an unauthorized person to perform any operation on the exposed publications and subscriptions, including the potential to delete the replication system.

> **Note**
>
> The publication database and subscription database can't be deleted, but unauthorized replications can be forced.

Thus, it is important that operating system accounts are secure on hosts that have access to a Replication Server console and a replication system.

8 Single-master replication operation

You can configure and run Replication Server for single-master replication systems.

For configuration and management of your replication system, the Replication Server console interface is used to show the steps to use. You can perform the same steps from the operating system command line using the Replication Server command line interface (CLI). The commands of the Replication Server CLI are described in Replication Server command line interface.

8.1 Prerequisite steps

You must take certain steps to prepare the host environments as well as the publication and subscription database servers before beginning the process of building a single-master replication system.

8.1.1 Setting heap memory size for the publication and subscription servers

The publication server and the subscription server are configured to run with a default set of heap size parameters. The default settings for 32-bit platforms and the default settings for 64-bit platforms are set by the parameter `JAVA_HEAP_SIZE` when Replication Server is installed.

Configure this parameter in the Replication Server startup configuration file. See Replication Server startup configuration file.

The following is an example of the Replication Server startup configuration file.

```
#!/bin/sh

JAVA_EXECUTABLE_PATH="/usr/bin/java"
JAVA_MINIMUM_VERSION=1.8
JAVA_BITNESS_REQUIRED=64
JAVA_HEAP_SIZE="-Xms2048m -Xmx4096m"
PUBPORT=9051
SUBPORT=9052
```

On a 32-bit system, the initial heap size is set to 128 megabytes (-Xms128m) and the maximum limit is set to 512 megabytes (-Xmx512m). On a 64-bit system the initial heap size is 256 megabytes (-Xms256m) and the maximum limit is 1536 megabytes (-Xmx1536m).

Typically, these values can handle the average workloads. However, depending on the average row size and pending backlog of replication updates, it might help to increase the default heap size settings.

You can change the default values by changing the `JAVA_HEAP_SIZE` parameter setting in the Replication Server startup configuration file. Be sure to restart the publication server and the subscription server after making such changes. See Registering a publication server and Registering a subscription server.

The heap size value must conform to the available RAM on the host running the publication server or subscription server. The basic guideline is for the maximum heap size not to exceed 25% of the total RAM size.

If both the publication server and subscription server are running on the same host, then the minimum and recommended RAM capacity are as follows:

- **Minimum RAM Size**. For a 32-bit system, use 4 gigabytes; for a 64-bit system use 8 gigabytes.
- **Recommended RAM Size**. For a 32-bit system, use 8 gigabytes; for a 64-bit system use 16 gigabytes.

By default, both the publication server and subscription server are started and both are required for single-master replication systems. However, if you're only configuring and using multi-master replication systems, then you don't need the subscription server. In such cases, stop the subscription server to avoid redundant use of memory.

If both the publication server and the subscription server are running on the same host, then each server reserves its own heap buffer. Thus, the total heap size for both the publication and subscription servers, obtained by adding the maximum heap size for both servers, must comply with the available RAM on the host.

Tuning heap size and configuration parameters for larger rows

When one or more publication tables contain a large size column, for example, `XMLType` (Oracle/EPAS data type), you must adjust specific parameters to avoid Out Of Memory errors. The `XMLType` column is stored in large objects (LOBs). The LOB storage maintains content accuracy to the original XML. So it retains and stores all the white spaces present in the XML. This data occupies large space when it is loaded and held in memory by Replication Server as part of the replication process.

Tune the following parameters to reduce the data maintained in the memory for `XMLType` and other large-size columns:

- Increase HEAP size between 4GB to 8GB depending on the maximum size of large column:
  - 4GB for column size below 30MB
  - 6GB for column size between 30 and 100 BM
  - 8GB for column size > 100 MB

- Set the configuration parameter `txSetMaxSize` to a lower value (10 to 50) depending on the average size of row data. For a large column (>100MB), set txSetMaxSize to less than or equal to 5.
- Set the configuration parameter `syncBatchSize` to a lower value (4 to 10 rows) depending on the size of the column data. For a very large column (>100MB), set `syncBatchSize` to less than 4.

> **Note**
>
> Higher values of `txSetMaxSize` and `syncBatchSize` boost the performance of the replication process. However, increasing it to a relatively larger value might result in an Out of Memory error. Tune these values based on the column size.

8.1.2 Enabling synchronization replication with the log-based method

This information applies only to Postgres database servers of version 9.4 and later. If you plan to use the log-based method of synchronization replication with any publication database running under the Postgres database server, the following configuration parameter settings are required in the configuration file, `postgresql.conf`, of the Postgres database server:

- `wal_level`. Set to `logical`.
- `max_wal_senders`. Specifies the maximum number of concurrent connections (that is, the maximum number of simultaneously running WAL sender processes). Set, at minimum, to the number of SMR publication databases on this database server that use the log-based method. In addition, if MMR primary nodes are to run on this database server, also add the number of MMR primary nodes that use the log-based method.
- `max_replication_slots`. Specifies the maximum number of replication slots. Set, at minimum, to the number of SMR publication databases on this database server that use the log-based method. In addition, if MMR primary nodes run on this database server with the log-based method, see Replication origin for information on the additional number of replication slots required.

See Synchronization replication with the log-based method.

You must restart the Postgres database server after altering any of these configuration parameters.

In addition, the `pg_hba.conf` file requires an entry for each publication database user of publication databases that use the log-based method. Include these database users as a replication database user in the `pg_hba.conf` file. See Postgres server authentication for additional information.

8.1.3 Enabling access to the database servers

Use these configuration steps to use Replication Server on various types of database servers.

No special steps are required to enable access to a Postgres database server.

Enabling access to Oracle

> **Note**
>
> The instructions in this section apply only if Oracle is used as the publication or subscription database.

An Oracle JDBC driver `jar` file, such as `ojdbc8.jar`, must be accessible to the Java virtual machine (JVM) on the host running the publication server and the subscription server. If the publication server and subscription server are running on separate hosts, the Oracle JDBC driver must be accessible to the JVM on each host. Use Oracle JDBC driver version ojdbc8 or later.

1. Download the Oracle JDBC driver, for example, `ojdbc8.jar`, from the Oracle download site to the host that runs the publication server.

2. Copy the file `ojdbc8.jar` to the directory `XDB_HOME/lib/jdbc.`

```
 $ su root
Password:
$ cd /usr/edb/xdb/lib/jdbc
$ cp /home/user/Downloads/ojdbc8.jar .
$ ls -l
total 4032
-rw-rw-r-- 1 root root  355655 Jan 25 02:38 edb-jdbc14.jar
-rw-rw-r-- 1 root root  716209 Jan 25 02:38 edb-jdbc17.jar
-rw-rw-r-- 1 root root  317816 Jan 25 02:38 jtds-1.3.1.jar
-rw-r--r-- 1 root root 2091137 Jan 28 16:45 ojdbc8.jar
-rw-rw-r-- 1 root root  642809 Jan 25 02:38 postgresql-9.4-1201.jdbc4.jar
```

> **Note**
>
> You can also copy the `ojdbc8.jar` file to the `jre/lib/ext` subdirectory of the location where you installed your Java runtime environment.

> **Note**
>
> Make sure to set the ODBC driver permission to a minimum of `644`.

> **Note**
>
> Make sure to copy `xdb6.jar` along with `ojdbc8.jar` at `/usr/edb/xdb/lib/jdbc/` if one or more tables in Oracle Publication contains an `XMLType` column when using Oracle to EDB Postgres Advanced Server/PostgreSQL replication.

3. If the subscription server is running on a host different from the publication server, repeat steps 1 and 2 for the subscription server host.

Enabling access to SQL Server

> **Note**
>
> These instructions apply only if you're using SQL Server as the publication or subscription database.

Replication Server now supports both Microsoft JDBC and jTDS JDBC drivers. A new parameter option `useJtdsDriver` is added with the default value of `true` in both `xdb_pubserver.conf` and `xdb_subserver.conf` configuration files. This option when true uses jTDS driver for SQL Server connection. To use a Microsoft specific JDBC driver, uncomment this option and specify the value `false` in both the configuration files.

The MS SQL JDBC driver jar file `mssql-jdbc-10.2.1.jre8.jar` or the jTDS JDBC driver jar file `jtds-1.3.1.jar` must be accessible to the Java virtual machine (JVM) on the host running the publication server and the subscription server depending on which driver is being used. If the publication server and subscription server are running on separate hosts, the Microsoft JDBC or jTDS JDBC driver must be accessible to the JVM on each host.

When you install Replication Server, there is no `jtds-1.3.1.jar` or `mssql-jdbc-10.2.1.jre8.jar` placed in the directory `XDB_HOME/lib/jdbc` by the Replication Server installation. The user has to copy the required JDBC driver and rename them as `mssql-jdbc.jar` and `jtds.jar`. Alternatively, the user can create a symlink in `XDB_HOME/lib/jdbc` for the required JDBC driver. The symlink names should be as below:

```
 mssql-jdbc.jar
jtds.jar
```

> **Note**

> For windows, we recommended that the user copy the required JDBC jars to the `XDB_HOME/lib/jdbc` folder and rename them.

1. Be sure SQL Server Authentication mode is enabled on your SQL Server database engine. SQL Server Authentication mode allows the use of SQL Server logins such as the built-in system administrator login, sa.

   Using the default settings for SQL Server installation, only Windows Authentication mode is enabled, which uses the accounts of the Windows operating system for authentication.

   To permit SQL Server Authentication mode, you must change the authentication mode to Mixed Mode Authentication, which permits both Windows Authentication and SQL Server Authentication.

   You can do this using SQL Server Management Studio. Refer to the SQL Server Management Studio documentation.

2. Be sure SQL Server is accepting TCP/IP connections. In the SQL Server Configuration Manager, under SQL Server Network Configuration, be sure the TCP/IP protocol for the SQL Server instance is set to `Enabled`. The typical, default SQL Server instance names are `MSSQLSERVER` or `SQLEXPRESS`.

3. (Required only for a SQL Server publication database) Be sure SQL Server Agent is enabled and running. SQL Server Agent is a Windows service that controls job scheduling and execution with SQL Server.

   Replication Server uses SQL Server Agent for certain operations such as for scheduled shadow table history cleanup (see Scheduling shadow table history cleanup).

   You can start SQL Server Agent by using SQL Server Configuration Manager. Refer to the SQL Server Configuration Manager documentation.

8.1.4 Preparing the publication database

Prepare the database that contains tables and views that will become members of publications.

The tables and views to use for any given publication must all reside in the same database. This database becomes the publication database of that publication. You need a publication database user name with the following characteristics:

- The publication database user can connect to the publication database.
- The publication database user has the privileges to create control schema objects to store metadata used for controlling and tracking the replication process.
- The publication database user can read the tables and views that will become members of publications.
- For publications that use synchronization replication with the trigger-based method, the publication database user can create triggers on the publication tables. (For Oracle, the publication database user must have trigger creation privilege even for snapshot-only publications, although triggers are created only for publications using synchronization replication.)

The examples used are based on the following:

- The publication database user name is pubuser.
- The tables and view used in publications reside in a schema named `edb`.
- Three tables named `dept`, `emp`, and `jobhist` are members of schema `edb`.
- One view named `salesemp` is a member of schema `edb`. This view is a `SELECT` statement over the `emp` table.
- The Oracle system identifier (SID) of the publication database is `xe`. The SQL Server publication database name is `edb`. The Postgres publication database name is `edb`. (The cases of Oracle as the publication database, SQL Server as the publication database, and Postgres as the publication database are presented with examples in this section.)

Oracle publication database

> **Note**

> (For Oracle 12c): The Oracle 12c multitenant architecture introduces the concept of the container database (CDB), which can contain multiple pluggable databases (PDBs). You can use a pluggable database as a publication database or a subscription database in a single-master replication system.

Oracle 12c still supports the use of a single database referred to as a non-container database (non-CDB) that is compatible with Oracle versions prior to 12c. You can also use an Oracle 12c non-container database as a publication database or a subscription database in a single-master replication system.

The setup instructions for using an Oracle 12c publication database or subscription database are the same as for previous Oracle versions. Any special distinctions are indicated by a note in the instructions.

1. Create a database user name for the publication database user. The publication database user name must have a password, and it must be able to create a database session. The publication database user becomes the owner of the control schema objects that are created in the publication database to track, control, and record the replication process and history.

   > **Notes**
   >
   > - (For Oracle 12c Pluggable Database): The publication database user can be an Oracle local user or a common user. The local user exists within and has access to only a single, user-created pluggable database (PDB), which is to be used as the publication database. Common user names typically begin with `C##` or `c##` and can access multiple pluggable databases.
   > - (For Oracle 12c Pluggable Database): Creation and granting of privileges for a local user must be done while connected to the pluggable database to be used as the publication database. Creation of a common user must be done within the Oracle 12c root container `CDB$ROOT`. Granting of privileges to the common user must be done while connected to the pluggable database to be used as the publication database.
   > - (For Oracle 12c Non-Container Database): Creation and granting of privileges to the publication database user are performed in the same manner as for Oracle versions prior to 12c.
   > - If you enable flashback functionality for the Oracle published table, you must also give the publication database user flashback privileges for the table.
   >
   > ```
   > GRANT flashback ON schema_name.table_name to pubuser;
   > ```

   When creating the publication database definition, enter the publication database user name in the Publication Service – Add Database dialog box (see Adding a publication database).

   ```
   CREATE USER pubuser IDENTIFIED BY password;
   GRANT CONNECT TO
   pubuser;
   ```

2. Grant the privileges needed to create the control schema objects. The control schema objects are created in the schema owned by, and with the same name as, the publication database user. That is, the publication database user's schema is the control schema for an Oracle publication database.

   ```
   GRANT RESOURCE TO
   pubuser;
   ```

3. Grant the privileges required to create triggers on the publication tables. Grant the `CREATE ANY TRIGGER` privilege to the publication database user.

   ```
   GRANT CREATE ANY TRIGGER TO
   pubuser;
   ```

4. Grant the privileges required to lock publication tables when creating triggers. Grant the `LOCK ANY TABLE` privilege to the publication database user.

   ```
   GRANT LOCK ANY TABLE TO
   pubuser;
   ```

5. (For Oracle 12c onward) Grant the privileges required to access tablespaces. Grant the `GRANT UNLIMITED TABLESPACE` privilege to the publication database user. This requirement applies to both a pluggable database and a non-container database.

```
GRANT UNLIMITED TABLESPACE TO
pubuser;
```

6. (For Oracle 19c onward) The `CREATE JOB` privilege is needed for the publication database user to schedule a job.

   This requirement applies to both pluggable and non-container databases.

   ```
   GRANT CREATE JOB TO
   pubuser;
   ```

7. The publication database user must be able to read the tables and views to include in publications.

   ```
   GRANT SELECT ON edb.dept TO
   pubuser;
   GRANT SELECT ON edb.emp TO
   pubuser;
   GRANT SELECT ON edb.jobhist TO
   pubuser;
   GRANT SELECT ON edb.salesemp TO
   pubuser;
   ```

8. (Optional) Create one or more `group` roles containing the required privileges to access the tables and views of the publications needed by application users.

   Using roles is convenient if you want to add new application users who need privileges to select, insert, update, or delete from any of the publication tables. A role containing the required privileges can then be granted to the new users instead of granting each privilege to each user.

   The following example shows creating the role and granting the privileges on the publication tables to the role:

   ```
   CREATE ROLE appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
   appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
   appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.salesemp TO appgroup;
   ```

   The following example shows creating a new user and granting the new role to the user:

   ```
   CREATE USER appuser IDENTIFIED BY password;
   GRANT CREATE SESSION TO
   appuser;
   GRANT appgroup TO
   appuser;
   ```

SQL Server publication database

In SQL Server, an application gains access to the database server by supplying a SQL Server login and its associated password.

When an application connects to a particular database, the application assumes the identity and privileges of a database user that was defined in that database. The database users in any given database are independent of database users in other databases with respect to their properties such as their role memberships and privileges. In fact, you can define the same database user name in more than one database, each with its own distinct properties.

In each database, you can map a database user to a SQL Server login. When an application connects to a database using a SQL Server login to which a database user is mapped, the application assumes the identity and privileges of that database user.

When using a SQL Server database as the publication database, you must define a number of database users and map them to a SQL Server login according to the following rules:

- A SQL Server login must exist for the publication server to use to connect to SQL Server. You specify the SQL Server login and password when creating the publication database definition.
- In the publication database, a database user must exist who is the creator and owner of the control schema objects. Map this database user to the SQL Server login used by the publication server.
- A schema must exist to contain certain control schema objects. The database user described in the preceding bullet point must either own this schema or have certain privileges on it so that they can create and update the control schema objects in this schema. This schema is one physical schema component of the overall control schema, and you must also define it as the default schema of that database user. The other physical schemas comprising the overall control schema are always created by the publication server as `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler`.
- The SQL Server database users that update the data in the application tables to replicate must have certain privileges on the control schema objects. When an update on a replicated table occurs, a trigger activates that accesses and updates certain control schema objects. Grant the appropriate privileges to SQL Server database users who require update access to the application tables.
- A database user must exist in the `msdb` database that is mapped to the SQL Server login used by the publication server. This database user must have certain privileges to execute jobs in the dbo schema of the `msdb` database. (SQL Server Agent uses the `msdb` database to schedule alerts and jobs. SQL Server Agent runs as a Windows service.)

This example uses the following SQL Server login, database users, and mappings to comply with these rules:

- The publication tables reside in database `edb`.
- The database user owning the schema containing the publication tables and the publication tables, themselves, is edb.
- The SQL Server login used by the publication server to connect to SQL Server is pubuser.
- The database user owning the control schema objects and mapped to SQL Server login pubuser in database `edb` is pubuser.
- The control schema used to contain certain control schema objects created by the publication server is pubuser. Other control schema objects are always created in `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler`.
- The database user mapped to SQL Server login pubuser in database `msdb` is pubuser_msdb.

> **Note**
>
> The `sqlcmd` utility program is used to execute the SQL statements in these examples. The `USE` command establishes the database to which to apply the subsequent statements. The `GO` command executes the preceding SQL statements as a batch. Placement of the `GO` command in a stream of SQL statements is sometimes important, depending on the SQL statements.

1. Create a SQL Server login for the Replication Server publication database user. The login must have a password.

   When creating the publication database definition, enter the SQL Server login in the Publication Service – Add Database dialog box (see Adding a publication database).

   ```
   USE tempdb;
   GO
   CREATE LOGIN pubuser WITH PASSWORD = 'password';
   GO
   ```

2. Create the database user and its required privileges for job scheduling in database `msdb`:

   ```
   USE msdb;
   GO
   CREATE USER pubuser_msdb FOR LOGIN
   pubuser;
   GO
   GRANT EXECUTE ON SCHEMA :: dbo TO
   pubuser_msdb;
   GRANT SELECT ON SCHEMA :: dbo TO
   pubuser_msdb;
   GO
   ```

3. Create the database user for the control schema object creation and ownership. Create the control schema objects in the publication database to track, control, and record the replication process and history. This example assumes some of the control schema objects are created in the schema named `pubuser`.

   > **Note**

> The schema name you specify in the `WITH DEFAULT_SCHEMA` clause must be the schema you choose in Step 5. This schema doesn't have to exist before you use it in the `CREATE USER FOR LOGIN WITH DEFAULT_SCHEMA` statement.

```
USE edb;
GO
CREATE USER pubuser FOR LOGIN pubuser WITH DEFAULT_SCHEMA =
pubuser;
GO
```

> **Note**
>
> The remaining steps assume that the commands are given in the publication database (that is, the `USE edb` command was previously given to establish the publication database `edb` as the current database.)

4. Grant the database level privileges needed by the publication database user to create the control schema objects.

```
GRANT CREATE TABLE TO
pubuser;
GRANT CREATE PROCEDURE TO
pubuser;
GRANT CREATE FUNCTION TO
pubuser;
GRANT CREATE SCHEMA TO
pubuser;
GO
```

5. Choose the control schema for some of the control schema objects.

   To create the control schema objects in a new schema owned by the publication database user and created exclusively for this purpose (recommended approach) use the following command:

```
CREATE SCHEMA pubuser AUTHORIZATION
pubuser;
GO
```

   Alternatively, to create the control schema objects in an existing schema, such as in the same schema containing the publication tables (schema `edb` in this example), use the following commands:

```
GRANT ALTER   ON SCHEMA :: edb TO
pubuser;
GRANT EXECUTE ON SCHEMA :: edb TO
pubuser;
GRANT SELECT  ON SCHEMA :: edb TO
pubuser;
GRANT INSERT  ON SCHEMA :: edb TO
pubuser;
GRANT UPDATE  ON SCHEMA :: edb TO
pubuser;
GRANT DELETE  ON SCHEMA :: edb TO
pubuser;
GO
```

6. Grant the privileges required to create triggers on the publication tables. The publication database user must have the ALTER privilege on the publication tables.

```
GRANT ALTER ON edb.dept TO
pubuser;
GRANT ALTER ON edb.emp TO
pubuser;
GRANT ALTER ON edb.jobhist TO
pubuser;
GO
```

7. The publication database user must be able to read the tables and views to include in publications.

```
GRANT SELECT ON edb.dept TO
pubuser;
GRANT SELECT ON edb.emp TO
pubuser;
GRANT SELECT ON edb.jobhist TO
pubuser;
GRANT SELECT ON edb.salesemp TO
pubuser;
GO
```

8. (Optional) Create one or more `group` roles containing the required privileges to access the tables and views of the publications needed by application users.

> **Note**
>
> You can create these roles only after creating the SQL Server publication database definition using the Replication Server console or CLI. For example, see Adding a publication database for how to use the Replication Server console.

Using roles is convenient if you want to add new application users who need privileges to select, insert, update, or delete from any of the publication tables. You can then grant a role containing the required privileges to the new users instead of granting each privilege to each user.

In addition to privileges on the publication tables, any user performing an insert, update, or delete operation on any of the publication tables requires privileges to certain control schema objects of the publication.

The following example shows creating the role `appgroup` and granting privileges on the publication tables to the role. The example assumes that, in Step 5, schema pubuser was chosen as the control schema to store some of the control schema objects.

```
CREATE ROLE appgroup AUTHORIZATION
edb;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
appgroup;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO appgroup;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
appgroup;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.salesemp TO appgroup;
GRANT EXECUTE ON SCHEMA :: _edb_replicator_pub TO
appgroup;
GRANT SELECT  ON SCHEMA :: _edb_replicator_pub TO
appgroup;
GRANT INSERT  ON SCHEMA :: _edb_replicator_pub TO
appgroup;
GRANT UPDATE  ON SCHEMA :: _edb_replicator_pub TO
appgroup;
GRANT INSERT  ON SCHEMA :: pubuser TO appgroup;
GO
```

The following example shows creating a new login and database user and adding the database user as a member of the role to inherit its privileges:

```
CREATE LOGIN applogin WITH PASSWORD = 'password', DEFAULT_DATABASE =
edb;
CREATE USER appuser FOR LOGIN applogin WITH DEFAULT_SCHEMA =
edb;
EXEC sp_addrolemember @rolename = 'appgroup', @membername =
'appuser';
GO
```

> **Note**
>
> Granting privileges to individual users: As previously described, each application database user that modifies the data in any of the publication tables needs certain privileges on the publication tables and the control schema objects. Using a group role for this purpose as described earlier in this step helps simplify this process.

You can grant individual database users the privileges to access the publication tables and the controls schema objects in a similar fashion.

The following example shows creating a new login and database user and granting the privileges on the publication tables and the control schema objects to the user:

```
CREATE LOGIN newlogin WITH PASSWORD = 'password', DEFAULT_DATABASE =
 edb;
CREATE USER newuser FOR LOGIN newlogin WITH DEFAULT_SCHEMA =
edb;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.salesemp TO
newuser;
GRANT EXECUTE ON SCHEMA :: _edb_replicator_pub TO
newuser;
GRANT SELECT  ON SCHEMA :: _edb_replicator_pub TO
newuser;
GRANT INSERT  ON SCHEMA :: _edb_replicator_pub TO
newuser;
GRANT UPDATE  ON SCHEMA :: _edb_replicator_pub TO
newuser;
GRANT INSERT  ON SCHEMA :: pubuser TO
newuser;
GO
```

> **Note**
>
> Instead of using the preceding statements, which grant privileges at the schema level, you can issue a more granular level of privileges at the database object level using the following statements:

```
CREATE LOGIN newlogin WITH PASSWORD = 'password', DEFAULT_DATABASE =
 edb;
CREATE USER newuser FOR LOGIN newlogin WITH DEFAULT_SCHEMA =
edb;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.salesemp TO
newuser;
GRANT INSERT  ON pubuser.rrst_edb_dept TO
newuser;
GRANT INSERT  ON pubuser.rrst_edb_emp TO
newuser;
GRANT INSERT  ON pubuser.rrst_edb_jobhist TO
newuser;
GO
```

In addition, depending on the version of SQL Server, grant the following privileges. Using this approach, however, requires you to issue additional privileges for each application table that is later added to the publication.

For SQL Server 2014:

```
GRANT UPDATE ON _edb_replicator_pub.rrep_tx_seq TO
 newuser;
GRANT UPDATE ON _edb_replicator_pub.rrep_txset_seq TO
newuser;
GRANT UPDATE ON _edb_replicator_pub.rrep_common_seq TO
newuser;
```

```
GO
```

Postgres publication database

When creating the publication database definition, you must specify a database user name that has the following characteristics:

- The database user can connect to the publication database.
- The database user has superuser privileges. Superuser privileges are required because the database configuration parameter `session_replication_role` is altered by the database user to replica for snapshot operations involving replication of the control schema from one publication database to another.
- The database user must have the ability to modify the system catalog tables to disable foreign key constraints on the control schema tables for snapshot operations involving replication of the control schema from one publication database to another. See Disabling foreign key constraints for snapshot replications for more information on this requirement.

1. Create a database superuser for the publication database user. The publication database user name must have a password, and it must have the ability to create a database session. The publication database user becomes the owner of the control schema objects created in the publication database to track, control, and record the replication process and history.

   When creating the publication database definition, enter the publication database user name in the Publication Service – Add Database dialog box. See Adding a publication database.

   ```
   CREATE ROLE pubuser WITH LOGIN SUPERUSER PASSWORD 'password';
   ```

2. (Optional) Create one or more `group` roles containing the required privileges to access the tables and views of the publications needed by application users.

   > **Note**
   >
   > The process described in this step applies to Postgres publications in both single-master and multi-master replication systems.

   Using roles is convenient if you want to add new application users who need privileges to select, insert, update, or delete from any of the publication tables. You can then grant a role containing the required privileges to the new users instead of granting each privilege to each user.

   Any user performing an insert, update, or delete operation on any of the publication tables requires privileges on the publication tables and its schema as well as to certain control schema objects of the publication. These control schema objects reside under schema `_edb_replicator_pub`.

   The following example shows creating the role `appgroup` and granting privileges on the publication tables to the role.

   ```
   CREATE ROLE appgroup;
   GRANT USAGE ON SCHEMA edb TO
   appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
   appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
   appgroup;
   GRANT SELECT, INSERT, UPDATE, DELETE ON edb.salesemp TO appgroup;
   ```

   In addition, for the log-based method of synchronization replication, if the `TRUNCATE` command is permitted on the publication tables, grant the following privileges:

   ```
   GRANT TRUNCATE ON edb.dept TO
   appgroup;
   GRANT TRUNCATE ON edb.emp TO appgroup;
   GRANT TRUNCATE ON edb.jobhist TO
   appgroup;
   ```

Also, for the log-based method of synchronization replication for use of the `TRUNCATE` command, grant the following privileges after creating the publication database definition. See Adding a publication database for information on creating the publication database definition for a single-master replication system. For a multi-master replication system, see Adding the primary definition node.

```
GRANT USAGE ON SCHEMA _edb_replicator_pub TO
appgroup;
GRANT INSERT ON _edb_replicator_pub.rrep_wal_events_queue TO appgroup;
```

Finally, grant the group role to the desired database users. The following example shows creating a new user and granting the role to the user:

```
CREATE ROLE appuser WITH LOGIN PASSWORD 'password';
GRANT appgroup TO
appuser;
```

> **Note**
>
> Granting privileges to roles after creating a publication: Create roles for containing publication table privileges before you create the publication. See Adding a publication for information on creating a publication for a single-master replication system. For a multi-master replication system, see Adding a publication.)

When you create the publication, the privileges that are granted on the publication tables to roles that exist at the time are applied to the control schema objects for those roles. So for the preceding example, the privileges required on the control schema objects for any publication created using `edb.dept`, `edb.emp`, `edb.jobhist`, or `edb.salesemp` are granted to role `appgroup` when you create that publication.

If, however, you create a role after the publication is created, you must explicitly grant the needed privileges on the publication tables and control schema objects to the new role.

When using the trigger-based method of synchronization replication, grant a role the following privileges on the control schema objects:

- `USAGE` privilege on schema `_edb_replicator_pub`.
- `USAGE` privilege on sequence `rrep_tx_seq`.
- `INSERT` privileges on the shadow tables corresponding to publication tables in which the role will be inserting, updating, or deleting rows. Shadow tables follow the naming convention `rrst_schema_table`. Shadow tables exist only when using the trigger-based method of synchronization.

The following example shows creating a new role and granting the privileges on the publication tables and the control schema objects to the role for the trigger-based method of synchronization replication:

```
CREATE ROLE newuser WITH LOGIN PASSWORD 'password';
GRANT USAGE ON SCHEMA edb TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
newuser;
GRANT USAGE ON SCHEMA _edb_replicator_pub TO
newuser;
GRANT USAGE ON SEQUENCE _edb_replicator_pub.rrep_tx_seq TO
newuser;
GRANT INSERT ON _edb_replicator_pub.rrst_edb_dept TO
newuser;
GRANT INSERT ON _edb_replicator_pub.rrst_edb_emp TO
newuser;
GRANT INSERT ON _edb_replicator_pub.rrst_edb_jobhist TO
newuser;
```

When using the log-based method, a role needs access to the publication tables and to certain control schema objects as well under certain circumstances.

When using the log-based method of synchronization replication, you must grant a role the following privileges on the control schema objects if the

role is permitted to use the TRUNCATE command on the publication tables:

- `USAGE privilege on schema _edb_replicator_pub`
- `INSERT` privilege on table `_edb_replicator_pub.rrep_wal_events_queue`

The following example shows creating a new role and granting the privileges on the publication tables to the role for the log-based method of synchronization replication:

```
CREATE ROLE newuser WITH LOGIN PASSWORD 'password';
GRANT USAGE ON SCHEMA edb TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.dept TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.emp TO
newuser;
GRANT SELECT, INSERT, UPDATE, DELETE ON edb.jobhist TO
newuser;
```

In addition, if the `TRUNCATE` command is permitted on the publication tables, grant the following privileges:

```
GRANT TRUNCATE ON edb.dept TO
 newuser;
GRANT TRUNCATE ON edb.emp TO
newuser;
GRANT TRUNCATE ON edb.jobhist TO
newuser;
GRANT USAGE ON SCHEMA _edb_replicator_pub TO
newuser;
GRANT INSERT ON _edb_replicator_pub.rrep_wal_events_queue TO
newuser;
```

8.1.5 Preparing the subscription database

You must prepare a database to use as a subscription database.

Replicate the tables and views in a given publication to the same database. The database you replicate to is called the subscription database. Create a subscription database user name with the following characteristics:

- The subscription database user can connect to the subscription database.
- The subscription database user has the privileges to create database objects for the replicated tables and views from publications.
- The subscription database user has the privileges needed to execute the `TRUNCATE` command on the replicated tables.

Postgres subscription database

Choose or create a database passworded user name to serve as the subscription database user. The subscription database user becomes the owner of the replicated database objects.

When creating the subscription database definition, enter the subscription database user name in the Subscription Service – Add Database dialog box (see Adding a subscription database).

When choosing the subscription database user name, you can either:

- For the subscription database user name, use the Postgres user name postgres created when you installed PostgreSQL (enterprisedb for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode). If you choose this option, skip Step 1 and proceed to Step 2.
- Create a new subscription database user name.

1. Create a superuser as the subscription database user.

```
CREATE ROLE subuser WITH LOGIN SUPERUSER PASSWORD '<password>';
```

2. Create or choose the subscription database.

   The names of the schemas containing the publication tables and views become the names of the Postgres schemas for the subscription tables. The subscription server creates these schemas in the subscription database when you create the subscription. If schemas with these names already exist in the subscription database, the existing schemas store the subscription tables.

   For a SQL Server publication database: If the schema containing the publication tables and views in SQL Server is named `dbo`, then the subscription server creates a schema named `dbo_sql` in the Postgres subscription database for the subscription tables. (Schema `dbo` is a special reserved schema in Postgres.)

   The existing schemas can't contain any tables or views with the same names as the publication tables and views. The subscription server returns an error if there are already identically named tables or views. You must delete or rename these tables and views before the subscription is created.

   You can create a new subscription database owned by the subscription database user subuser:

```
CREATE DATABASE subdb OWNER
subuser;
```

Oracle subscription database

1. (Optional) If you don't have an existing database that you want to use as your subscription database, create a new database. This step can be fairly complicated. Refer to the appropriate Oracle documentation for performing this task.

2. Create a database user name for the subscription database user. The subscription database user name must have a password, and it must have the ability to create a database session. The subscription database user becomes the owner of the replicated database objects.

   > **Note**
   >
   > (For Oracle 12c pluggable database): The subscription database user can be an Oracle local user or a common user. The local user exists in and has access to only a single, user-created pluggable database (PDB) to use as the subscription database. Common user names typically begin with C## or c## and can access multiple pluggable databases.

   > **Note**
   >
   > (For Oracle 12c pluggable database): Create and grant privileges for a local user while connected to the pluggable database to use as the subscription database. Create a common user in the Oracle 12c root container `CDB$ROOT`. Grant privileges to the common user while connected to the pluggable database to use as the subscription database.

   > **Note**
   >
   > (For Oracle 12c non-container database): Create and grant privileges to the subscription database user in the same manner as for Oracle versions prior to 12c.

   When creating the subscription database definition, enter the subscription database user name in the Subscription Service – Add Database dialog box (see Adding a subscription database).

```
CREATE USER subuser IDENTIFIED BY password;
GRANT CONNECT TO
subuser;
```

3. Grant the privileges needed to create the replicated database objects.

Create the replicated database objects in the schema owned by and with the same name as the subscription database user.

```
GRANT RESOURCE TO
subuser;
```

4. (For Oracle 12c only) Grant the privileges required to access tablespaces. Grant the `GRANT UNLIMITED TABLESPACE` privilege to the subscription database user. This requirement applies to both a pluggable database and a non-container database.

```
GRANT UNLIMITED TABLESPACE TO
subuser;
```

SQL Server subscription database

1. Create or choose the subscription database.

   The names of the schemas containing the publication tables and views become the names of the SQL Server schemas for the subscription tables. The subscription server creates these schemas in the subscription database when you create the subscription. If schemas with these names already exist in the subscription database, the existing schemas store the subscription tables.

   > **Note**
   >
   > If the schema containing the publication tables and views is named `public`, then the subscription server creates a schema named `public_sql` in the SQL Server subscription database for the subscription tables.

   The existing schemas can't contain any tables or views with the same names as the publication tables and views. The subscription server returns an error if there are already identically named tables or views. You must delete or rename these tables and views before the subscription is created.

   You can create a new subscription database as follows:

```
USE primary;
GO
CREATE DATABASE
subdb;
GO
```

2. Create a passworded SQL Server login for the subscription database user.

   When creating the subscription database definition, enter the SQL Server login in the Subscription Service – Add Database dialog box (seeAdding a subscription database).

```
CREATE LOGIN subuser WITH PASSWORD = '<password>';
GO
```

3. The subscription database must have a database user who is the creator and owner of the subscription tables. This database user must be mapped to the SQL Server login created in Step 2.

   In this example, the database user is given the same name as the SQL Server login `subuser`.

```
USE
 subdb;
GO
CREATE USER subuser FOR LOGIN
subuser;
GO
```

4. Grant the database-level privileges needed by the subscription database user to create the schema and tables for the subscription.

```
GRANT CREATE SCHEMA TO
 subuser;
GRANT CREATE TABLE TO
subuser;
GO
```

8.1.6 Verifying host accessibility

If you use more than one computer to host the components of the replication system, each computer must be able to communicate with the others on a network.

Firewalls and access to ports

Verify that the firewalls on the hosts allow access from the other hosts running replication system components. Refer to the instructions for your host's operating system to enable accessibility.

In addition, you might run the Replication Server console or CLI on a host different from where the publication server or subscription server are running. In this case, be sure the firewall on the host running these servers allows access to the ports used by the publication server and subscription server.

The Replication Server console and CLI access the publication server and the subscription server using Java remote method invocation (RMI) through the designated ports.

The publication server uses the port number you specified during installation as well as the port offset by a value of 2 greater than this specified port number. So, for a default publication server installation, access is required for port numbers `9051` and `9053`.

The subscription server uses the port number you specified during installation as well as the port offset by a value of 2 greater than this specified port number. So, for a default subscription server installation, access is required for port numbers `9052` and `9054`.

When you install Replication Server, the port numbers you specify for the publication server and the subscription server are stored in the Replication Server startup configuration file as shown in the following example. See Replication Server startup configuration file for more information.

```
#!/bin/sh

JAVA_EXECUTABLE_PATH="/usr/bin/java"
JAVA_MINIMUM_VERSION=1.9
JAVA_BITNESS_REQUIRED=64
JAVA_HEAP_SIZE="-Xms2048m -Xmx4096m"
PUBPORT=9051
SUBPORT=9052
```

If you want to use different port numbers, modify the `PUBPORT` and `SUBPORT` entries in the Replication Server startup configuration file and restart the publication server and subscription server.

> **Note**
>
> If you change the port numbers for the publication server or subscription server for which there are existing replication systems, you must perform additional updates on these existing replication systems.See Subscription server network location for changes to make for the publication server metadata in the control schema if the port number used by the subscription server changes. See Updating a subscription for changes to make for the subscription metadata in the control schema if the port number used by the publication server changed.

Network IP addresses

When configuring a replication system, you must supply the network location of various components such as the publication server, subscription server, publication database server, and subscription database server. This information, consisting of the component's IP address and port number, is stored in the control schema.

When one component needs to access another, it refers to the network location stored in the control schema.

During replication system configuration, we recommend that you supply the actual network IP address of each component and avoid using the `loopback` address, `localhost` or `127.x.x.x`, even if all components are running on the same host.

You can obtain the network IP address using the following command:

For Linux only: Use the `/sbin/ifconfig` command.

For Windows only: Use the `ipconfig` command.

The loopback address works as long as the communicating components are on the same host. But if, in the future, you decide to move a component to a different host on the network, the loopback address stored as the component's network address in the control schema will no longer work for the component trying to make the connection.

For Linux only: You might need to modify the `/etc/hosts` file so that a host's network IP address is associated with the host's name.

(For an alternative to modifying the `/etc/hosts` file, see Assigning an IP Address for remote method invocation.)

The default configuration on Linux systems associates the host name with the loopback address in the `/etc/hosts` file as shown by the following example:

```
127.0.0.1               localhost.localdomain localhost
```

You can verify this configuration by using the `hostname -i` command, which returns the IP address associated with the host name:

```
$ hostname -i
127.0.0.1
```

In these circumstances, certain Replication Server components can have trouble locating the other components on the network as in the following cases:

- When the user interface attempts to connect to the publication server or subscription server
- When the subscription server attempts to connect to the publication server

If the loopback address `127.x.x.x` is returned such as in the preceding example, edit the `/etc/hosts` file so that the network IP address is associated with the host name instead.

The following example shows the modified `/etc/hosts` file so that the host name `localhost` is now associated with the network IP address `192.168.2.22` instead of the loopback address `127.0.0.1`:

```
#127.0.0.1              localhost.localdomain localhost
192.168.2.22            localhost.localdomain localhost
::1                     localhost6.localdomain6 localhost6
```

On some Linux systems, you might need to restart the network service after you modify the `/etc/hosts` file. You can do this in a number of different ways, depending on the Linux system you are using, as shown by the following variations:

```
service network restart
/etc/init.d/networking restart
sudo /etc/init.d/networking restart
```

The following example shows the service network command:

```
$ su root
Password:
$ service network restart
Shutting down loopback interface:                          [  OK  ]
Bringing up loopback interface:                            [  OK  ]
```

Use the following command for CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8:

```
systemctl restart network
```

The `hostname -i` command now returns the network IP address of the host:

```
$ hostname -i
192.168.2.22
```

Postgres server authentication

#postgres-server-authentication

A Postgres database server uses the host-based authentication file `pg_hba.conf` to control access to the databases in the database server. You need to modify the `pg_hba.conf` file in the following locations:

- On each Postgres database server that contains a Postgres publication database
- On each Postgres database server that contains a Postgres subscription database

In a default Postgres installation, this file is located in the directory `POSTGRES_INSTALL_HOME/data`.

The modifications needed to the `pg_hba.conf` file for each of these cases follow.

Postgres publication database

For a Postgres publication database, you need the following to allow access to the publication database:

```
host <pub_dbname>    <pub_dbuser>    <pub_ipaddr>/32   md5
host <pub_dbname>    <pub_dbuser>    <sub_ipaddr>/32   md5
```

The value you substitute for `pub_dbname` is the name of the Postgres publication database you intend to use. The value you substitute for `pub_dbuser` is the publication database user name you created in Step 1 of Postgres Publication Database.

For a Postgres publication database named edb, the resulting `pg_hba.conf` file appears as follows:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                                md5
# IPv4 local connections:
host    edb         pubuser        192.168.2.22/32     md5
host    all         all            127.0.0.1/32        md5
# IPv6 local connections:
```

```
host    all         all            ::1/128             md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                      md5
#host    replication  enterprisedb  127.0.0.1/32        md5
#host    replication  enterprisedb  ::1/128             md5
```

> **Note**
>
> The preceding example assumes the publication server and the subscription server are running on the same host, hence the single entry for database edb. If the publication server and subscription server are running on separate hosts, then the `pg_hba.conf` file on the publication database server appears as follows:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                              md5
# IPv4 local connections:
host    edb         pubuser        192.168.2.22/32     md5
host    edb         pubuser        192.168.2.24/32     md5
host    all         all            127.0.0.1/32        md5
# IPv6 local connections:
host    all         all            ::1/128             md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                      md5
#host    replication  enterprisedb  127.0.0.1/32        md5
#host    replication  enterprisedb  ::1/128             md5
```

In addition, the preceding examples assume publication database edb is using the trigger-based method of synchronization replication. If you are using the log-based method, the `pg_hba.conf` file must contain an additional entry with the DATABASE field set to replication for `<pub_dbname>`, `<pub_dbuser>`, and `<pub_ipaddr>` to allow replication connections from the publication server on the host on which it's running.

The following shows a modification of the preceding example with this additional entry as the last line in the file:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                              md5
# IPv4 local connections:
host    edb         pubuser        192.168.2.22/32     md5
host    edb         pubuser        192.168.2.24/32     md5
host    all         all            127.0.0.1/32        md5
# IPv6 local connections:
host    all         all            ::1/128             md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                      md5
#host    replication  enterprisedb  127.0.0.1/32        md5
#host    replication  enterprisedb  ::1/128             md5
host    replication  pubuser       192.168.2.22/32     md5
```

See Synchronization replication with the log-based method and Enabling synchronization replication with the log-based method for additional information.

Reload the configuration file after making the modifications. Select **Reload Configuration** (**Expert Configuration > Reload Configuration** on EDB Postgres Advanced Server) from the Postgres application menu. Reloading puts the modified `pg_hba.conf` file into effect.

Postgres subscription database

For a Postgres subscription database, you need the following entries to allow access to the subscription database:

```
host <sub_dbname>    <sub_dbuser>    <pub_ipaddr>/32   md5
host <sub_dbname>    <sub_dbuser>    <sub_ipaddr>/32   md5
```

The values you substitute for `<sub_dbuser>` and `<sub_dbname>` are the subscription database user name and the subscription database name you created in steps 1 and 2 of Postgres subscription database.

For a Postgres subscription database named subdb, the resulting `pg_hba.conf` file appears as follows:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                               md5
# IPv4 local connections:
host    subdb       subuser        192.168.2.22/32    md5
host    all         all            127.0.0.1/32       md5
# IPv6 local connections:
host    all         all            ::1/128            md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                      md5
#host    replication  enterprisedb  127.0.0.1/32       md5
#host    replication  enterprisedb  ::1/128            md5
```

> **Note**
>
> The preceding example assumes that the publication server and the subscription server are running on the same host. Hence, only one entry is needed for database subdb. If the publication server and subscription server are running on separate hosts, then the `pg_hba.conf` file on the subscription database server looks like the following:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                               md5
# IPv4 local connections:
host    subdb       subuser        192.168.2.22/32    md5
host    subdb       subuser        192.168.2.24/32    md5
host    all         all            127.0.0.1/32       md5
# IPv6 local connections:
host    all         all             ::1/128           md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                      md5
#host    replication  enterprisedb  127.0.0.1/32       md5
#host    replication  enterprisedb  ::1/128            md5
```

Reload the configuration file after making the modifications.

Select **Reload Configuration** (**Expert Configuration > Reload Configuration** on EDB Postgres Advanced Server) from the Postgres application menu. Reloading puts the modified `pg_hba.conf` file into effect.

8.2 Creating a publication

Creating your first publication requires the following steps:

- Registering the publication server
- Adding the publication database
- Creating a publication by choosing the tables and views for the publication along with creating any optional filter clauses

After you add the publication database, you can create as many publications as there are available tables and views. These tables and views must be readable by the publication database user and meet the criteria outlined in Design considerations and Restrictions on replicated database objects.

8.2.1 Registering a publication server

When you register a publication server, you're identifying the network location, admin user name, and password of a specific, running, publication server instance. You want to use this instance to manage all aspects of the publications you create subordinate to it.

It's important that you record the login information for the publication server. You must always use this same publication server instance to manage all publications created subordinate to it as represented in the Replication Server console replication tree.

1. Start the publication server if it's not already running.

   > **Note**
   >
   > If you're using Oracle publication or subscription databases, and the publication server wasn't restarted since copying the Oracle JDBC driver to the `lib/jdbc` subdirectory of your Replication Server installation, restart the publication server.

   For Linux only: You can verify the publication server is running by using the `systemctl` command for CentOS 7 or RHEL 7 and Rocky Linux, AlmaLinux, or RHEL 8, and the service command for previous Linux versions.

   Use the following command for CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8:

   ```
   systemctl status edb-xdbpubserver
   ```

   For previous Linux versions:

   ```
   service edb-xdbpubserver status
   ```

   If the publication server is running and you want to restart it, use the `restart` option.

   For CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8:

   ```
   systemctl restart edb-xdbpubserver
   ```

   For previous Linux versions:

   ```
   service edb-xdbpubserver restart
   ```

   If the publication server isn't running, use the `start` option.

   For CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8:

   ```
   systemctl start edb-xdbpubserver
   ```

For previous Linux versions:

```
service edb-xdbpubserver start
```

Similarly, use the `stop` option to stop the publication server.

For Windows only: Open **Control Panel > System and Security > Administrative Tools > Services**. The publication server runs as a service named Publication Service for Replication Server. Use the **Start** or **Restart** link for the service.

If the publication server doesn't start, see Publication and subscription server startup failures.

2. Register the publication server. Open the Replication Server console from the system's application menu. For Replication Server installed from an Replication Server RPM package, invoke the script `XDB_HOME/bin/runRepConsole.sh` to start the Replication Server console.

3. Select the top-level Replication Servers node. Select **File > Publication Server > Register Server**.

   In the Register Publication Server dialog box, enter the values you supplied while installing Replication Server unless otherwise specified.

   - **Host**. Network IP address of the host running the publication server. This is the network IP address used for `pub_ipaddr` in the `pg_hba.conf` file in Postgres server authentication. (Don't use `localhost` for this field.)
   - **Port**. Port number the publication server is using. You can find the port number in the `xdbReplicationServer-xx.config` file. See Installation details for the location of the file.
   - **User Name**. Admin user name used to authenticate your use of this publication server. You can find the user name in the `edb-repl.conf` file. See Installation details for the location of the file.
   - **Password**. Password of the admin user given in the **User Name** field.
   - **Save login information**. Select this box if you don't want to register the publication server each time you open the Replication Server console. See Saving server login information for additional information on the advantages and disadvantages of saving server login information.

   > **Note**
   >
   > The user name and password combination you enter is authenticated against the admin user name and password in the Replication Server configuration file residing on the host with the IP address you enter in the **Host** field.

   After you fill in the fields, select **Register**. A Publication Server node appears in the replication tree of the Replication Server console. Expand the Publication Server node to expose the SMR and MMR type nodes.

   Continue to build the single-master replication system under the SMR type node.

8.2.2 Adding a publication database

You must identify to Replication Server the database for publications by creating a publication database definition.

After you create the publication database definition, a Publication Database node representing that publication database definition appears in the replication tree of the Replication Server console. You can then create publications for containing tables and views residing in this database under the Publication Database node.

You must enter database connection information such as the database server network address, database identifier, and database login user name and password when you create the publication database definition. The publication server uses the connection information to access the publication tables and views when it performs replication.

1. Make sure the database server in which the publication database resides is running and accepting client connections.

2. Select the SMR type node under the Publication Server node. Select **Publication > Publication Database > Add Database**.

3. In the Publication Service – Add Database dialog box, fill in the following fields:

   - **Database Type**. Select **Oracle**, **SQL Server**, **PostgreSQL**, or **Postgres Plus Advanced Server** for the type of publication database. For an EDB Postgres Advanced Server Oracle-compatible installation, select the **Postgres Plus Advanced Server** option. For PostgreSQL or an Advanced Server PostgreSQL-compatible installation, select the **PostgreSQL** option.

   - **Host**. IP address of the host where the publication database server is running.

   - **Port**. Port on which the publication database server is listening for connections.

   - **User**. The publication database user name created in Step 1 of Preparing the publication database.

   - **Password**. Password of the database user.

   - **Service ID** (For Oracle). Enter the Oracle system identifier (SID) of the Oracle instance running the publication database if **SID** is selected. Enter the net service name of a connect descriptor as defined in the `TNSNAMES.ORA` file if **Service Name** is selected. For Oracle 12c pluggable database), use the service name.

   - **Database (For Postgres or SQL Server)**. Enter the Postgres or SQL Server database name.

   - **URL Options (For SSL connectivity).** Enter the URL options to establish SSL connectivity to the publication database. See Using secure sockets layer (SSL) connections for more information.

   - **Changeset Logging (For Postgres)**. Select **Table Triggers** to use the trigger-based method of synchronization replication. Select **WAL Stream** to use the log-based method of synchronization replication. For more information, see Synchronization replication with the trigger-based method and Synchronization replication with the log-based method.

     > **Note**
     >
     > If the controller database is an Oracle or SQL Server publication database, then you can't add a second Oracle or SQL Server publication database to create a second single-master replication system. For Replication Server to run more than one single-master replication systems consisting of Oracle or SQL Server publication databases, designate a Postgres publication database as the controller database. See Controller database for more information.

4. Select **Test**. When Test Result: Success appears, select **OK** and then select **Save**.

   When the publication database definition is successfully saved, a Publication Database node is added to the replication tree under the Publication Server node.

   For Oracle only: You can add multiple Oracle databases as publication databases by completing the Add Database dialog box for each database. You can also add the same Oracle database as two or more distinct publication database definitions if you use different publication database user names for each publication database definition.

   For Postgres or SQL Server: You can add multiple Postgres or SQL Server databases as publication databases by completing the Add Database dialog box for each database. However, unlike Oracle, you can add a given Postgres or SQL Server database as a publication database definition only once.

8.2.3 Adding a publication

Subordinate to a publication database definition, you create publications that contain tables and views of the database identified in the publication database definition.

1. Select the Publication Database node. Then select **Publication > Create Publication**.

2. In the Create Publication dialog box, on the **Create Publication** tab, fill in the following fields:

- **Publication Name**. Enter a name that is unique among all publications.
- **Snapshot-Only Replication**. Select this box to perform replication by snapshot only. Tables included in a snapshot-only publication don't require identity columns (primary key or unique columns). See Design considerations for details. Tables included in publications for synchronization replication must have identity columns.
- **Publish**. Select the boxes next to the tables to include in the publication. If you select**Snapshot-Only Replication**, then views appear in the Publish list as well.
- **Select All**. Select this option if you want to include all tables and views in the Available Tables list in the publication.
- **Use Wildcard Selection.** Select this option to use the wildcard selector to choose tables for the publication. See Selecting tables with the wildcard selector for more information.

3. (Optional) Table filters consist of a set of filter rules that control the selection criteria for rows replicated to the subscription tables during a snapshot or a synchronization replication. See Table settings and restrictions for table filters for table setup requirements for a log-based replication system as well as general restrictions on the use of table filters.

   A filter rule consists of a filter name and a `SQL WHERE` clause (omitting the WHERE keyword) called the filter clause, which you specify for a table or view that defines the selection criteria for rows include during a replication.

   You can define multiple filter rules for each table or view in the publication. If you don't define a filter rule for a given table or view, then you can't later enable filtering on a corresponding subscription table associated with that publication table.

   After filter rules are defined for a publication table or view, you can later choose whether to enable those filter rules on any subscription that you associate with that publication in accordance with the following rules.

   - You can enable at most one filter rule on a given table in a given subscription.
   - You can enable the same filter rule on the same given table in several, different subscriptions.
   - You can enable different filter rules n the same given table but in different subscriptions.

   If you want to define table filters on the publication tables or views, select the**Table Filters** tab. From the **Table/View** list, select the table or view for which you want to add a filter rule. Select **Add Filter**.

   In the Filter dialog box, enter a descriptive filter name and the filter clause to select the rows you want to replicate. The filter name and filter clause must meet the following conditions:

   - For any given table or view, each filter rule must be assigned a unique filter name.
   - For any given table or view, the filter clauses must have different syntaxes (that is, the filtering criteria must be different).

   Repeating this process, you can add filter rules for the EMP table.

   To remove a filter rule, select the filter rule you want to remove and then select **Remove Filter**.

   You can also modify the filter name or filter clause of a filter rule listed in the **Table Filters** tab. Double-click the cell of the filter name or filter clause you want to change. Enter the text for the desired change.

   When creating a subscription, you can selectively enable these table filters on the corresponding subscription tables. See Adding a subscription for information on creating a subscription.

4. Select **Create**. When Publication Created Successfully appears, select**OK**.

   A Publication node is added to the replication tree.

8.2.4 Control schema objects created for a publication

After you add a publication database definition and publications, the following control schema objects are created in addition to your original publication tables and views:

- In the publication database, control schema objects are created to control and manage the Replication Server replication systems. How the control

schema objects are organized under the actual, physical database schemas depends on the publication database server type, that is, whether it is Oracle, SQL Server, or Postgres.

- If the publication isn't a snapshot-only publication, that is, synchronization replication is permitted and synchronization replication is implemented using the trigger-based method, then three triggers and one shadow table are created for each publication table as part of the control schema.
- If the publication is using synchronization replication with the log-based method, then a single trigger is created for each publication table as part of the control schema.

Don't manually delete any of these database objects as the replication system control schema will become corrupted.

Removing publications and publication database definitions using the Replication Server console or CLI deletes the control schema objects.

Oracle control schema objects

The control schema objects created in the publication database user's schema (that is, the control schema) are shown in the following output:

```
 SQL> CONNECT pubuser/password
Connected.
SQL> SET PAGESIZE 9999
SQL> SELECT table_name FROM user_tables ORDER BY table_name;

TABLE_NAME
------------------------------
RREP_LOCK
RREP_MMR_PUB_GROUP
RREP_MMR_TXSET
RREP_PROPERTIES
RREP_PUBLICATION_SUBSCRIPTIONS
RREP_PUBLICATION_TABLES
RREP_TABLES
RREP_TXSET
RREP_TXSET_HEALTH
RREP_TXSET_LOG
RREP_TX_MONITOR
RREP_TX_MONITOR_TEMP
RRST_EDB_DEPT
RRST_EDB_EMP
SCH_PUB_BLOB_TRIGGERS
SCH_PUB_CALENDARS
SCH_PUB_CRON_TRIGGERS
SCH_PUB_FIRED_TRIGGERS
SCH_PUB_JOB_DETAILS
SCH_PUB_JOB_LISTENERS
SCH_PUB_LOCKS
SCH_PUB_PAUSED_TRIGGER_GRPS
SCH_PUB_SCHEDULER_STATE
SCH_PUB_SIMPLE_TRIGGERS
SCH_PUB_TRIGGERS
SCH_PUB_TRIGGER_LISTENERS
SCH_SUB_BLOB_TRIGGERS
SCH_SUB_CALENDARS
SCH_SUB_CRON_TRIGGERS
SCH_SUB_FIRED_TRIGGERS
SCH_SUB_JOB_DETAILS
SCH_SUB_JOB_LISTENERS
SCH_SUB_LOCKS
SCH_SUB_PAUSED_TRIGGER_GRPS
SCH_SUB_SCHEDULER_STATE
SCH_SUB_SIMPLE_TRIGGERS
```

```
SCH_SUB_TRIGGERS
SCH_SUB_TRIGGER_LISTENERS
XDB_CLEANUP_CONF
XDB_CONFLICTS
XDB_CONFLICTS_OPTIONS
XDB_EVENTS
XDB_EVENTS_STATUS
XDB_MMR_PUB_GROUP
XDB_PUBLICATIONS
XDB_PUBLICATION_FILTER
XDB_PUBLICATION_FILTER_RULE
XDB_PUBLICATION_SUBSCRIPTIONS
XDB_PUBTABLES_IGNOREDCOLS
XDB_PUB_DATABASE
XDB_PUB_REPLOG
XDB_PUB_TABLE_REPLOG
XDB_SUBSCRIPTIONS
XDB_SUBSCRIPTION_TABLES
XDB_SUB_DATABASE
XDB_SUB_SERVERS
XDB_TABLES

57 rows selected.

SQL> SELECT sequence_name FROM user_sequences ORDER BY sequence_name;

SEQUENCE_NAME
------------------------------
RREP_COMMON_SEQ
RREP_TXSET_SEQ
RREP_TX_SEQ

SQL> SELECT DISTINCT name FROM user_source WHERE type = 'PACKAGE';

NAME
------------------------------
RREP_PKG

SQL> SELECT trigger_name FROM user_triggers ORDER BY trigger_name;

TRIGGER_NAME
------------------------------
RRPD_EDB_DEPT
RRPD_EDB_EMP
RRPI_EDB_DEPT
RRPI_EDB_EMP
RRPU_EDB_DEPT
RRPU_EDB_EMP
SCH_PUB_BLOB_TRIGGERS_TRIGGER
SCH_PUB_CALENDARS_TRIGGER
SCH_PUB_CRON_TRIGGERS_TRIGGER
SCH_PUB_JOB_DETAILS_TRIGGER
SCH_PUB_JOB_LISTENERS_TRIGGER
SCH_PUB_SIMPLE_TRIGGERS_TRIG
SCH_PUB_TRIGGERS_TRIG
SCH_PUB_TRIGGER_LISTENERS_TRIG
SCH_SUB_BLOB_TRIGGERS_TRIGGER
SCH_SUB_CALENDARS_TRIGGER
SCH_SUB_CRON_TRIGGERS_TRIGGER
SCH_SUB_JOB_DETAILS_TRIGGER
SCH_SUB_JOB_LISTENERS_TRIGGER
```

```
SCH_SUB_SIMPLE_TRIGGERS_TRIG
SCH_SUB_TRIGGERS_TRIG
SCH_SUB_TRIGGER_LISTENERS_TRIG
XDB_CLEANUP_CONF_TRIGGER
XDB_CONFLICTS_OPTIONS_TRIGGER
XDB_CONFLICTS_TRIGGER
XDB_MMR_PUB_GROUP_TRIGGER
XDB_PUBLICATIONS_TRIGGER
XDB_PUBLICATION_FILTER_TRIGGER
XDB_PUBLICATION_SUBSCRIPT_TRIG
XDB_PUBLIC_FILTER_RULE_TRIGGER
XDB_PUBTABLES_IGNOREDCOLS_TRIG
XDB_PUB_DATABASE_TRIGGER
XDB_PUB_REPLOG_TRIGGER
XDB_PUB_TABLE_REPLOG_TRIGGER
XDB_SUBSCRIPTIONS_TRIGGER
XDB_SUBSCRIPTION_TABLES_TRIG
XDB_SUB_DATABASE_TRIGGER
XDB_SUB_SERVERS_TRIGGER
XDB_TABLES_TRIGGER

39 rows selected.

SQL> SELECT type_name, typecode FROM user_types;

TYPE_NAME                      TYPECODE
------------------------------ ------------------------------
RREP_SYNCID_ARRAY              COLLECTION
```

Note the following:

- The tables named according to the convention `RRST_schema_table` from the SELECT statement on user_tables are found only for synchronization publications. In this example, these tables are `RRST_EDB_DEPT` and `RRST_EDB_EMP`.
- The triggers named according to the convention RRPD_schema_table— `RRPI_schema_table`, and `RRPU_schema_table` from the `SELECT statement` on `user_triggers` —are found only for synchronization publications. In this example, these triggers are `RRPU_EDB_DEPT`, `RRPI_EDB_DEPT`, `RRPD_EDB_DEPT`, `RRPI_EDB_EMP`, `RRPU_EDB_EMP`, and `RRPD_EDB_EMP`.

The following example shows what the same set of queries look like for a snapshot-only publication:

```
 SQL> CONNECT pubuser/password
Connected.
SQL> SET PAGESIZE 9999
SQL> SELECT table_name FROM user_tables ORDER BY table_name;

TABLE_NAME
------------------------------
RREP_LOCK
RREP_MMR_PUB_GROUP
RREP_MMR_TXSET
RREP_PROPERTIES
RREP_PUBLICATION_SUBSCRIPTIONS
RREP_PUBLICATION_TABLES
RREP_TABLES
RREP_TXSET
RREP_TXSET_HEALTH
RREP_TXSET_LOG
RREP_TX_MONITOR
RREP_TX_MONITOR_TEMP
SCH_PUB_BLOB_TRIGGERS
SCH_PUB_CALENDARS
```

```
SCH_PUB_CRON_TRIGGERS
SCH_PUB_FIRED_TRIGGERS
SCH_PUB_JOB_DETAILS
SCH_PUB_JOB_LISTENERS
SCH_PUB_LOCKS
SCH_PUB_PAUSED_TRIGGER_GRPS
SCH_PUB_SCHEDULER_STATE
SCH_PUB_SIMPLE_TRIGGERS
SCH_PUB_TRIGGERS
SCH_PUB_TRIGGER_LISTENERS
SCH_SUB_BLOB_TRIGGERS
SCH_SUB_CALENDARS
SCH_SUB_CRON_TRIGGERS
SCH_SUB_FIRED_TRIGGERS
SCH_SUB_JOB_DETAILS
SCH_SUB_JOB_LISTENERS
SCH_SUB_LOCKS
SCH_SUB_PAUSED_TRIGGER_GRPS
SCH_SUB_SCHEDULER_STATE
SCH_SUB_SIMPLE_TRIGGERS
SCH_SUB_TRIGGERS
SCH_SUB_TRIGGER_LISTENERS
XDB_CLEANUP_CONF
XDB_CONFLICTS
XDB_CONFLICTS_OPTIONS
XDB_EVENTS
XDB_EVENTS_STATUS
XDB_MMR_PUB_GROUP
XDB_PUBLICATIONS
XDB_PUBLICATION_FILTER
XDB_PUBLICATION_FILTER_RULE
XDB_PUBLICATION_SUBSCRIPTIONS
XDB_PUBTABLES_IGNOREDCOLS
XDB_PUB_DATABASE
XDB_PUB_REPLOG
XDB_PUB_TABLE_REPLOG
XDB_SUBSCRIPTIONS
XDB_SUBSCRIPTION_TABLES
XDB_SUB_DATABASE
XDB_SUB_SERVERS
XDB_TABLES

55 rows selected.

SQL> SELECT sequence_name FROM user_sequences ORDER BY sequence_name;

SEQUENCE_NAME
------------------------------
RREP_COMMON_SEQ
RREP_TXSET_SEQ
RREP_TX_SEQ

SQL> SELECT DISTINCT name FROM user_source WHERE type = 'PACKAGE';

NAME
------------------------------
RREP_PKG

SQL> SELECT trigger_name FROM user_triggers ORDER BY trigger_name;

TRIGGER_NAME
```

```
------------------------------
SCH_PUB_BLOB_TRIGGERS_TRIGGER
SCH_PUB_CALENDARS_TRIGGER
SCH_PUB_CRON_TRIGGERS_TRIGGER
SCH_PUB_JOB_DETAILS_TRIGGER
SCH_PUB_JOB_LISTENERS_TRIGGER
SCH_PUB_SIMPLE_TRIGGERS_TRIG
SCH_PUB_TRIGGERS_TRIG
SCH_PUB_TRIGGER_LISTENERS_TRIG
SCH_SUB_BLOB_TRIGGERS_TRIGGER
SCH_SUB_CALENDARS_TRIGGER
SCH_SUB_CRON_TRIGGERS_TRIGGER
SCH_SUB_JOB_DETAILS_TRIGGER
SCH_SUB_JOB_LISTENERS_TRIGGER
SCH_SUB_SIMPLE_TRIGGERS_TRIG
SCH_SUB_TRIGGERS_TRIG
SCH_SUB_TRIGGER_LISTENERS_TRIG
XDB_CLEANUP_CONF_TRIGGER
XDB_CONFLICTS_OPTIONS_TRIGGER
XDB_CONFLICTS_TRIGGER
XDB_MMR_PUB_GROUP_TRIGGER
XDB_PUBLICATIONS_TRIGGER
XDB_PUBLICATION_FILTER_TRIGGER
XDB_PUBLICATION_SUBSCRIPT_TRIG
XDB_PUBLIC_FILTER_RULE_TRIGGER
XDB_PUBTABLES_IGNOREDCOLS_TRIG
XDB_PUB_DATABASE_TRIGGER
XDB_PUB_REPLOG_TRIGGER
XDB_PUB_TABLE_REPLOG_TRIGGER
XDB_SUBSCRIPTIONS_TRIGGER
XDB_SUBSCRIPTION_TABLES_TRIG
XDB_SUB_DATABASE_TRIGGER
XDB_SUB_SERVERS_TRIGGER
XDB_TABLES_TRIGGER

33 rows selected.

SQL> SELECT type_name, typecode FROM user_types;

TYPE_NAME                      TYPECODE
------------------------------ ------------------------------
RREP_SYNCID_ARRAY              COLLECTION
```

> **Note**
>
> The `RREP_SYNCID_ARRAY` collection type is found only in an Oracle publication database.

SQL Server control schema objects

Most of the control schema objects are created in `schemas _edb_replicator_pub` , `_edb_replicator_sub` , and `_edb_scheduler` . Additional control schema objects are created in the schema you chose in Step 5 of SQL Server publication database. The following examples assume the schema you chose is `pubuser` . The publication tables are `dept` and `emp` located in the `edb` schema.

The following query lists the control schema objects located in these schemas:

```
1> USE edb;
```

```
2> GO
Changed database context to 'edb'.
1> SELECT s.name + '.' + o.name "Object Name", o.type_desc "Object Type"
2>   FROM sys.objects o,
3>        sys.schemas s
4>   WHERE s.name IN ('_edb_replicator_pub', '_edb_replicator_sub',
5>                    '_edb_scheduler', 'pubuser')
6>     AND o.type IN ('U','P','FN')
7>     AND o.schema_id = s.schema_id
8>   ORDER BY 1, 2;
9> GO
Object Name                                         Object Type
--------------------------------------------------- ---------------------
_edb_replicator_pub.nextval                         SQL_STORED_PROCEDURE
_edb_replicator_pub.rrep_common_seq                 USER_TABLE
_edb_replicator_pub.rrep_lock                       USER_TABLE
_edb_replicator_pub.rrep_MMR_pub_group              USER_TABLE
_edb_replicator_pub.rrep_MMR_txset                  USER_TABLE
_edb_replicator_pub.rrep_properties                 USER_TABLE
_edb_replicator_pub.rrep_publication_subscriptions USER_TABLE
_edb_replicator_pub.rrep_publication_tables         USER_TABLE
_edb_replicator_pub.rrep_tables                     USER_TABLE
_edb_replicator_pub.rrep_tx_monitor                 USER_TABLE
_edb_replicator_pub.rrep_tx_seq                     USER_TABLE
_edb_replicator_pub.rrep_txset                      USER_TABLE
_edb_replicator_pub.rrep_txset_health               USER_TABLE
_edb_replicator_pub.rrep_txset_log                  USER_TABLE
_edb_replicator_pub.rrep_txset_seq                  USER_TABLE
_edb_replicator_pub.sp_createsequence               SQL_STORED_PROCEDURE
_edb_replicator_pub.sp_dropsequence                 SQL_STORED_PROCEDURE
_edb_replicator_pub.xdb_cleanup_conf                USER_TABLE
_edb_replicator_pub.xdb_conflicts                   USER_TABLE
_edb_replicator_pub.xdb_conflicts_options           USER_TABLE
_edb_replicator_pub.xdb_events                      USER_TABLE
_edb_replicator_pub.xdb_events_status               USER_TABLE
_edb_replicator_pub.xdb_MMR_pub_group               USER_TABLE
_edb_replicator_pub.xdb_pub_database                USER_TABLE
_edb_replicator_pub.xdb_pub_replog                  USER_TABLE
_edb_replicator_pub.xdb_pub_table_replog            USER_TABLE
_edb_replicator_pub.xdb_publication_filter          USER_TABLE
_edb_replicator_pub.xdb_publication_filter_rule     USER_TABLE
_edb_replicator_pub.xdb_publication_subscriptions   USER_TABLE
_edb_replicator_pub.xdb_publications                USER_TABLE
_edb_replicator_pub.xdb_pubtables_ignoredcols       USER_TABLE
_edb_replicator_pub.xdb_sub_servers                 USER_TABLE
_edb_replicator_sub.rrep_common_seq                 USER_TABLE
_edb_replicator_sub.xdb_sub_database                USER_TABLE
_edb_replicator_sub.xdb_subscription_tables         USER_TABLE
_edb_replicator_sub.xdb_subscriptions               USER_TABLE
_edb_replicator_sub.xdb_tables                      USER_TABLE
_edb_scheduler.sch_pub_BLOB_TRIGGERS                USER_TABLE
_edb_scheduler.sch_pub_CALENDARS                    USER_TABLE
_edb_scheduler.sch_pub_CRON_TRIGGERS                USER_TABLE
_edb_scheduler.sch_pub_FIRED_TRIGGERS               USER_TABLE
_edb_scheduler.sch_pub_JOB_DETAILS                  USER_TABLE
_edb_scheduler.sch_pub_JOB_LISTENERS                USER_TABLE
_edb_scheduler.sch_pub_LOCKS                        USER_TABLE
_edb_scheduler.sch_pub_PAUSED_TRIGGER_GRPS          USER_TABLE
_edb_scheduler.sch_pub_SCHEDULER_STATE              USER_TABLE
_edb_scheduler.sch_pub_SIMPLE_TRIGGERS              USER_TABLE
_edb_scheduler.sch_pub_TRIGGER_LISTENERS            USER_TABLE
```

```
_edb_scheduler.sch_pub_TRIGGERS                    USER_TABLE
_edb_scheduler.sch_sub_BLOB_TRIGGERS               USER_TABLE
_edb_scheduler.sch_sub_CALENDARS                   USER_TABLE
_edb_scheduler.sch_sub_CRON_TRIGGERS               USER_TABLE
_edb_scheduler.sch_sub_FIRED_TRIGGERS              USER_TABLE
_edb_scheduler.sch_sub_JOB_DETAILS                 USER_TABLE
_edb_scheduler.sch_sub_JOB_LISTENERS               USER_TABLE
_edb_scheduler.sch_sub_LOCKS                       USER_TABLE
_edb_scheduler.sch_sub_PAUSED_TRIGGER_GRPS         USER_TABLE
_edb_scheduler.sch_sub_SCHEDULER_STATE             USER_TABLE
_edb_scheduler.sch_sub_SIMPLE_TRIGGERS             USER_TABLE
_edb_scheduler.sch_sub_TRIGGER_LISTENERS           USER_TABLE
_edb_scheduler.sch_sub_TRIGGERS                    USER_TABLE
pubuser.CleanupShadowTables                        SQL_STORED_PROCEDURE
pubuser.ConfigureCleanUpJob                        SQL_STORED_PROCEDURE
pubuser.ConfigureCreateTxSetJob                    SQL_STORED_PROCEDURE
pubuser.CreateMultiTxSet                           SQL_STORED_PROCEDURE
pubuser.CreateTableLogTrigger                      SQL_STORED_PROCEDURE
pubuser.CreateTxSet                                SQL_STORED_PROCEDURE
pubuser.CreateTxSet_old                            SQL_STORED_PROCEDURE
pubuser.CreateUniTxSet                             SQL_STORED_PROCEDURE
pubuser.GetNewTxsCount                             SQL_STORED_PROCEDURE
pubuser.getPackageVersionNumber                    SQL_SCALAR_FUNCTION
pubuser.JobCleanup                                 SQL_STORED_PROCEDURE
pubuser.JobCreateTxSet                             SQL_STORED_PROCEDURE
pubuser.LoadPubTableList                           SQL_STORED_PROCEDURE
pubuser.RemoveCleanupJob                           SQL_STORED_PROCEDURE
pubuser.RemoveCreateTxSetJob                       SQL_STORED_PROCEDURE
pubuser.rrst_edb_dept                              USER_TABLE
pubuser.rrst_edb_emp                               USER_TABLE

(78 rows affected)
```

Note (For SQL Server 2012, 2014): The following database objects from the preceding list are no longer created as part of the control schema when the publication database is SQL Server 2012 or 2014:

```
 Object Name                                         Object Type
---------------------------------------------------- --------------------
_edb_replicator_pub.nextval                         SQL_STORED_PROCEDURE
_edb_replicator_pub.rrep_common_seq                 USER_TABLE
_edb_replicator_pub.rrep_tx_seq                     USER_TABLE
_edb_replicator_pub.rrep_txset_seq                  USER_TABLE
_edb_replicator_pub.sp_createsequence               SQL_STORED_PROCEDURE
_edb_replicator_pub.sp_dropsequence                 SQL_STORED_PROCEDURE
_edb_replicator_sub.rrep_common_seq                 USER_TABLE
```

SQL Server versions 2014 and 2012 support creation of sequence objects that can now perform the functionality previously provided by the preceding list of objects. The following are the sequence objects that are now used when the publication database is SQL Server 2012 or 2014:

```
 1> USE edb;
 2> GO
Changed database context to 'edb'.
 1> SELECT s.name + '.' + o.name "Object Name", o.type_desc "Object Type"
 2>   FROM sys.objects o,
 3>        sys.schemas s
 4>   WHERE s.name IN ('_edb_replicator_pub', '_edb_replicator_sub',
 5>                    '_edb_scheduler', 'pubuser')
 6>     AND o.type IN ('SO')
 7>     AND o.schema_id = s.schema_id
 8>   ORDER BY 1, 2;
```

```
9> GO
Object Name                                        Object Type
-------------------------------------------------- --------------------
_edb_replicator_pub.rrep_common_seq                SEQUENCE_OBJECT
_edb_replicator_pub.rrep_tx_seq                    SEQUENCE_OBJECT
_edb_replicator_pub.rrep_txset_seq                 SEQUENCE_OBJECT

(3 rows affected)
```

The following is a continuation of the list of control schema objects for all SQL Server versions:

```
 1> USE edb;
2> GO
Changed database context to 'edb'.
1> SELECT s.name + '.' + o.name "Trigger Name", o.type_desc "Object Type"
2>   FROM sys.objects o,
3>        sys.schemas s
4>   WHERE s.name IN ('_edb_replicator_pub', '_edb_replicator_sub',
5>                    '_edb_scheduler', 'pubuser')
6>     AND o.type = 'TR'
7>     AND o.schema_id = s.schema_id
8>   ORDER BY 1;
9> GO
Trigger Name                                                Object Type
----------------------------------------------------------- --------------
_edb_replicator_pub.xdb_cleanup_conf_trigger                SQL_TRIGGER
_edb_replicator_pub.xdb_conflicts_options_trigger           SQL_TRIGGER
_edb_replicator_pub.xdb_conflicts_trigger                   SQL_TRIGGER
_edb_replicator_pub.xdb_MMR_pub_group_trigger               SQL_TRIGGER
_edb_replicator_pub.xdb_pub_database_trigger                SQL_TRIGGER
_edb_replicator_pub.xdb_pub_replog_trigger                  SQL_TRIGGER
_edb_replicator_pub.xdb_pub_table_replog_trigger            SQL_TRIGGER
_edb_replicator_pub.xdb_public_filter_rule_trigger          SQL_TRIGGER
_edb_replicator_pub.xdb_publication_filter_trigger          SQL_TRIGGER
_edb_replicator_pub.xdb_publication_subscription_triggers   SQL_TRIGGER
_edb_replicator_pub.xdb_publications_trigger                SQL_TRIGGER
_edb_replicator_pub.xdb_pubtables_ignoredcols_trig          SQL_TRIGGER
_edb_replicator_pub.xdb_sub_servers_trigger                 SQL_TRIGGER
_edb_replicator_sub.xdb_sub_database_trigger                SQL_TRIGGER
_edb_replicator_sub.xdb_subscription_tables_trig            SQL_TRIGGER
_edb_replicator_sub.xdb_subscriptions_trigger               SQL_TRIGGER
_edb_replicator_sub.xdb_tables_trigger                      SQL_TRIGGER
_edb_scheduler.sch_pub_blob_triggers_trigger                SQL_TRIGGER
_edb_scheduler.sch_pub_calendars_trigger                    SQL_TRIGGER
_edb_scheduler.sch_pub_cron_triggers_trigger                SQL_TRIGGER
_edb_scheduler.sch_pub_job_details_trigger                  SQL_TRIGGER
_edb_scheduler.sch_pub_job_listeners_trigger                SQL_TRIGGER
_edb_scheduler.sch_pub_simple_triggers_trigger              SQL_TRIGGER
_edb_scheduler.sch_pub_trigger_listeners_trigger            SQL_TRIGGER
_edb_scheduler.sch_pub_triggers_trigger                     SQL_TRIGGER
_edb_scheduler.sch_sub_blob_triggers_trigger                SQL_TRIGGER
_edb_scheduler.sch_sub_calendars_trigger                    SQL_TRIGGER
_edb_scheduler.sch_sub_cron_triggers_trigger                SQL_TRIGGER
_edb_scheduler.sch_sub_job_details_trigger                  SQL_TRIGGER
_edb_scheduler.sch_sub_job_listeners_trigger                SQL_TRIGGER
_edb_scheduler.sch_sub_simple_triggers_trigger              SQL_TRIGGER
_edb_scheduler.sch_sub_trigger_listeners_trigger            SQL_TRIGGER
_edb_scheduler.sch_sub_triggers_trigger                     SQL_TRIGGER

(33 rows affected)
```

For non-snapshot-only publication tables, triggers are created that reside in the schema containing the publication tables as shown by the following:

```
 1> USE edb;
2> GO
Changed database context to 'edb'.
1> SELECT s.name + '.' + o.name "Trigger Name"
2>   FROM sys.objects o,
3>        sys.schemas s
4>   WHERE s.name = 'edb'
5>     AND o.type = 'TR'
6>     AND o.name LIKE 'rr%'
7>     AND o.schema_id = s.schema_id
8>   ORDER BY 1;
9> GO
Trigger Name
------------------------------------
edb.rrpd_edb_dept
edb.rrpd_edb_emp
edb.rrpi_edb_dept
edb.rrpi_edb_emp
edb.rrpu_edb_dept
edb.rrpu_edb_emp

(6 rows affected)
```

Finally, some jobs are created in the msdb database after the subscription is created as shown by the following:

```
 1> USE msdb;
2> GO
Changed database context to 'msdb'.
1> SELECT j.name "Job Name"
2>   FROM msdb.dbo.sysjobs j,
3>        primary.dbo.syslogins l
4>   WHERE l.name = 'pubuser'
5>     AND j.name LIKE 'rrep%'
6>     AND j.owner_sid = l.sid
7>   ORDER BY 1;
8> GO
Job Name
------------------------------------
rrep_cleanup_job_edb
rrep_txset_job_edb

(2 rows affected)
```

Postgres control schema objects

The control schema objects are created in three schemas named `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler`.

The control schema objects contained in _edb_replicator_pub are shown by the following:

```
 edb=# SET search_path TO _edb_replicator_pub;
SET
edb=# \dt
                          List of relations
       Schema        |              Name              | Type  |  Owner
```

```
--------------------+--------------------------------+-------+---------
 _edb_replicator_pub | rrep_lock                      | table | pubuser
 _edb_replicator_pub | rrep_MMR_pub_group             | table | pubuser
 _edb_replicator_pub | rrep_MMR_txset                 | table | pubuser
 _edb_replicator_pub | rrep_properties                | table | pubuser
 _edb_replicator_pub | rrep_publication_subscriptions | table | pubuser
 _edb_replicator_pub | rrep_publication_tables        | table | pubuser
 _edb_replicator_pub | rrep_tables                    | table | pubuser
 _edb_replicator_pub | rrep_tx_monitor                | table | pubuser
 _edb_replicator_pub | rrep_txset                     | table | pubuser
 _edb_replicator_pub | rrep_txset_health              | table | pubuser
 _edb_replicator_pub | rrep_txset_log                 | table | pubuser
 _edb_replicator_pub | rrep_wal_events_queue          | table | pubuser
 _edb_replicator_pub | rrst_edb_dept                  | table | pubuser
 _edb_replicator_pub | rrst_edb_emp                   | table | pubuser
 _edb_replicator_pub | xdb_cleanup_conf               | table | pubuser
 _edb_replicator_pub | xdb_conflicts                  | table | pubuser
 _edb_replicator_pub | xdb_conflicts_options          | table | pubuser
 _edb_replicator_pub | xdb_events                     | table | pubuser
 _edb_replicator_pub | xdb_events_status              | table | pubuser
 _edb_replicator_pub | xdb_MMR_pub_group              | table | pubuser
 _edb_replicator_pub | xdb_pub_database               | table | pubuser
 _edb_replicator_pub | xdb_pub_replog                 | table | pubuser
 _edb_replicator_pub | xdb_pub_table_replog           | table | pubuser
 _edb_replicator_pub | xdb_publication_filter         | table | pubuser
 _edb_replicator_pub | xdb_publication_filter_rule    | table | pubuser
 _edb_replicator_pub | xdb_publication_subscriptions  | table | pubuser
 _edb_replicator_pub | xdb_publications               | table | pubuser
 _edb_replicator_pub | xdb_pubtables_ignoredcols      | table | pubuser
 _edb_replicator_pub | xdb_sub_servers                | table | pubuser
(29 rows)

edb=# \ds
                    List of relations
       Schema        |      Name       |   Type   |  Owner
---------------------+-----------------+----------+---------
 _edb_replicator_pub | rrep_common_seq | sequence | pubuser
 _edb_replicator_pub | rrep_tx_seq     | sequence | pubuser
 _edb_replicator_pub | rrep_txset_seq  | sequence | pubuser
(3 rows)

edb=# SELECT nspname, pkgname FROM edb_package pk, pg_namespace ns
edb-#   WHERE nspname IN ('_edb_replicator_pub', '_edb_replicator_sub')
edb-#     AND pk.pkgnamespace = ns.oid;
       nspname       | pkgname
---------------------+----------
 _edb_replicator_pub | rrep_pkg
(1 row)

edb=# SELECT nspname, funname, typname FROM pg_function fn, pg_namespace ns,
edb-#     pg_type ty
edb-#   WHERE nspname = '_edb_replicator_pub'
edb-#     AND ns.oid = fn.funnamespace
edb-#     AND ty.oid = fn.funrettype
edb-#   ORDER BY typname, funname;
       nspname       |                   funname                   | typname
---------------------+---------------------------------------------+---------
 _edb_replicator_pub | capturetruncateevent                        | trigger
 _edb_replicator_pub | erep_filter_rule_delete_trigger_tgfunc      | trigger
 _edb_replicator_pub | erep_pub_database_trigger_tgfunc            | trigger
 _edb_replicator_pub | erep_publication_delete_trigger_tgfunc      | trigger
```

```
 _edb_replicator_pub | xdb_cleanup_conf_trigger_tgfunc               | trigger
 _edb_replicator_pub | xdb_conflicts_options_trigger_tgfunc         | trigger
 _edb_replicator_pub | xdb_conflicts_trigger_tgfunc                 | trigger
 _edb_replicator_pub | xdb_MMR_pub_group_trigger_tgfunc             | trigger
 _edb_replicator_pub | xdb_pub_database_trigger_tgfunc              | trigger
 _edb_replicator_pub | xdb_pub_replog_trigger_tgfunc                | trigger
 _edb_replicator_pub | xdb_pub_table_replog_trigger_tgfunc          | trigger
 _edb_replicator_pub | xdb_publication_filter_rule_trigger_tgfunc   | trigger
 _edb_replicator_pub | xdb_publication_filter_trigger_tgfunc        | trigger
 _edb_replicator_pub | xdb_publication_subscriptions_trigger_tgfunc | trigger
 _edb_replicator_pub | xdb_publications_trigger_tgfunc              | trigger
 _edb_replicator_pub | xdb_pubtables_ignoredcols_trigger_tgfunc     | trigger
 _edb_replicator_pub | xdb_sub_servers_trigger_tgfunc               | trigger
 _edb_replicator_pub | getpackageversionnumber                      | varchar
(18 rows)
```

The control schema objects contained in `\_edb_replicator_sub` are shown by the following:

```
 edb=# SET search_path TO _edb_replicator_sub;
SET
edb=# \dt
                        List of relations
       Schema        |          Name           | Type  |  Owner
---------------------+-------------------------+-------+---------
 _edb_replicator_sub | xdb_sub_database        | table | pubuser
 _edb_replicator_sub | xdb_subscription_tables | table | pubuser
 _edb_replicator_sub | xdb_subscriptions       | table | pubuser
 _edb_replicator_sub | xdb_tables              | table | pubuser
(4 rows)

edb=# \ds
                     List of relations
       Schema        |      Name       |   Type   |  Owner
---------------------+-----------------+----------+---------
 _edb_replicator_sub | rrep_common_seq | sequence | pubuser
(1 row)

edb=# SELECT nspname, funname, typname FROM pg_function fn, pg_namespace ns,
edb-#     pg_type ty
edb-#   WHERE nspname = '_edb_replicator_sub'
edb-#     AND ns.oid = fn.funnamespace
edb-#     AND ty.oid = fn.funrettype
edb-#   ORDER BY typname, funname;
       nspname       |               funname                | typname
---------------------+--------------------------------------+---------
 _edb_replicator_sub | xdb_sub_database_trigger_tgfunc      | trigger
 _edb_replicator_sub | xdb_subscription_tables_trigger_tgfunc | trigger
 _edb_replicator_sub | xdb_subscriptions_trigger_tgfunc     | trigger
 _edb_replicator_sub | xdb_tables_trigger_tgfunc            | trigger
(4 rows)
```

The control schema objects contained in `_edb_scheduler` are shown by the following:

```
 edb=# SET search_path TO _edb_scheduler;
SET
edb=# \dt
                      List of relations
     Schema     |            Name             | Type  |  Owner
----------------+-----------------------------+-------+---------
```

```
 _edb_scheduler | sch_pub_blob_triggers       | table | pubuser
 _edb_scheduler | sch_pub_calendars           | table | pubuser
 _edb_scheduler | sch_pub_cron_triggers       | table | pubuser
 _edb_scheduler | sch_pub_fired_triggers      | table | pubuser
 _edb_scheduler | sch_pub_job_details         | table | pubuser
 _edb_scheduler | sch_pub_job_listeners       | table | pubuser
 _edb_scheduler | sch_pub_locks               | table | pubuser
 _edb_scheduler | sch_pub_paused_trigger_grps | table | pubuser
 _edb_scheduler | sch_pub_scheduler_state     | table | pubuser
 _edb_scheduler | sch_pub_simple_triggers     | table | pubuser
 _edb_scheduler | sch_pub_trigger_listeners   | table | pubuser
 _edb_scheduler | sch_pub_triggers            | table | pubuser
 _edb_scheduler | sch_sub_blob_triggers       | table | pubuser
 _edb_scheduler | sch_sub_calendars           | table | pubuser
 _edb_scheduler | sch_sub_cron_triggers       | table | pubuser
 _edb_scheduler | sch_sub_fired_triggers      | table | pubuser
 _edb_scheduler | sch_sub_job_details         | table | pubuser
 _edb_scheduler | sch_sub_job_listeners       | table | pubuser
 _edb_scheduler | sch_sub_locks               | table | pubuser
 _edb_scheduler | sch_sub_paused_trigger_grps | table | pubuser
 _edb_scheduler | sch_sub_scheduler_state     | table | pubuser
 _edb_scheduler | sch_sub_simple_triggers     | table | pubuser
 _edb_scheduler | sch_sub_trigger_listeners   | table | pubuser
 _edb_scheduler | sch_sub_triggers            | table | pubuser
(24 rows)

edb=# SELECT nspname, funname, typname FROM pg_function fn, pg_namespace ns,
edb-#     pg_type ty
edb-#   WHERE nspname = '_edb_scheduler'
edb-#     AND ns.oid = fn.funnamespace
edb-#     AND ty.oid = fn.funrettype
edb-#   ORDER BY typname, funname;
    nspname     |                 funname                  | typname
----------------+------------------------------------------+---------
 _edb_scheduler | sch_pub_blob_triggers_trigger_tgfunc     | trigger
 _edb_scheduler | sch_pub_calendars_trigger_tgfunc         | trigger
 _edb_scheduler | sch_pub_cron_triggers_trigger_tgfunc     | trigger
 _edb_scheduler | sch_pub_job_details_trigger_tgfunc       | trigger
 _edb_scheduler | sch_pub_job_listeners_trigger_tgfunc     | trigger
 _edb_scheduler | sch_pub_simple_triggers_trigger_tgfunc   | trigger
 _edb_scheduler | sch_pub_trigger_listeners_trigger_tgfunc | trigger
 _edb_scheduler | sch_pub_triggers_trigger_tgfunc          | trigger
 _edb_scheduler | sch_sub_blob_triggers_trigger_tgfunc     | trigger
 _edb_scheduler | sch_sub_calendars_trigger_tgfunc         | trigger
 _edb_scheduler | sch_sub_cron_triggers_trigger_tgfunc     | trigger
 _edb_scheduler | sch_sub_job_details_trigger_tgfunc       | trigger
 _edb_scheduler | sch_sub_job_listeners_trigger_tgfunc     | trigger
 _edb_scheduler | sch_sub_simple_triggers_trigger_tgfunc   | trigger
 _edb_scheduler | sch_sub_trigger_listeners_trigger_tgfunc | trigger
 _edb_scheduler | sch_sub_triggers_trigger_tgfunc          | trigger
(16 rows)
```

In addition, triggers and trigger functions are created in the schema containing the publication tables if the trigger-based method of synchronization replication is used.

```
 edb=# SET search_path TO edb;
SET
edb=# \df rr*
                               List of functions
 Schema |         Name         | Result data type | Argument data types |  Type
```

```
--------+----------------------+------------------+---------------------+---------
 edb    | rrpd_edb_dept_tgfunc | trigger          |                     | trigger
 edb    | rrpd_edb_emp_tgfunc  | trigger          |                     | trigger
 edb    | rrpi_edb_dept_tgfunc | trigger          |                     | trigger
 edb    | rrpi_edb_emp_tgfunc  | trigger          |                     | trigger
 edb    | rrpu_edb_dept_tgfunc | trigger          |                     | trigger
 edb    | rrpu_edb_emp_tgfunc  | trigger          |                     | trigger
(6 rows)
```

If the log-based method of synchronization replication is used, the following triggers are created on the publication tables:

```
 edb=# SELECT t.tgname AS "Trigger Name", c.relname AS "Table Name",
edb-#        f.funname AS "Trigger Function"
edb-#   FROM pg_trigger t, pg_function f, pg_class c
edb-#   WHERE tgname LIKE 'rrpt%'
edb-#     AND t.tgfoid = f.oid
edb-#     AND t.tgrelid = c.oid
edb-#   ORDER BY t.tgname;
 Trigger Name  | Table Name |   Trigger Function
---------------+------------+----------------------
 rrpt_edb_dept | dept       | capturetruncateevent
 rrpt_edb_emp  | emp        | capturetruncateevent
(2 rows)
```

These triggers are used to support synchronization replication of the `TRUNCATE` command when using the log-based method.

8.3 Creating a subscription

Creating your first subscription requires the following steps:

- Registering the subscription server
- Adding the subscription database
- Creating a subscription by choosing the publication to subscribe to

You can create multiple subscriptions in a subscription database. You can also create more than one subscription to subscribe against the same publication.

8.3.1 Registering a subscription server

When you register a subscription server, you're identifying the network location, admin user name, and password of a specific, running, subscription server instance that you want to use to manage all aspects of the subscriptions you create subordinate to it.

It's important that you record the login information for the subscription server. You must always use this same subscription server instance to manage all subscriptions created subordinate to it as represented in the Replication Server console replication tree.

1. Start the subscription server if it's not already running. Repeat the same process as in Step 1 of Registering a publication server.

> **Note**
>
> If you're using Oracle publication or subscription databases and the subscription server wasn't restarted after copying the Oracle JDBC driver to the `lib/jdbc` subdirectory of your Replication Server installation, restart the subscription server.

For Linux only: Use the `systemctl` command for CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8, and the service command for previous Linux versions to start, stop, or restart `edb-xdbsubserver` for the subscription server. See Registering a Publication Server for information on how these commands are used.

For Windows only: Open **Control Panel > System and Security > Administrative Tools > Services**. Use the **Start** or **Restart** link for the service named Subscription Service for Replication Server.

If the subscription server doesn't start, see Publication and subscription server startup failures.

2. Register the subscription server. Open the Replication Server console from the system's application menu. For Replication Server installed from a Replication Server RPM package, start the Replication Server console by invoking the script `XDB_HOME/bin/runRepConsole.sh`.

3. Select the top-level Replication Servers node. Select **File > Subscription Server > Register Server**.

   In the `Register Subscription Server` dialog box, enter the values you supplied during the Replication Server installation unless otherwise specified.

   - **Host**. Network IP address of the host running the subscription server. This is the network IP address used for `sub_ipaddr` in the `pg_hba.conf` file in Postgres server authentication. Don't use `localhost` for this field.
   - **Port**. Port number the subscription server is using. You can find the port number in the `xdbReplicationServer-xx.config file`. See Installation details for the location of the file.
   - **User Name**. Admin user name used to authenticate your use of this subscription server. You can find the user name in the `edb-repl.conf` file. See Installation details for the location of the file.
   - **Password**. Password of the admin user given in the **User Name** field.
   - **Save login information**. Select this box if you don't want to register the subscription server each time you open the Replication Server console. See Saving server login information for additional information on the advantages and disadvantages of saving server login information.

   > **Note**
   >
   > The user name and password combination you enter is authenticated against the admin user name and password in the Replication Server configuration file residing on the host with the IP address you enter in the **Host** field.

   After you fill in the fields, select **Register**. A Subscription Server node appears in the replication tree of the Replication Server console.

8.3.2 Adding a subscription database

You must identify to Replication Server the database for subscriptions. You do this buy creating a subscription database definition.

After you create the subscription database definition, a Subscription Database node representing that subscription database definition appears in the replication tree of the Replication Server console. Subscriptions created subordinate to this subscription database definition have their publications replicated to the database identified by the subscription database definition.

You must enter database connection information such as the database server network address, database identifier, and database login user name and password when you create the subscription database definition. The subscription server uses the connection information to create the subscription table definitions. The publication server uses this information to perform replications.

Note the following restriction on the subscription database:

- **For Oracle only.** There must be no existing tables or views owned by the Oracle subscription database user that has the same name as a table or view in a publication replicated to this database. For example, if the Oracle subscription database user name is `subuser`, and if a Postgres publication contains a table with the name `dept`, then the Oracle subscription database must not have an existing table or view with the schema-qualified name `subuser.dept` when you create the subscription.
- **For Postgres only.** There must be no existing tables or views with the same schema-qualified name as a table or view in a publication replicated to this database. For example, if the publication contains a table with the schema-qualified name `edb.dept`, then the Postgres subscription database must not have an existing table or view with the schema-qualified name `edb.dept` when you create the subscription.

> **Note**
>
> If the SQL Server publication schema name is `dbo`, the subscription tables are created under a schema named `dbo_sql` in Postgres.

- **For SQL Server only.** There must be no existing tables or views with the same schema-qualified name as a table or view in a publication replicated to this database. For example, if the publication contains a table with the schema-qualified name `edb.dept`, then the SQL Server subscription database must not have an existing table or view with the schema-qualified name `edb.dept` when you create the subscription.

> **Note**
>
> If the Postgres publication schema name is `public`, the subscription tables are created under a schema named `public_sql` in SQL Server.

> **Note**
>
> You can use a database added as a publication database as a subscription database.

1. Make sure the database server where the subscription database resides is running and accepting client connections.

2. Select the Subscription Server node. Select **Subscription > Subscription Database > Add Database**.

3. In the Subscription Service – Add Database dialog box, fill in the following fields:

   - **Database Type**. Select **Oracle**, **SQL Server**, **PostgreSQL**, or **Postgres Plus Advanced Server** for the type of subscription database. For an EDB Postgres Advanced Server Oracle-compatible installation, select the **Postgres Plus Advanced Server** option. For PostgreSQL or an EDB Progres Advanced Server PostgreSQL-compatible installation, select the **PostgreSQL** option.
   - **Host**. IP address of the host where the subscription database server is running.
   - **Port**. Port on which the subscription database server is listening for connections.
   - **User**. The subscription database user name:
     - Selected in Postgres subscription database for a Postgres subscription database
     - Created in Step 2 of Oracle subscription database for an Oracle subscription database
     - created in Step 2 of SQL Server subscription database for a SQL Server subscription database

   - **Password**. Password of the database user.
   - **Service ID (For Oracle)**. Enter the Oracle system identifier (SID) of the Oracle instance running the subscription database if **SID** is selected. Enter the net service name of a connect descriptor as defined in the `TNSNAMES.ORA` file if **Service Name** is selected. Note for Oracle 12c pluggable database): Use the service name.
   - **Database (For Postgres or SQL Server)**. Enter the Postgres or SQL Server database name.
   - **URL Options (For SSL connectivity)**. Enter the URL options to establish SSL connectivity to the subscription database. See Using secure sockets layer (SSL) connections.

4. Select **Test**. When Test Result: Success appears, select **OK**, and then select **Save**.

   A Subscription Database node is added to the replication tree under the Subscription Server node.

8.3.3 Adding a subscription

Subordinate to a subscription database definition, you create subscriptions. A subscription assigns the publication to replicate to the database identified by the subscription database definition.

1. Select the Subscription Database node. Select **Subscription > Create Subscription**.

2. In the Create Subscription dialog box, fill in the following fields:

   - **Subscription Name**. Enter a name for the subscription that is unique among all subscription names.
   - **Host**. Network IP address of the publication server that is the parent node of the publication to subscribe to. This IP address is the same value entered in the **Host** field in Step 3 of Registering a publication server.

- **Port**. Port used by the publication server. This is the same value entered in the **Port** field in Step 3 of Registering a publication server.
- **User Name**. Admin user name of the publication server. This is the same value entered in the **User Name** field in Step 3 of Registering a publication server.
- **Password**. Password of the admin user. This is the same value entered in the **Password** field in Step 3 of Registering a publication server.
- **Publication Name**. Select **Load** to get a list of available publications. Select the publication to subscribe to.

3. (Optional) If you defined a set of available table filters for the publication, you have the option of enabling these filters on this subscription. See Adding a publication for instructions on defining table filters.

   Select the **Filter Rules** tab to enable one or more filter rules on the subscription. You can enable at most one filter rule on any given subscription table.

4. Select **Create**. When Subscription Created Successfully appears, select **OK**.

   A Subscription node is added to the replication tree.

   The tables and views from the publication are created in the subscription database but without any rows. Rows are populated into the subscription tables when the first snapshot replication occurs.

8.3.4 Subscription metadata object

After you add a subscription database definition, a single table named `rrep_txset_health` is created as the subscription metadata object.

For Oracle only: The `RREP_TXSET_HEALTH` table is created in the subscription database user's schema as shown in the following output:

```
 SQL> CONNECT subuser/password
Connected.
SQL> SET PAGESIZE 9999
SQL> SELECT table_name FROM user_tables ORDER BY table_name;

TABLE_NAME
------------------------------
RREP_TXSET_HEALTH
```

For SQL Server only: The `rrep_txset_health` table is created in the schema named `_edb_replicator_sub`.

```
 1> USE subdb;
2> GO
Changed database context to 'subdb'.
1> SELECT s.name + '.' + o.name "Object Name", o.type_desc "Object Type"
2>   FROM sys.objects o,
3>        sys.schemas s
4>   WHERE s.name <> 'edb'
5>     AND o.type IN ('U','P','FN')
6>     AND o.schema_id = s.schema_id
7>   ORDER BY 2, 1;
8> GO
Object Name                           Object Type
------------------------------------- ---------------------------------------
_edb_replicator_sub.rrep_txset_health  USER_TABLE
(1 rows affected)
```

For Postgres only: The `rrep_txset_health` table is created in the schema named `_edb_replicator_sub`.

```
 subdb=# SET search_path TO _edb_replicator_sub;
SET
subdb=# \dt
                    List of relations
       Schema        |       Name        | Type  |  Owner
---------------------+-------------------+-------+---------
 _edb_replicator_sub | rrep_txset_health | table | subuser
(1 row)
```

In all subscription database types (Oracle, SQL Server, and Postgres), when you remove the subscription database definitions using the Replication Server console or CLI, the subscription metadata object is deleted from the subscription database.

8.4 On-demand replication

After you create a publication and subscription, you can start the replication process in either of these ways:

- You can perform replication immediately by taking an on-demand snapshot.
- You can schedule replication to start later by creating a schedule. See Creating a schedule.

8.4.1 Performing snapshot replication

You must perform the first replication using snapshot replication. After the first snapshot replication, you can perform later replications using either the synchronization method (if the publication wasn't initially defined as a snapshot-only publication) or the snapshot method.

1. In the Replication Server console, select the Subscription node of the subscription for which you wish to perform snapshot replication.

2. Select **Subscription > Snapshot**.

3. Select **Verbose Output** if you want to display the output from the snapshot in the dialog box. Don't select this option in a network address translation (NAT) environment, as a large amount of output from the snapshot can delay the response from the Snapshot dialog box.

4. Select **Snapshot** to start snapshot replication.

   Snapshot Taken Successfully appears if the snapshot was successful.

5. Select **OK**. If the snapshot wasn't successful, scroll through the messages in the Snapshot dialog box if**Verbose Output** was selected or check the log files.

The status messages of each snapshot are saved in the Migration Toolkit log files named `mtk.log[.n]` in the following directories. `[.n]` is an optional history file count if log file rotation is enabled.

For Linux:

`/var/log/xdb-x.x`

For Windows:

`POSTGRES_HOME\.enterprisedb\xdb\x.x`

`POSTGRES_HOME` is the home directory of the Windows postgres account (enterprisedb account for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode). The specific location of `POSTGRES_HOME` depends on your version of Windows. The Replication Server version number

is represented by `x.x`.

The publication is now replicated to the subscription database. A record of the snapshot is kept in the replication history. See Viewing replication history for more information.

> **Note**
>
> Before version 7.0.0, Replication Server required superuser privileges to disable/enable constraints and indexes as part of its snapshot operation. Starting with version 7.0.0, the superuser privileges are no longer required, and the constraints/indexes are now dropped and re-created as part of the snapshot operation. This change might cause any views/materialized views with a dependency on the constraints from the target subscription/master database to be dropped. These views are then re-created at the end of snapshot operation. Depending on the permissions assigned to the Replication Server subscription database user, the re-created views/materialized views might have permissions that are inconsistent with those of the originally assigned permissions. Therefore, we recommend manually verifying the assigned permissions of the relevant views/materialized views in the target database and making any corrections where applicable.

8.4.2 Performing synchronization replication

After the first snapshot replication, you cam perform later replications using synchronization replication if the publication wasn't created as a snapshot-only publication.

1. In the Replication Server console, when the trigger-based method of synchronization replication is in use, select the Subscription node of the subscription for which you want to perform synchronization replication.

   When the log-based method of synchronization replication is in use, select the Subscription node of any subscription. For the log-based method, the synchronization replication is performed on all subscriptions regardless of which one you select.

2. Select **Subscription > Synchronize**.

3. In the Synchronize dialog box, select **Synchronize** to start synchronization replication. Subscription Synchronized Successfully appears if the synchronization was successful.

4. Select **OK**.

You can see the operations that were applied to the subscription tables in the replication history. See Viewing replication history for more information.

8.5 Managing a subscription

After you create a subscription, you might later change aspects of the underlying replication system environment. Attributes that might change include:

- The network location of the subscription database server
- The network location of the host running the subscription server, database, or operating system user names and passwords, and so forth

This information is saved in the replication system metadata when you create a subscription. Changes to these attributes result in inaccurate replication system metadata, which in turn can result in errors during subsequent replication attempts or replication system administration.

You can update the metadata stored for the subscription server, the subscription database definition, and subscriptions to keep the information consistent with the actual replication system environment.

> **Note**
>
> For managing a publication of a replication system, see Managing a publication.

Update a subscription server

When you register a subscription server in the Replication Server console, you can choose to save the subscription server's network location (IP address and port number), admin user name, and encrypted password in a server login file on the computer on which you're running the Replication Server console. See Saving server login information for details.

To update the subscription server, first open the Replication Server console. The subscription server whose login information you want to alter in the server login file must appear as a Subscription Server node in the replication tree.

You can perform the following actions on the server login file:

- Change the subscription server's login information (host IP address, port number, admin user name, and password) that you last saved in the server login file.
- Delete the subscription server's login information that's currently saved in the server login file. This is the default action and requires you to register the subscription server again the next time you open the Replication Server console.
- Resave the subscription server's login information in the server login file. Each time you open the Update Subscription Server dialog box, you must choose to save the login information if you want it recorded in the server login file.

The following steps change only the content of the server login file residing on the host under the current Replication Server console user's home directory. These changes don't alter any characteristic of the actual subscription server daemon (on Linux) or service (on Windows). These changes affect only how a subscription server is viewed through the Replication Server console on this host by this user.

The subscription server whose login information you want to save, change, or delete in the server login file must be running before you can make any changes to the file. See Step 1 of Registering a subscription server for instructions on starting the subscription server.

1. Right-click the Subscription Server node and select **Update**.

2. In the Update Subscription Server dialog box, complete the fields according to your purpose for updating the server login file:

    - If the subscription server now runs on a host with a different IP address or port number from what's shown in the dialog box, enter the correct information. You must also enter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address you entered in the **Host** field. Select **Save Login Information** if you want the new login information saved in the server login file. If you don't select this option, access to the subscription server is available for the current session, but for later sessions you must register the subscription server again.
    - If you want to delete previously saved login information, make sure the network location shown in the dialog box is still correct. Reenter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address in the **Host** field. Leave the **Save Login Information** box cleared. Access to the subscription server is available for this session, but for later sessions you must register the subscription server again.
    - If you want to save the current login information shown in the dialog box, make sure the network location shown in the dialog box is correct. Reenter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address in the **Host** field. Select **Save Login Information**.

3. Select **Update**. The dialog box closes after the update to the server login file completes. Select **Refresh** in the tool bar to show the updated Subscription Server node.

Updating a subscription database

When you create a subscription database definition, you save the subscription database server's network location (IP address and port number), the database identifier, a database login user name, and the user's password in the control schema accessed by the subscription server. This login information is used whenever you need to start a session with the subscription database. See Adding a subscription database for information on creating a subscription database definition.

Update the subscription database login information stored in the control schema if any of these attributes of the physical database change.

> **Note**
>
> Depending on the database type (Oracle, SQL Server, or Postgres), you must not change certain attributes. If you already added subscriptions,

> you must not change any attribute that alters access to the schema where the subscription tables were created.

Attributes you must not change if existing subscriptions include the following:

- The Oracle login user name, as the subscription tables already reside in this Oracle user's schema
- The database server network location if the new network location references a database server that doesn't access the database that already contains the subscription tables
- The database identifier if the new database identifier references a different physical database from where the subscription tables already reside

With no existing subscriptions, you can change any attribute. With existing subscriptions, you can change these attributes:

- The login user name's password to match a changed database user password
- The database server network location if the corresponding location change was made to the database server that accesses the subscription database
- The database identifier, such as the Oracle service name, SQL Server database name, or Postgres database name if the corresponding name change was made on the database server

1. Make sure the database server that you want to save as the subscription database definition is running and accepting client connections.

2. Make sure the subscription server whose node is the parent of the subscription database definition you want to change is running and registered in the Replication Server console you're using. See Registering a subscription server for instructions on starting and registering a subscription server.

3. Select the Subscription Database node corresponding to the subscription database definition that you want to update.

4. Select **Subscription > Subscription Database > Update Database**.

5. In the Update Database Source dialog box, enter your changes. See Step 3 of Adding a subscription database for the meanings of the fields.

6. Select **Test**. When Test Result: Success appears, select **OK**, and then select **Save**.

7. Select **Refresh** in the tool bar to show the updated Subscription Database node and any of its subscriptions.

Updating a subscription

When you create a subscription, certain attributes of the subscribed publication are stored as part of the metadata for the subscription in the control schema. These include the following:

- The network IP address of the host running the publication server that's the parent of the subscribed publication
- The port number of the publication server

If these attributes of the publication server change in the replication system environment, then you must also change the corresponding subscription metadata so the subscription server can communicate with the correct publication server.

You can update the publication server network IP address and port number in the subscription server's metadata.

1. Make sure the subscription server whose node is the parent of the subscription you want to change is running and is registered in the Replication Server console. See Registering a subscription server for instructions on starting and registering a subscription server.

2. Select the Subscription node whose attributes you want to update.

3. Select **Subscription > Update Subscription**.

4. If the publication server now runs on a host with a different IP address or port number from what's shown in the dialog box, enter the correct information in the Update Subscription dialog box. You must also enter the admin user name and password saved in the Replication Server configuration file that resides on the host on which the publication server is running.

5. Select **Update**.

6. When Subscription Updated Successfully appears, select **OK**.

7. If the publication server with the new network location manages publications subscribed to by other subscriptions, repeat steps 1 through 5 for these other subscriptions.

Enabling and disabling table filters on a subscription

You must first define table filters in a set of available table filters in the publication before you can enable them on a subscription. See Adding a publication for information on defining table filters in a single-master replication system.

To enable or disable table filters on an existing subscription:

1. Make sure the publication server whose node is the parent of the publication associated with the subscription you want to change is running and registered in the Replication Server console you're using. See Registering a publication server for instructions on starting and registering a publication server.

2. Make sure the subscription server whose node is the parent of the subscription you want to change is running and registered in the Replication Server console you're using. See Registering a subscription server for instructions on starting and registering a subscription server.

3. Select the Subscription node of the subscription on which you want to enable or disable individual filter rules.

4. Select **Subscription > Update Filter Rule**.

5. In the Filter Rules dialog box, select options to specify the filter rules to enable or disable on the subscription. You can enable at most one filter rule on any given subscription table.

6. Select **Update**.

   A confirmation box appears with a warning message and a recommendation to perform a snapshot replication to any subscription on which you changed the filtering criteria.

7. To proceed with the update to the filter rule selections, select **OK**.

   When the update is complete, Filter Rules Updated Successfully appears.

8. We recommend that you perform a snapshot replication on the subscription that contains tables on which you changed the filtering criteria.

   A snapshot ensures that the content of the subscription tables is consistent with the updated filtering criteria. See Performing snapshot replication for details.

Removing a subscription

After you remove a subscription, replication can no longer occur for the publication that was associated with it until the publication is subscribed to with a new subscription.

Removing a subscription doesn't delete the subscription tables in the subscription database. It removes the identity and association of these tables to Replication Server. The tables remain in the database until the DBA deletes them with `DROP TABLE SQL` statements.

1. Make sure the subscription server whose node is the parent of the subscription you want to remove is running and registered in the Replication Swerver console you're using. See Registering a subscription server for instructions on starting and registering a subscription server.

2. Select the Subscription node of the subscription that you want to remove.

3. Select **Subscription > Remove Subscription**.

4. In the Remove Subscription confirmation box, select **Yes**.

The Subscription node no longer appears under the Subscription Database node.

Removing a subscription database

Removing a subscription database definition from Replication Server is equivalent to removing its Subscription Database node. Before you can remove a Subscription Database node, you must remove all subscriptions under that Subscription Database node. See Removing a subscription for details.

Removing a Subscription Database node doesn't delete the physical database from the database server. It removes the identity and association of the database to Replication Server. In this case, no further replications can create or update tables in the database unless there are other subscription database definitions in Replication Server with the same host and database identifier. You can remove the physical database only using the database management system's database removal procedures.

1. Make sure the subscription server whose node is the parent of the subscription database definition you want to remove is running and registered in the Replication Server console you're using. See Registering a subscription server for instructions on starting and registering a subscription server.

2. Select the Subscription Database node that you want to remove.

3. Select **Subscription > Subscription Database > Remove Database**.

4. In the Remove Subscription Database confirmation box, select **Yes**.

The Subscription Database node no longer appears under the Subscription Server node.

8.6 Performing controlled switchover

Controlled switchover is the exchanging of roles between a publication database and a subscription database. The tables that were formerly publications become the subscription tables. The former subscription tables become the publications.

Controlled switchover is useful for situations in which you must take the publication database offline, such as for periodic maintenance. After the switchover, applications connect to the former subscription database to perform their queries and updates, while the former publication database is kept synchronized by replication.

Updates for replication are accumulated in shadow tables that are created on the former subscription tables during the controlled switchover procedure. When the former publication database is online, it's synchronized as the target of replication.

When you determine that you want to reverse the roles again so that the original publication database directly receives queries and updates from applications and the original subscription database receives updates by replication, you perform the controlled switchover procedure again, switching the roles back.

> **Note**
>
> This discussion assumes that the trigger-based method of synchronization replication is used by the publication database. If the publication database uses the log-based method, then you must determine whether the current subscription database meets the criteria for using the log-based method if you want to when it is switched to the role of the publication database. If the subscription database doesn't meet the criteria, then you must implement and use the trigger-based method. See Synchronization replication with the log-based method for information on the log-based method and the configuration steps to perform when using the log-based method.

Controlled switchover overview

When you perform controlled switchover, you're modifying the replication system so that the database identified and referenced in the control schema as the publication database is the physical database that was originally defined as the subscription database.

Similarly, the database identified and referenced in the control schema as the subscription database is changed to the physical database on which the publication was originally defined.

You must also create the database objects on the former subscription database that Replication Server uses to capture and store updates for replication. To accomplish the controlled switchover, perform the following:

- Copy the control schema of the publication database (that is, all control schema objects, shadow tables, sequences, triggers, and a package) to the subscription database.
- Copy the control schema of the subscription database to the publication database.
- Update certain control schema tables to exchange the connection information for the publication database and subscription database. Make these in the control schema of all publication databases to ensure consistency of the control schema across all publication databases.
- Modify the Replication Server configuration file to reference a new controller database if the former publication database was the designated controller database.

Controlled switchover steps

In these examples, the following assumptions are made about the replication system environment:

- Node 1 is the server where the publication database originally resides. Its network IP address is 192.168.2.19.
- Node 2 is the server where the subscription database originally resides. Its network IP address is 192.168.2.20.
- The publication and subscription databases have the same name.
- You use the publication database user for the role of the subscription database user and the subscription database user for the role of the publication database user in the switched environment.
- The publication server and subscription server are running on the same host (node 1).

1. Stop all transaction processing against the publication database.

2. Perform an on-demand synchronization replication or a snapshot replication (for snapshot-only publications) to replicate any pending updates in the publication database shadow tables to the subscription database.

3. Stop the publication server and the subscription server.

4. Review the prerequisites in Prerequisite steps to ensure that you can use the subscription database and its host in the role of a publication database. Also make sure that you can use the publication database and its host in the role of a subscription database.

The following items are the most likely to be affected:

- The publication database user must be a superuser with system catalog modification privileges to allow it to act as the new subscription database user.
- Additional entries might be needed in the `pg_hba.conf` files.

1. Create a backup of schemas `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler` from the publication database on node 1. Delete these schemas from the publication database on node 1 after making the backup.

2. Create a backup of the replication triggers and their corresponding trigger functions on the publication tables on node 1. For the trigger-based method, these triggers have the prefixes `rrpd_`, `rrpi_`, and `rrpu_`. The trigger functions have the same prefixes. For the log-based method, a trigger for each table has the prefixed `rrpt_`. The function is named `capturetruncateevent`.

   Delete or disable these triggers on node 1.

3. Create a backup of schema `_edb_replicator_sub` from the subscription database on node 2. Delete this schema from the subscription database on node 2 after making the backup.

4. Restore the backups of `schemas _edb_replicator_pub` , `_edb_replicator_sub` , and `_edb_scheduler` to the subscription database on node 2. Also restore the backup of the replication triggers and trigger functions to the subscription database on node 2.

5. Restore the backup of schema `_edb_replicator_sub` to the publication database on node 1.

6. Update the control schema objects so that the publication database definition references the new publication database (that is, the former subscription database) on node 2 and the subscription database definition references the new subscription database (that is, the former publication database) on node 1.

The connection information that might require updating includes:

- Host IP address
- Port number
- User name
- Password

Make these updates in the control schema of all publication databases to ensure consistency of the control schema information in case the controller database is switched later.

For example, the following shows the update to the publication database definition so that its network IP address is now node 2 (`192.168.2.20`).

```
UPDATE _edb_replicator_pub.xdb_pub_database SET db_host = '192.168.2.20';
```

The following shows the update to the subscription database definition so that its network IP address is now node 1 (192.168.2.19).

```
UPDATE _edb_replicator_sub.xdb_sub_database SET db_host = '192.168.2.19';
```

1. If you decide to use a publication server or subscription server on a new host, perform this step.

The following example assumes you decide to use the publication server and subscription server running on node 2. Update the subscription metadata to the new location of the publication server managing its associated publication.

```
UPDATE _edb_replicator_sub.xdb_subscriptions SET pub_server_ip = '192.168.2.20';
```

Update the publication metadata to the new location of the subscription server managing its associated subscription.

```
UPDATE _edb_replicator_pub.xdb_sub_servers SET sub_server_ip = '192.168.2.20';
```

1. Edit the Replication Server configuration file on the publication server and subscription server host so that it contains the controller database connection and authentication information for the new publication database now running on node 2.

The following is the modified Replication Server configuration file with the network location and authentication information of the new controller database now running on node 2.

```
#xDB Replication Server Configuration Properties
#Fri Jan 30 17:34:06 GMT-05:00 2015
port=5444
admin_password=ygJ9AxoJEX854elcVIJPTw\=\=
user=enterprisedb
```

```
admin_user=enterprisedb
type=enterprisedb
database=edb
password=ygJ9AxoJEX854elcVIJPTw\=\=
host=192.168.2.20
```

1. Update the `pg_hba.conf` files of the database servers to allow access to the subscription database now on node 1 and the publication database now on node 2 in accordance with Postgres server authentication.

2. When using the log-based method, create a replication slot on the database server that now contains the publication database.

Use the following query to get the slot name from the database server that was previously running the publication database but is now the subscription database server:

```
SELECT slot_name FROM pg_replication_slots WHERE plugin =
'test_decoding';
  slot_name
-------------

xdb_47919_5
(1 row)
```

Create a new replication slot on the database server that is now running the publication database but was previously the subscription database server. The slot name from the previous query is used when creating the new replication slot.

```
SELECT pg_create_logical_replication_slot('xdb_47919_5', 'test_decoding');
pg_create_logical_replication_slot
-----------------------------------
 (xdb_47919_5,0/37A1270)
(1 row)
```

You might choose to keep the replication slot on the database server that now contains the subscription database, particularly if you plan to switch the publication and subscription databases back to their original roles later. Keeping the replication slot on the database server eliminates the need to recreate the replication slot since it still exists but is inactive until the publication is switched back to that database server.

Alternatively, you can delete the replication slot from the database server that now contains the subscription database. Delete the replication slot with the following command:

```
SELECT pg_drop_replication_slot('xdb_47919_5');
```

See Dropping replication slots for log-based synchronization replication for more information on deleting the replication slot if the `pg_drop_replication_slot` function isn't successful. If you switch the databases back to their original roles, you have to recreate the replication slot on the publication database server as described in this step.

1. The controlled switchover is now complete. Start the publication server and the subscription server.

After confirming that the publication tables are consistent with the subscription tables, the first replication operation must be a snapshot. After performing a snapshot, you can perform synchronization replications.

8.7 Performing failover

Failover is replacing the publication database with the subscription database if a failure occurs on the publication database or its host. Failover is an irreversible action, so the subscription database permanently takes over the role of the publication database.

Generally, to perform a failover, follow the same steps as for a controlled switchover, described in Performing controlled switchover. However, also take the following points into consideration:

- If you can't salvage the control schema objects on the publication database (that is, schemas `_edb_replicator_pub`, `_edb_replicator_sub`, `_edb_scheduler`, and their objects) or restore them from a backup, then performing a failover might be possible only with the help of EnterpriseDB Technical Support Services.
- Pending updates not yet applied to the subscription might be lost. The chances of this are greater if the interval between synchronizations is long.

If you determine that a failover is possible, follow the steps for a controlled switchover.

8.8 Optimizing performance

Once you're familiar with setting up and managing your replication system, you mmight look for ways to optimize the performance of replications. Various publication server and subscription server configuration options are available for improving the performance of snapshot and synchronization replications.

Set the publication server and subscription server configuration options in the publication server and subscription server configuration files. See Publication and subscription server configuration options for a detailed explanation of how to set the configuration options in these files.

> **Note**
>
> Most of these configuration options also apply to multi-master replication systems. Options that apply to multi-master replication systems are those that apply to the publication server. They aren't specific to a database product other than Postgres (such as an Oracle feature).

8.8.1 Optimizing snapshot replication

You can set various configuration options to improve snapshot replication performance.

These options apply only to the publication server. You set them in the publication server configuration file unless otherwise specified.

`copyViaDBLinkOra`

When the `copyViaDBLinkOra` option is set to true, the Oracle database link API, `dblink_ora`, is used instead of JDBC `COPY` to populate EDB Postgres Advanced Server subscription tables from an Oracle publication during snapshot replication.

Oracle database link provides an additional performance improvement over JDBC `COPY`.

> **Note**
>
> The Oracle database link API feature isn't available with PostgreSQL, therefore the copyViaDBLinkOra option doesn't apply to PostgreSQL subscription tables.

> **Note**
>
> Prior to using `dblink_ora` with Replication Server, you must perform some required configuration steps in EDB Postgres Advanced Server. For EDB Postgres Advanced Server versions 9.3 or earlier, see `README-dblink_ora_setup.txt` located in the `POSTGRES_INSTALL_HOME/doc/contrib` directory for instructions. For EDB Postgres Advanced Server versions 9.4 or later, see dblink_ora in Database compatibility for Oracle developers.

`copyViaDBLinkOra={true | false}`

The default value is false.

`useFastCopy`

Set the `useFastCopy` option to `true` to skip write-ahead log (WAL) logging during `COPY` operations to optimize data transfer speed.

The `archive_mode` configuration parameter in the `postgresql.conf` file of the target Postgres database server must be off (thereby disabling archiving of WAL data) to use the `useFastCopy` option.

`useFastCopy={true | false}`

The default value is false.

`cpBatchSize`

Use the `cpBatchSize` option to set the batch size (in MB) to use in the JDBC `COPY` operation during a snapshot. Increase the value of this option for large publication tables.

This option is influential when Postgres is the subscription database since the JDBC `COPY` operation is used to load Postgres subscription tables.

This option has no effect when Oracle or SQL Server is the subscription database. To tune loading of Oracle or SQL Server tables, change the `batchSize` option.

`cpBatchSize=n`

The default value for n is 8.

`batchSize`

The `batchSize` option controls the number of `INSERT` statements in a JDBC batch.

This option is particularly important when Oracle or SQL Server is the subscription database since tables of these database types are loaded using JDBC batches of `INSERT` statements.

For a Postgres subscription database, tables are loaded using JDBC `COPY`. However, if the `COPY` operation fails, then table loading is retried using JDBC batches of `INSERT` statements as in the case of Oracle and SQL Server.

`batchSize=n`

The default value for n is 100.

`skipAnalyze`

Set the `skipAnalyze` option to true if you want to skip executing the `ANALYZE` command after loading Postgres subscription tables. The `ANALYZE` command gathers statistical information on the table contents. These statistics are used by the query planner.

`skipAnalyze={true | false}`

The default value is false.

`snapshotParallelLoadCount`

> **Note**
>
> To apply this option to a single-master replication system, you must set it for the subscription server in the subscription server configuration file.
> To apply this option to a multi-master replication system, set it for the publication server in the publication server configuration file.

The `snapshotParallelLoadCount` option controls the number of threads used to perform snapshot data replication in parallel mode. The default behavior is to use a single thread. However, if the target system architecture contains `multi-CPUs/cores`, you can specify a value greater than 1,

normally equal to the `CPU/core` count, to fully use the system resources.

`snapshotParallelLoadCount=n`

The default value is 1.

`lobBatchSize`

If a table contains a column with a data type typically used for large objects such as BYTEA, BLOB, or CLOB, a heap space error is more likely to occur because of a potentially large amount of data (hundreds of MB) brought into memory. To minimize the chance of this error, a snapshot replication loads tables containing a large object data type, one row at a time using a single `INSERT` statement per batch.

If, however, the large object data type column is known to contain relatively small amounts of data, you can increase the speed of a snapshot replication by increasing the value of the `lobBatchSize` option to allow a greater number of rows (specified by n) in each batch.

`lobBatchSize=n`

The default value is 1.

8.8.2 Optimizing synchronization replication

You can set various configuration options for improving synchronization replication performance.

These options apply only to the publication server. You set them in the publication server configuration file.

For configuration options that apply to publication databases configured with the log-based method of synchronization replication, see Specifying a custom URL for an Oracle JDBC connection.

8.8.2.1 Using prepared SQL statements

When synchronization replication occurs, the changes recorded in the shadow tables are applied to the subscription tables in JDBC batch updates. In each batch, you can apply changes using either an individual SQL statement for each change, or you can can apply a set of changes using a single, prepared SQL statement. A prepared SQL statement is parsed and compiled only once, but you can execute it multiple times using different values for certain components of the SQL statement in each execution. A SQL statement that isn't prepared is parsed, compiled, and executed only once.

Prepared statements are useful only if the same type of SQL statement (`INSERT, UPDATE` , or `DELETE`) is executed repeatedly and consecutively with the same target table but with different values. If there's a sequence of consecutive changes that occur to the same table using the same operation such as inserting a set of rows into the same table populating the same columns, the publication server can apply these changes using a prepared statement. Otherwise, each change is applied with its own individual SQL statement.

A number of server configuration options control the characteristics of the JDBC batch along with if, when, and how often prepared statements are used.

`defaultBatchUpdateMode`

The `defaultBatchUpdateMode` option controls whether the default mode uses individual SQL statements in the JDBC batch update (referred to as `BUS` mode) or to use prepared SQL statements in the JDBC batch update (referred to as `BUP` mode).

`defaultBatchUpdateMode={BUS | BUP}`

The default value is `BUS` .

`switchBatchUpdateMode`

The `switchBatchUpdateMode` option controls whether the publication server dynamically switches between `BUS` mode and `BUP mode during the replication process. This switching depends on the type and sequence of updates it encounters in the shadow tables for the trigger-based method or the changeset stream for the log-based method.

`switchBatchUpdateMode={true | false}`

The default value is true.

Using the default settings of `defaultBatchUpdateMode=BUS` and `switchBatchUpdateMode=true`, the publication server starts out by applying updates with individual SQL statements. When it encounters a stream of consecutive changes that can all be processed in a single prepared statement, it switches to using prepared SQL statements.

> **Note**
>
> If you want a certain batch update mode used throughout all synchronization replications applied by a given publication server without switching update modes, set the `defaultBatchUpdateMode` option to the desired mode in combination with `switchBatchUpdateMode=false`. For example, if you want only prepared statements used, set the following options:

`defaultBatchUpdateMode=BUP`

`switchBatchUpdateMode=false`

> **Note**
>
> When Oracle is the subscription database, synchronization replication always occurs in `BUP` mode as if the preceding two options were always set. The reason for this is so that large columns of TEXT data type from Postgres publications can successfully replicate to Oracle CLOB columns. In `BUS` mode an individual Oracle SQL statement has a string literal maximum length of 4000 characters. This limitation doesn't occur for prepared SQL statements that are used in `BUP` mode.

`busBatchThresholdCount`

The `busBatchThresholdCount` option sets the number of consecutive updates of the same type that must be encountered in the shadow tables for the trigger-based method or the changeset stream for the log-based method before the publication server switches from `BUS` mode to `BUP` mode. This option applies if dynamic switching is permitted, that is `switchBatchUpdateMode=true`.

`busBatchThresholdCount=n`

The default value for n is 5.

The number of consecutive changes using the same table and SQL statement type must exceed the specified value n before a prepared statement is used.

Setting this threshold to a low value encourages higher use of prepared statements. Setting it to a high value limits the use of prepared statements.

If changes to the publication were made using many SQL statements, where each statement affected more than one row, then it might help to lower `busBatchThresholdCount`. Lowering this value encourages the use of prepared statements on the multiple shadow table rows resulting from each change on the publication.

`bupBatchThresholdCount and bupBatchThresholdRepeatLimit`

If you use `BUP` mode but the number of updates using the same prepared statement is low, this configuration can cause frequent switches to a new prepared statement. In this case, it might be more helpful to use individual SQL statements (`BUS` mode).

For example, the following sequence of updates are better processed in `BUS` mode:

```
INSERT INTO
emp
INSERT INTO dept
INSERT INTO
emp
INSERT INTO dept
DELETE FROM
emp
UPDATE
emp
UPDATE dept
INSERT INTO
emp
INSERT INTO dept
DELETE FROM dept
INSERT INTO
emp
DELETE FROM
emp
INSERT INTO dept
```

However, in the following sequence, it is better to use `BUP` mode. Updates 1 through 3 are batched in one prepared statement, 4 through 7 in another prepared statement, 8 in its own prepared statement, and then 9 through 15 in one prepared statement.

```
1. INSERT INTO
emp
2. INSERT INTO
emp
3. INSERT INTO
emp
4. UPDATE dept
5. UPDATE dept
6. UPDATE dept
7. UPDATE dept
8. INSERT INTO
emp
9. INSERT INTO dept
10. INSERT INTO dept
11. INSERT INTO dept
12. INSERT INTO dept
13. INSERT INTO dept
14. INSERT INTO dept
15. INSERT INTO dept
```

Use the `bupBatchThresholdCount` option with the `bupBatchThresholdRepeatLimit` option to control the frequency of mode switches based on the volatility of expected update types to the publication.

`bupBatchThresholdCount=m`

The default value for m is 5.

`bupBatchThresholdRepeatLimit=n`

The default value for n is 10.

Each time the same prepared SQL statement is consecutively executed, an internal `batch` counter is incremented. If this batch count falls below `bupBatchThresholdCount` for the number of executions of a given prepared statement, then a second internal `repeat` counter is incremented by one. If the repeat counter eventually reaches `bupBatchThresholdRepeatLimit`, the update mode is switched from `BUP` to `BUS`.

Thus, if there are frequent, consecutive changes of prepared SQL statements (as measured against `bupBatchThresholdRepeatLimit`), each of which is executed a small number of times (as measured against `bupBatchThresholdCount`), then the mode of execution changes back to individual SQL statements instead of prepared statements.

> **Note**
>
> The publication server changes back to prepared statements when the threshold set by `busBatchThresholdCount` is met.

The following example shows the processing of updates when `bupBatchThresholdCount` is set to 3 and `bupBatchThresholdRepeatLimit` is set to 4. A change to the "query domain" referred to in this example means a different statement type (`INSERT` , `UPDATE` , or `DELETE`) or a different target table are encountered in the next update. This condition requires the use of a different prepared SQL statement.

```
1.    INSERT INTO emp
2.    INSERT INTO emp
3.    INSERT INTO dept
```

At this point, the query domain is changed after the first two updates (change from table `emp` to `dept`), and the number of executions of the prior prepared statement (2) is less than `bupBatchThresholdCount` , so the repeat counter is set to 1.

```
4.    INSERT INTO dept
5.    INSERT INTO dept
6.    INSERT INTO dept
7.    INSERT INTO emp
```

The query domain is changed again (change from table `dept` to `emp`), but this time the number of executions (4) for the same query domain (updates 3 through 6) exceeds `bupBatchThresholdCount` . Thus the repeat counter is reset to 0.

```
8.    INSERT INTO
emp
9.    UPDATE
emp
```

The query domain is changed again (`INSERT` statement to `UPDATE` statement), and the number of executions (2) is less than `bupBatchThresholdCount` , so the repeat counter is incremented to 1.

```
10.   UPDATE
emp
11.   INSERT INTO dept
12.   DELETE FROM dept
13.   INSERT INTO
emp
```

The query domain is changed between updates 10 and 11, between updates 11 and 12, and between updates 12 and 13. At this point, the repeat counter was incremented three more times to a value of 4. This now equals `bupBatchThresholdRepeatLimit` , so processing is changed from `BUP` mode to `BUS` mode.

8.8.2.2 Parallel synchronization

Parallel synchronization takes advantage of multi-CPUs or cores in the system architecture by using multiple threads to apply transaction sets in parallel. Parallel synchronization is applied in two ways:

- Multiple threads are used to load data for multiple tables in parallel from the source database. Each thread opens a separate connection. Therefore, you see multiple connections with the source database. The pooling framework caches the connections. After the threads are finished with the data load, the idle connections are returned to the pool and remain there for a period of three minutes before being removed from the pool (as long as these aren't reused).
- Changes are applied to multiple target databases in parallel. A transaction set from the source database is loaded only once. The target databases are updated in parallel from this loaded transaction set. When this transaction set has been applied to all targets (either successfully or with failures on some targets), the next transaction set is loaded and applied in parallel. This aspect of parallel synchronization is particularly relevant to multi-master replication systems.

The following configuration options affect the use of parallel synchronization.

`syncLoadThreadLimit`

The `syncLoadThreadLimit` option controls the maximum number of threads used to load data from source publication tables during parallel synchronization. The default count is 4. However, depending on the target system architecture (specifically, multi-CPUs/cores), you can choose to specify a custom count, normally equal to the CPU/core count, to fully utilize the system resources.

`syncLoadThreadLimit=n`

The default value is 4.

`dataSyncThreadCount`

The `dataSyncThreadCount` option controls the maximum number of threads used to apply incremental changes during synchronization replication to the target secondary databases (for single-master replication systems) or to the target primary nodes (for multi-master replication systems) in parallel mode. The default behavior (when `dataSyncThreadCount` is set to 0) is to use as many threads as there are target nodes.

However, depending on the target system architecture (specifically, multi-CPUs/cores) you can choose to specify a custom count, normally equal to the CPU/core count, to fully utilize the system resources.

`dataSyncThreadCount=n`

The default value is 0.

`targetDBQueryTimeout`

The `targetDBQueryTimeout` option controls the timeout interval (in milliseconds) before an attempt by the publication server to apply a transaction set on a target database is aborted by the database server. This condition is due to a lock acquired by another application on one or more of the target tables).

The `targetDBQueryTimeout` option sets the default lock timeout value to 10 minutes. Change the 10-minute default value to a higher value if you want to allow a longer wait time before the transaction is aborted. Change the value to 0 if you want to turn off use of the `targetDBQueryTimeout` option. In this case, the timeout interval is controlled by setting the Postgres database server `statement_timeout` configuration parameter.

A higher value of `targetDBQueryTimeout` delays processing of subsequent transaction sets on other target databases. If a transaction set is blocked, the next transaction set can't load until either:

- The lock is released and the blocked transaction set can then be applied to completion.
- The targetDBQueryTimeout interval is exceeded.

If a timeout occurs, the waiting transaction set is marked as aborted for the particular blocked target database. The remaining pending transaction sets in this synchronization session are skipped for this target database but are applied to all other target databases once the timeout interval is exceeded. The aborted and skipped transaction sets are tried again when the next synchronization replication event occurs.

So, for example, in a three-node cluster with 10 pending transaction sets, assume transaction set 1 is loaded and begins replicating to nodes 2 and node 3. Now, another application acquires a lock on one or more tables in node 2, putting the updates to these tables in a wait state. Replication of transaction set 1 can run to completion on node 3. However, if the wait time exceeds the `targetDBQueryTimeout` interval, the database server cancels transaction set 1 on node 2. Replication of this transaction set to node 2 is marked as aborted in the Replication Server metadata.

Transaction set 2 can now be loaded and run against node 3. Executing transaction set 2 against node 2 is skipped since transaction sets must be applied in order and transaction set 1 was not successfully applied to node 2. Transaction sets 3 through 10 are loaded and applied in order against node 3 but skipped for node 2.

In the next synchronization replication, transaction set 2 is tried again on node 2. If the lock was released and the transaction set is applied successfully, the remaining transaction sets 3 through 10 are applied to node 2. Finally, synchronization replication continues with any new transaction sets.

`targetDBQueryTimeout=n`

The default value is 600000.

8.8.2.3 Other synchronization configuration options

The following are other configuration options affecting synchronization replication.

`syncBatchSize`

The `syncBatchSize` option controls the number of statements in a synchronization replication JDBC batch.

`syncBatchSize=n`

The default value for n is 100.

`syncFetchSize`

The `syncFetchSize` option controls how many rows are fetched from the publication database in one network round trip. For example, for 1000 pending row changes, the default fetch size requires five database round trips. Using a fetch size of 500 retrieves all changes in two round trips. Fine tune the performance by using a fetch size that conforms to the average data volume consumed by rows fetched in one round trip.

`syncFetchSize=n`

The default value for n is 200.

`txSetMaxSize`

The `txSetMaxSize` option defines the maximum number of transactional rows that you can group in a single transaction set. The publication server loads and processes the changes by fetching as many rows in memory as grouped in a single transaction set.

A higher value is expected to boost performance. However a very large value might result in an out-of-memory error. Increase or decrease the value in accordance with the average row size (low/high).

`txSetMaxSize=n`

The default value for n is 10000.

`enablePerformanceStats`

Set `enablePerformanceStats` to true only if you need to conduct performance testing and analyze the replication statistics. When enabled, the publication server creates more triggers on the publication tables in each primary node. The triggers produce transaction statistics that are recorded in the `MMR_transaction_history` table in the control schema. Disable this option in a production environment to avoid performance overhead.

`enablePerformanceStats={true | false}`

The default value is false.

9 Multi-master replication operation

You can configure and run Replication Server for multi-master replication systems.

The Replication Server console interface is used for steps and examples that show how multi-master replication operation. You can perform the same steps from the operating system command line using the Replication Server command line interface (CLI). See Replication Server command line interface.

9.1 Prerequisite steps

Before beginning the process of building a multi-master replication system, you must take certain steps to prepare the host environments and the database servers used as primary nodes.

Set heap memory size for the publication server

Replication speed and efficiency can be affected by the heap memory size set for the publication server. The Replication Server startup configuration file sets a parameter controlling the minimum and maximum heap size allocated for the publication server. See Setting heap memory size for the publication and subscription servers for guidelines and information on setting this parameter.

Enable synchronization replication with the log-based method

For Postgres database servers of version 9.4 and later: These settings apply if you plan to use the log-based method of synchronization replication with any primary node running under a given Postgres database server. The following configuration parameter settings are required in the configuration file `postgresql.conf` of each such Postgres database server:

- `wal_level` . Set to logical.
- `max_wal_senders` . Specifies the maximum number of concurrent connections (that is, the maximum number of simultaneously running WAL sender processes). Set, at minimum, to the number of MMR primary nodes on this database server that use the log-based method. In addition, if SMR publication databases runs on this database server, add the number of SMR publication databases that use the log-based method.
- `max_replication_slots` . Specifies the maximum number of replication slots. For support of MMR systems, the minimum is the total number of primary nodes in the multi-master replication system multiplied by the number of primary nodes residing on this database server. For information, see Replication origin. In addition, if SMR publication databases run on this database server, add the number of SMR publication databases that use the log-based method.
- `track_commit_timestamp` . Set to on. This configuration parameter applies only to Postgres database servers of version 9.5 or later. See Configuration parameter and table setting requirements for more information.

See Synchronization replication with the log-based method for information on the log-based method of synchronization replication.

Restart the Postgres database server after modifying any of these configuration parameters.

In addition, the `pg_hba.conf` file requires an entry for each publication database user of primary nodes that use the log-based method. You must include these database users as a replication database user in the `pg_hba.conf` file. See Verify host accessibility for more information.

Preparing the primary definition node

When creating the publication database definition for the primary definition node, specify a database user name that has the following characteristics:

- The database user can connect to the primary definition node.
- The database user has superuser privileges. Superuser privileges are required because the database configuration parameter `session_replication_role` is modified by the database user when the primary definition node receives updates from other primary nodes during a synchronization replication. The database user temporarily changes `session_replication_role` to `replica` to prevent the

triggers on the publication tables from activating. This session change also occurs for snapshot operations involving replication of the control schema from one publication database to another.
- The database user must be able to modify the system catalog tables. This ability is needed to disable foreign key constraints on the publication tables for snapshots targeted to the publication. It is also needed for the control schema tables for snapshot operations involving the replication of the control schema from one publication database to another. (See Disabling foreign key constraints for snapshot replications for more information on this requirement.)

The examples we use are based on the following primary definition node:

- The database user name for the primary definition node is `pubuser`.
- The tables used in the publication reside in a schema named `edb`.
- Three tables named dept, emp, and jobhist are members of schema `edb`.
- The database name of the primary definition node is edb.

These steps show how to prepare the primary definition node database user.

1. Create a user name with login and superuser privileges for the primary definition node. This user becomes the owner of Replication Server metadata database objects created in the primary definition node to track, control, and record the replication process and history. The Replication Server metadata database objects are created in a schema named `_edb_replicator_pub`.

   When creating the publication database definition, enter the database user name in the Publication Service – Add Database dialog box (see Adding the primary definition node).

   ```
   CREATE ROLE pubuser WITH LOGIN SUPERUSER PASSWORD 'password';
   ```

2. (Optional) If users need to access the data in the publication tables residing on this primary node, it's convenient to have one or more “group” roles containing the required privileges to access these tables. You must also grant privileges on the control schema objects to users who perform inserts, updates, or deletions on the publication tables.

   When adding users, granting these users membership in these roles gives them the privileges to access the publication tables. This setup eliminates the need to grant these privileges individually to each new user.

   See Step 2 of Postgres publication database for information on creating such roles.

Preparing more primary nodes

You can create a database user and a database for an additional primary node.

When creating the publication database definition for an additional primary node, specify a database user name that has the following characteristics:

- The database user can connect to the primary node.
- The database user has superuser privileges. Superuser privileges are required because the database user modfies the database configuration parameter `session_replication_role` when the primary node receives updates from other primary nodes during a synchronization replication. The database user temporarily changes `session_replication_role` to `replica` to prevent the triggers on the publication tables from activating. This session change also occurs for snapshot operations involving replication of the control schema from one publication database to another.
- The database user must be able to modify the system catalog tables. This abiity is needed to disable foreign key constraints on the publication tables for snapshots targeted to the publication. It is also needed for the control schema tables for snapshot operations involving the replication of the control schema from one publication database to another. See Disabling foreign key constraints for snapshot replications for more information on this requirement.
- If the added primary node resides on a different database server instance (that is, on a different host or port number) from the primary definition node, then use the same database user name for this additional primary node as used for the primary definition node. This isn't necessary if the publication server configuration option `skipTablePrivileges` is changed from its default value of false to true. See Skipping grants of table-level user privileges on MMR target tables for information on `skipTablePrivileges`.

Two possible options are available with respect to how to create the publication tables in the primary node:

- Allow the publication server to create the publication table definitions in the primary node by copying the definitions from the primary definition node when you add the publication database definition for the primary node.
- Define the publication tables in the primary node beforehand by running SQL `DDL` statements in the `PSQL` command line utility program or by using Postgres Enterprise Manager Client to create the tables.

If you create the table definitions manually, be sure the publication tables are defined identically to the tables in the primary definition node including schema names, table names, number of columns, column names, column data types, column lengths, primary key definitions, unique constraints, foreign key constraints, and so on.

The examples are based on the following:

- The database user name of the second primary node is `MMRuser`.
- The database name of the second primary node is `MMRnode`.

1. Create a database user name for the primary node. This user becomes the owner of Replication Server metadata database objects created in the primary node to track, control, and record the replication process and history. The Replication Server metadata database objects are created in a schema named `_edb_replicator_pub`.

   When creating the publication database definition for the primary node, enter the database user name in the Publication Service – Add Database dialog box. See Creating Additional Primary nodes.

   ```
   CREATE ROLE MMRuser WITH LOGIN SUPERUSER PASSWORD 'password';
   ```

2. Create a database to use as the primary node if this database doesn't already exist.

   ```
   CREATE DATABASE
   MMRnode;
   ```

Verifying host accessibility

If more than one computer is used to host the components of the replication system, each computer must be able to communicate with the others on a network.

- For a discussion of firewalls and access to ports see Firewalls and access to ports.
- For a discussion of network IP addresses see Network IP addresses.

A Postgres database server uses the host-based authentication file `pg_hba.conf` to control access to the databases in the database server. You need to modify the `pg_hba.conf` file on each Postgres database server that contains a primary node.

In a default Postgres installation, this file is located in the directory `POSTGRES_INSTALL_HOME/data`.

Primary nodes

On each database server running a primary node, you need the following to allow access to the database:

```
host <primarynode_db>   <primarynode_user>   <pub_ipaddr>/32   md5
```

The value you substitute for `<primarynode_db>` is the name of the database you intend to use as the primary node. The value you substitute for `<primarynode_user>` is the database user name you created in Step 1 of Preparing the primary definition node or Step 1 of Preparing more primary nodes.

For two primary nodes using databases named edb and MMRnode running on the same database server, the resulting `pg_hba.conf` file appears as follows:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                              md5
# IPv4 local connections:
host    edb         pubuser        192.168.2.22/32    md5
host    MMRnode     MMRuser        192.168.2.22/32    md5
host    all         all            127.0.0.1/32       md5
# IPv6 local connections:
host    all         all            ::1/128            md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                     md5
#host    replication  enterprisedb  127.0.0.1/32       md5
#host    replication  enterprisedb  ::1/128            md5
```

If the primary node using database MMRnode with database user name `MMRuser` is running on a separate host from where database edb is running, the `pg_hba.conf` file on the database server with database `MMRnode` looks like the following:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                              md5
# IPv4 local connections:
host    MMRnode     MMRuser        192.168.2.22/32    md5
host    all         all            127.0.0.1/32       md5
# IPv6 local connections:
host    all         all            ::1/128            md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                     md5
#host    replication  enterprisedb  127.0.0.1/32       md5
#host    replication  enterprisedb  ::1/128            md5
```

These examples assume databases edb and MMRnode are using the trigger-based method of synchronization replication. If you're using the log-based method, the `pg_hba.conf` file must contain additional entries with the `DATABASE` field set to replication for `primarynode_user` and `pub_ipaddr` to allow replication connections from the publication server on the host on which it is running.

The following shows changes to the first example with these added entries as the last two lines in the file:

```
# TYPE  DATABASE    USER           ADDRESS           METHOD

# "local" is for Unix domain socket connections only
local   all         all                              md5
# IPv4 local connections:
host    edb         pubuser        192.168.2.22/32    md5
host    MMRnode     MMRuser        192.168.2.22/32    md5
host    all         all            127.0.0.1/32       md5
# IPv6 local connections:
host    all         all            ::1/128            md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local   replication  enterprisedb                     md5
#host    replication  enterprisedb  127.0.0.1/32       md5
#host    replication  enterprisedb  ::1/128            md5
host    replication  pubuser       192.168.2.22/32    md5
host    replication  MMRuser       192.168.2.22/32    md5
```

See Synchronization replication with the log-based method and Enabling synchronization replication with the log-based method for more information on

synchronization replication with the log-based method.

Reload the configuration file after making the modifications. From the Postgres application menu, select **Reload Configuration** (**Expert Configuration > Reload Configuration** on EDB Postgres Advanced Server). Reloading puts the modified `pg_hba.conf` file into effect.

9.2 Creating a publication

Creating a publication requires:

- Registering the publication server
- Adding the primary definition node
- Creating a publication by choosing the tables for the publication along with the conflict resolution options
- Defining table filters to be enabled on any primary nodes

Registering a publication server

Register a publication server in a same way as for single-master replication. See Registering a publication server for details.

After you successfully register a publication server, the replication tree of the Replication Server console displays a Publication Server node under which are the SMR and MMR type nodes.

Continue to build the multi-master replication system under the MMR type node.

Adding the primary definition node

The first database to identify to Replication Server is the primary definition node. You do this by creating a publication database definition subordinate to the MMR type node under the Publication Server node.

After you create the publication database definition, a Publication Database node representing the primary definition node appears in the replication tree of the Replication Server console. You can then create a publication containing tables residing in this database under the Publication Database node.

You must enter database connection information such as the database server network address, database identifier, and database login user name and password when you create the publication database definition. The connection information is used by the publication server to access the publication tables when it performs replication.

1. Make sure the database server for the primary definition node is running and accepting client connections.

2. Select the MMR type node under the Publication Server node. Select **Publication > Publication Database > Add Database**.

3. In the Publication Service – Add Database dialog box, fill in the following fields:

    - **Database Type**. Select **PostgreSQL** or **Postgres Plus Advanced Server** for the primary definition node. For an EDB Postgres Advanced Server Oracle-compatible installation, select the **Postgres Plus Advanced Server** option. For PostgreSQL or an Advanced Server PostgreSQL-compatible installation, select the **PostgreSQL** option.
    - **Host.** IP address of the host on which the primary definition node is running.
    - **Port.** Port on which the primary definition node is listening for connections.
    - **User.** The database user name for the primary definition node created in Step 1 of Preparing the primary definition node.
    - **Password.** Password of the database user.
    - **Database.** The database name of the primary definition node.
    - **URL Options (For SSL connectivity)**. Enter the URL options to establish SSL connectivity to the primary definition node. See Using secure sockets layer (SSL) connections for more information.

- **Changeset Logging (For Postgres)**. Select **Table Triggers** to use the trigger-based method of synchronization replication. Select **WAL Stream** to use the log-based method of synchronization replication. See Synchronization replication with the trigger-based method and Synchronization replication with the log-based method for information on the log-based method.
- **Node Priority Level.** Enter an integer from 1 to 10, which is the priority level assigned to this primary node for conflict resolution based on node priority. The highest priority is 1 while the lowest is 10. See Conflict resolution strategies for more information. The default is 1 for the primary definition node.

4. Select **Test**. When Test Result: Success appears, select **OK**, and then select **Save**.

   When the publication database definition is successfully saved, a Publication Database node is added to the replication tree under the MMR type node of the Publication Server node.

   The label **MDN** appears at the end of the node in the replication tree. In addition, the **MDN** field is set to **Yes** in the Property window to indicate this is the primary definition node.

Adding a publication

Subordinate to the primary definition node, you create a publication that contains tables of the database.

1. Select the Publication Database node. Select **Publication > Create Publication**.

2. In the Create Publication dialog box, on the **Create Publication** tab, fill in the following fields:

   - **Publication Name.** Enter a name that is unique among all publications.
   - **Publish.** Select the boxes next to the tables to include in the publication. You can optionally select **Use Wildcard Selection** to use wildcard pattern matching for selecting publication tables.
   - **Select All.** Select this option to include all tables in the Available Tables list in the publication.
   - **Use Wildcard Selection.** Select to use the wildcard selector to choose tables for the publication. See Selecting tables with the wildcard selector for more information.

3. (Optional) Table filters consist of a set of filter rules that control the selection criteria for rows replicated between primary nodes during a snapshot or a synchronization replication.

   > **Note**
   >
   > See Table settings and restrictions for table filters for table setup requirements for a log-based replication system as well as general restrictions on the use of table filters.

   A filter rule consists of a filter name and a SQL `WHERE` clause (omitting the `WHERE` keyword) called the filter clause, which you specify for a table that defines the selection criteria for rows to include during a replication.

   You can define multiple filter rules for each table in the publication. If you don't define a filter rule for a given table, then you cannot enable filtering later on that corresponding table in any primary node of the multi-master replication system.

   After you define the filter rules for a publication table, you can later choose whether to enable those filter rules on any primary node in the replication system in accordance with the following rules:

   - You can enable at most one filter rule on a given table in a given primary node.
   - You can enable the same filter rule on the same given table in several different primary nodes.
   - You can enable different filter rules on the same given table but in different primary nodes.

   If you want to define table filters on the publication tables, select the **Table Filters** tab. Select the table from the **Table/View** list for which you want to add a filter rule. Select **Add Filter**.

   In the Filter dialog box, enter a descriptive filter name and the filter clause to select the rows you want to replicate. The filter name and filter clause must meet the following conditions:

- For any given table, each filter rule must be assigned a unique filter name.
- For any given table, the filter clauses must have different syntaxes (that is, the filtering criteria must be different).

To remove a filter rule, select the filter rule you want to remove, and then select **Remove Filter**.

You can also modify the filter name or filter clause of a filter rule listed in the **Table Filters** tab. Double-click the cell of the filter name or filter clause you want to change and enter the text for your change.

When creating more primary nodes, you can selectively enable these table filters on the corresponding tables in the added primary nodes. See Creating more primary nodes for more information.

> **Note**
>
> To enable table filters on the primary definition node under which you are currently creating the publication, you must first switch the role of the primary definition node to a different primary node (see Switching the primary definition node) and then follow the directions in Enabling/disabling table filters on a primary node to enable the table filters.

4. (Optional) If you want to modify or see the current conflict resolution options, select the **Conflict Resolution Options** tab. For each table, you can select the primary conflict resolution strategy and a standby strategy by selecting from a list of options.

   If, during synchronization replication, conflicting changes are pending against the same row from different primary nodes, the conflict resolution strategy determines which of the conflicting changes is accepted and replicated to all primary nodes. The conflicting changes that aren't accepted are discarded.

   If the selection from the Conflict Resolution Strategy column doesn't resolve the conflict, the selection from the Standby Conflict Resolution Strategy column is applied. If neither strategy resolves the conflict, the event is marked as Pending in the **Conflict History** tab. See Viewing conflict history for more information.

   An example of a conflict is when the same column of the same row is changed by transactions in two different primary nodes. Depending on the conflict resolution strategy in effect for the table, one of the transactions is accepted and replicated to all primary nodes. The other transaction is discarded and not replicated to any primary node.

   The following is a summary of each conflict-resolution strategy:

   - **Earliest Timestamp.** The conflicting change with the earliest timestamp is accepted and replicated to all other primary nodes. All other conflicting changes are discarded.
   - **Latest Timestamp.** The conflicting change with the latest timestamp is accepted and replicated to all other primary nodes. All other conflicting changes are discarded.
   - **Node Priority.** The conflicting change occurring on the primary node with the highest priority level is accepted and replicated to all other primary nodes. All other conflicting changes are discarded.
   - **Custom.** Update/update conflicts are resolved with a PL/pgSQL custom conflict-handling program.
   - **Manual.** The conflict remains unresolved. Conflicting changes remain applied in each primary node where they originated but aren't replicated to other primary nodes. You must apply the proper adjustments in each primary node.

   See Conflict resolution strategies for more information.

5. If you expect update/update conflicts, then set the `REPLICA IDENTITY` option to `FULL` on those tables where you expect the conflicts to occur. See Configuration parameter and table setting requirements for more information.

6. Select **Create**. When Publication Created Successfully appears, select **OK**.

   Upon successful publication creation, a Publication node is added to the replication tree.

9.3 Creating more primary nodes

Once you have created the primary definition node, you add more databases to the multi-master replication system by defining more primary nodes.

You co this by creating more publication database definitions subordinate to the MMR type node under the Publication Server node that contains the primary definition node.

After you create the publication database definition, a Publication Database node representing the primary node appears in the replication tree of the Replication Server console. The publication that was defined under the primary definition node appears under the Publication Database node.

You must enter database connection information such as the database server network address, database identifier, and database login user name and password when you create the publication database definition. The publication server uses the connection information to access the publication tables when it performs replication.

1. Make sure the database server for the primary definition node is running and accepting client connections.

2. Select the MMR type node under the same Publication Server node that contains the primary definition node. Select **Publication > Publication Database > Add Database**.

3. In the Publication Service – Add Database dialog box, fill in the following fields:

    - **Database Type.** Select **PostgreSQL** or **Postgres Plus Advanced Server** for the primary node. For an EDB Postgres Advanced Server Oracle-compatible installation, select the **Postgres Plus Advanced Server** option. For PostgreSQL or an EDB Postgres Advanced Server PostgreSQL-compatible installation, select the **PostgreSQL** option.
    - **Host.** IP address of the host on which the primary node is running.
    - **Port.** Port on which the primary node is listening for connections.
    - **User.** The database user name for the primary node created in Step 1 of Preparing more primary nodes.
    - **Password.** Password of the database user.
    - **Database.** The database name of the primary node.
    - **URL Options (For SSL connectivity).** Enter the URL options to establish SSL connectivity to the primary node. See Using secure sockets layer (SSL) Connections for more information.
    - **Changeset Logging (for Postgres).** This setting is baed on the selection on the primary definition node (see Adding the primary definition node). **Table Triggers** is for the trigger-based method of synchronization replication. **WAL Stream** is for the log-based method of synchronization replication. See Synchronization replication with the trigger-based method and Synchronization replication with the log-based method for more information.
    - **Node Priority Level.** Enter an integer from 1 to 10, which is the priority level assigned to this primary node for conflict resolution based on node priority. The highest priority is 1 and the lowest is 10. See Conflict resolution strategies for more information. As you add each primary node, the default priority level number increases, assigning a lower priority level to each additional node.
    - **Replicate Publication Schema.** Select this option if you want the publication server to create the publication table definitions in the new primary node by copying the definitions from the primary definition node. If you don't select this option, it's assumed that you already created the table definitions in the primary node. If you're using the offline snapshot technique to create this primary node, don't select this option. See Loading tables from an external data source (offline snapshot) for information on using an offline snapshot.
    - **Perform Initial Snapshot.** Select this option if you want the publication server to perform a snapshot from the primary definition node to this primary node when you select **Save**. If you don't select this option, the tables on the primary node aren't loaded until you perform a replication later. If you're using the offline snapshot technique to create this primary node, you already loaded the table rows. Therefore, don't select this option unless you want to reload the data. See Loading tables from an external data source (offline snapshot) for information on using an offline snapshot.

    > **Note**
    >
    > Unless you intend to use the offline snapshot technique (see Loading tables from an external data source (offline snapshot), we suggest that you select **Perform Initial Snapshot**. You must perform an initial snapshot replication from the primary definition node to every other primary node before performing synchronization replications on demand or by a schedule. (See Performing synchronization replication and Creating a schedule). If a newly added primary node didn't undergo an initial snapshot, any subsequent synchronization replication might fail to apply the transactions to that primary node. You can also take the initial snapshot by performing an on-demand snapshot. See Performing snapshot replication.

4. Select **Test**. When Test Result: Success appears, select **OK**.

5. (Optional) If you defined a set of available table filters for the publication, you can enable these filters on this primary node. See Adding a publication for instructions on defining table filters.

    > **Note**
    >
    > See Table settings and restrictions for table filters for table setup requirements for a log-based replication system and general restrictions on using table filters.

Select the **Filter Rules** tab to apply one or more filter rules to the primary node. At most you can enable one filter rule on any table in the primary node.

6. Select **Perform Initial Snapshot** if you want the publication server to perform a snapshot from the primary definition node to this primary node when you select **Save**. If you don't select this option, the tables on the primary node aren't loaded until you perform a replication later.

   If you're using the offline snapshot technique to create this primary node, you already loaded the table rows. Therefore don't select this option unless you want to reload the data. See Loading tables from an external data source (offline snapshot) for information on using an offline snapshot.

   If you select *Perform Initial Snapshot**, the **Verbose Output** option appears.

   If you skipped the enabling of table filters as described in Step 5 and you selected **Perform Initial Snapshot** after Step 4, the **Verbose Output** option appears. Select **Verbose Output** if you want to display the output from the snapshot in the dialog box. Don't select this option in a network address translation (NAT) environment as a large amount of output from the snapshot can delay the response from the Snapshot dialog box.

   Select **Save**.

   When the publication database definition is successfully saved, a Publication Database node is added to the replication tree under the MMR type node of the Publication Server node.

   Unlike the primary definition node, the label MDN doesn't appear at the end of the node in the replication tree. The**MDN** field is set to **No** in the Property window to indicate this is not the primary definition node.

   In addition, a Publication node appears under the newly added primary node. This Publication node represents the publication in the primary definition node, which is replicated to the primary node.

   If, in Step 6, you select the**Perform Initial Snapshot**, an initial snapshot replication is performed. If you selected **Verbose Output**, the log of the snapshot is displayed as well.

   If the snapshot is successful, the replicated tables in the primary node are loaded with the rows from the publication tables of the primary definition node.

7. If you expect update/update conflicts, then set the `REPLICA IDENTITY` option to `FULL` on those tables where you expect the conflicts to occur. See Configuration parameter and table setting requirements for more information.

8. (Optional) If users need to access the data in the publication tables residing on this primary node, it's convenient to have one or more group roles containing the required privileges to access these tables. For the trigger-based method, you must also grant privileges on the control schema objects to users who perform inserts, updates, or deletions on the publication tables. When using the log-based method, a user needs access to the publication tables and to certain control schema objects under certain circumstances.

   When adding new users, granting these users membership in these roles gives them the privileges to access the publication tables. This approach eliminates the need to grant these privileges individually to each new user.

   After you perform the replication of the publication schema as shown in Step 3, you can grant the required privileges needed to access the publication tables and its control schema objects. See Step 2 of Postgres publication database for information on how to do this.

9.4 Control schema objects created in primary nodes

Creating primary nodes creates control schema objects in each primary node database.

See Control schema objects created for a publication for the control schema objects created in each primary node.

Don't delete any of these control schema objects as the replication system metadata will become corrupted.

When you remove a primary node using the Replication Server console or CLI, all of its control schema objects are deleted from that primary node database.

9.5 On-demand replication

After you create a primary definition node, its publication, and additional primary nodes, you have two choices for starting the replication process:

- Perform replication immediately by performing an initial on-demand snapshot. After you perform the on-demand snapshot, you can perform synchronization replication.
- Schedule replication to start later by creating a schedule. See Creating a schedule.

Perform snapshot replication

A snapshot replication occurs from the primary definition node to a selected primary node.

When you create a primary node for the first time, you can perform an initial snapshot (see Step 3 of Creating additional primary nodes). You can also perform snapshots to this primary node later.

1. In the Replication Server cojsole, select the Publication node under the primary node for which you want to perform snapshot replication.
2. Select the **Snapshot** icon.
3. In the Snapshot dialog box, select **Verbose Output** if you want to display the output from the snapshot in the dialog box. Don't select this option in a network address translation (NAT) environment as a large amount of output from the snapshot can delay the response from the Snapshot dialog box.
4. To start snapshot replication, select **Snapshot**. Snapshot Taken Successfully appears when the snapshot is complete.
5. Select **OK**.

The status messages of each snapshot are saved in the Migration Toolkit log files named `mtk.log[.n]` in the following directories. (`[.n]` is an optional history file count if log file rotation is enabled.)

For Linux:

`/var/log/edb/xdb/`

For Windows:

`POSTGRES_HOME\.enterprisedb\xdb\x.x`

`POSTGRES_HOME` is the home directory of the Windows postgres account (enterprisedb account for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode). The specific location of `POSTGRES_HOME` depends on your version of Windows. The Replication Server version number is represented by `x.x`.

The publication is now replicated from the primary definition node to the selected primary node. A record of the snapshot is saved in the replication history. See Viewing replication history for more information.

> **Note**
>
> Before version 7.0.0, Replication Server required superuser privileges to disable/enable constraints and indexes as part of its snapshot operation. Starting with version 7.0.0, the superuser privileges are no longer required, and the constraints/indexes are now dropped and re-created as part of the snapshot operation. This change might cause any views/materialized views with a dependency on the constraints from the target subscription/master database to be dropped. These views are then re-created at the end of snapshot operation. Depending on the permissions assigned to the Replication Server subscription database user, the re-created views/materialized views might have permissions that are inconsistent with those of the originally assigned permissions. Therefore, we recommend manually verifying the assigned permissions of the relevant views/materialized views in the target database and making any corrections where applicable.

Perform synchronization replication

> **Note**
>
> Be sure an initial snapshot replication was performed from the primary definition node to every other primary node in the multi-master replication system. If a newly added primary node didn't undergo an initial snapshot, any later synchronization replication might not apply the transactions to that primary node. You can take the initial snapshot when the primary node is first added (see Creating additional primary nodes) or by performing an on-demand snapshot (see Performing snapshot replication).

When you perform synchronization replication in a multi-master replication system, a series of synchronization operations occur between every primary node pair in the replication system.

For example, if a replication system consists of primary nodes A, B, and C, synchronization is applied to the following node pairs whenever synchronization replication is initiated:

- Changes on node A are applied to node B.
- Changes on node A are applied to node C.
- Changes on node B are applied to node A.
- Changes on node B are applied to node C.
- Changes on node C are applied to node A.
- Changes on node C are applied to node B.

Changes made on different nodes might result in conflicts.Conflict resolution discusses the types of conflicts that can occur and how to resolve them.

To start an on-demand synchronization replication:

1. In the Replication Server console, select the Publication node under any primary node. Whatever primary node you choose, synchronization is applied to every primary node pair in the replication system.

2. Select the **Synchronize** icon.

3. In the Synchronize dialog box, to start synchronization replication, select **Synchronize**. When synchronization is complete, Publication Synchronized Successfully appears.

4. Select **OK**.

You can now see the operations that were applied to the publication tables in the replication history. See Viewing replication history for more information.

You can see conflicting changes that occurred in the conflict history. See Viewing conflict history for more information.

9.6 Conflict resolution

In certain situations, synchronization replication can result in data conflicts arising from the row changes that took place on different primary nodes.

Conflict resolution concerns the types of conflicts that might occur, the strategies for dealing with conflicts, and the options available for automatically resolving such conflicts.

9.6.1 Configuration parameter and table setting requirements

Depending on the multi-master replication system environment, you might need certain configuration settings for the conflict resolution process to operate properly.

The following are required only for the log-based method. These do not apply to the trigger-based method.

- `track_commit_timestamp`. Any Postgres 10 and later database server containing a primary node must have its `track_commit_timestamp` configuration parameter enabled. The `track_commit_timestamp` parameter is located in the `postgresql.conf` file. If `track_commit_timestamp` isn't enabled, then update/update conflicts aren't automatically resolved such as by using the earliest timestamp of the conflicting transactions. As a result, these conflicting transactions are left in a pending state. See Automatic conflict resolution example for an example of how update/update conflicts are automatically resolved.
- `REPLICA IDENTITY FULL`. If `update/update conflicts` are expected to occur on a given publication table, then you must set the `REPLICA IDENTITY` setting for the table to `FULL` on every primary node. The case where update transactions occur on separate primary nodes but updating different columns in the same row isn't considered an update/update conflict. However, if `REPLICA IDENTITY` isn't set to `FULL`, then this case is recorded as an update/update conflict.

Set the `REPLICA IDENTITY` option to `FULL` using the `ALTER TABLE` command as shown by the following:

```
ALTER TABLE schema.table_name REPLICA IDENTITY
FULL
```

The following is an example of the `ALTER TABLE` command:

```
ALTER TABLE edb.dept REPLICA IDENTITY
FULL;
```

You can display the `REPLICA IDENTITY` setting with the `PSQL` utility using the `\d+` command:

```
edb=# \d+
edb.dept
```

```
                                Table "edb.dept"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 deptno | numeric(2,0)          | not null  | main     |              |
 dname  | character varying(14) |           | extended |              |
 loc    | character varying(13) |           | extended |              |
Indexes:
    "dept_pk" PRIMARY KEY, btree (deptno)
    "dept_dname_uq" UNIQUE CONSTRAINT, btree (dname)
Referenced by:
    TABLE "emp" CONSTRAINT "emp_ref_dept_fk" FOREIGN KEY (deptno) REFERENCES dept(deptno)
    TABLE "jobhist" CONSTRAINT "jobhist_ref_dept_fk" FOREIGN KEY (deptno) REFERENCES dept(deptno) ON DELETE
SET NULL
Replica Identity: FULL
```

> **Note**
>
> In addition to conflict resolution requirements, you might need the `REPLICA IDENTITY FULL` setting on publication tables for other reasons in Replication Server. See Table settings and restrictions for table filters for additional requirements.

9.6.2 Conflict types

The types of conflicts can be summarized as follows:

- **Uniqueness conflict.** A uniqueness conflict occurs when the same value is used for a primary key or unique column in an insert transaction on two or more primary nodes. This conflict is also referred to as an insert/insert conflict.
- **Update conflict.** An update transaction modifies a column value in the same row on two or more primary nodes. For example, an employee address column is updated on primary node A, and another user updates the address column for the same employee on primary node B. The timestamps of when the transactions occur on each node might differ, but both transactions occur in a time interval during which synchronization hasn't yet occurred. Thus when synchronization does take place, both conflicting transactions are applied. This conflict is also referred to as an update/update

conflict.

- **Delete conflict.** The row corresponding to an update transaction on the source node isn't found on the target node as the row was already deleted on the target node. This case is referred to as an update/delete conflict. Conversely, if there's a delete transaction on the source node and an update transaction for the same row on the target node, this case is referred to as a delete/update conflict. The case when the row corresponding to a delete transaction on the source node isn't found on the target node as the row was already deleted on the target node is referred to as a delete/delete conflict.

The following table definition shows conflict resolution examples:

```
CREATE TABLE addrbook
(
    id              SERIAL PRIMARY
KEY,
    name            VARCHAR(20),
    address         VARCHAR(50)
);
```

The following table shows an example of a uniqueness conflict.

Timestamp	Action		
t1	**Node A:** INSERT INTO addrbook (name, address) VALUES ('A', 'ADDR A'); **Node A:** INSERT INTO addrbook (name, address) VALUES ('B', 'ADDR B');	Primary node A id = 1, name = 'A', address = 'ADDR A' id = 2, name = 'B', address = 'ADDR B'	Primary node B
t2	**Node B:** INSERT INTO addrbook (name, address) VALUES ('C', 'ADDR C');	id = 1, name = 'A', address = 'ADDR A' id = 2, name = 'B', address = 'ADDR B'	id = 1, name = 'C', address = 'ADDR C'
t3	Synchronization pushes Node A changes to Node B		Row change for INSERT tx id = 1 on Node A results in unique key conflict on Node B id = 1, name = 'C', address = 'ADDR C' id = 1, name = 'A', address = 'ADDR A'

The following table shows an example of an update conflict.

Timestamp	Action		
`t0'			
t1	**Node A:** UPDATE addrbook SET address = 'ADDR B1' WHERE id = 2;	id = 2, address = **'ADDR B1'**	id = 2, address = 'ADDR B'
t2	**Node B:** UPDATE addrbook SET address = 'ADDR B2' WHERE id = 2; Synchronization pushes Node A changes to Node B	id = 2, address = 'ADDR B1'	id = 2, address = **'ADDR B2'**
t3	Synchronization pushes Node A changes to Node B		**Current value of address on Node B not equal old value on Node A ('ADDR B2' <> 'ADDR B')**

The following table shows an example of a delete conflict.

Timestamp	Action

Timestamp	Action		
		Primary node A	Primary node B
`t0`		id = 2, address = 'ADDR B'	id = 2, address = 'ADDR B'
`t1`	**Node A:** UPDATE addrbook SET address = 'ADDR B1' WHERE id = 2;	id = 2, address = **'ADDR B1'**	id = 2, address = 'ADDR B'
`t2`	**Node B:** DELETE FROM addrbook WHERE id = 2;	id = 2, address = 'ADDR B1'	**Row with id = 2 deleted**
`t3`	Synchronization pushes Node A changes to Node B		**The row with id = 2 is already deleted on target Node B, hence update from Node A fails.**

9.6.3 Conflict detection

When synchronization replication occurs, either on demand or on a scheduled basis, each of the primary node changes is pushed to the other primary nodes. See Multi-master parallel replication for information on this process.

Using a three-node example, the following describes the conflict detection process.

- The replication server loads the first set of pending transactions from primary node A. Transactions are processed on a transaction-set basis. (The same process is used for single-master replication.) All pending transactions are grouped in one or more transaction sets to avoid loading a very large chunk of rows in memory that might result in an out-of-heap-space issue.
- For an update transaction, the replication server queries the first target primary node B to load the related row. If the old column value on the source primary node A is different from the current column value on target primary node B, the transaction is marked as an update/update conflict. If a related row isn't found on the target primary node, it's marked as an update/delete conflict.
- For a delete transaction, the replication server queries the target primary node to load the related row. If a related row isn't found on the target primary node, the transaction is marked as a delete/delete conflict.
- When a conflict is detected, the conflict information such as the transaction ID, conflict type, and conflict detection timestamp are logged in the conflict table on the target primary node.
- For a conflicting transaction, the replication server checks if any conflict resolution strategy was selected for the specific table. If a strategy is found, it's applied accordingly and the conflict status is marked as resolved. If a strategy can't be applied, the conflict status is marked as unresolved (also called pending).
- If no conflict is detected, the transactional change is replicated to the target primary node, and the transaction status for that target node is marked as completed in the source primary node control schema. A transaction status mapping for each target primary node is maintained on all primary nodes. For example, node A contains two mappings of status: one for node B and another for node C.
- All of these prior steps are repeated to process and replicate all pending transaction sets available on primary node A to primary node B.
- Next, the publication server proceeds to replicate primary node A pending transactional changes to the next target primary node, C.
- Once the primary node A changes are replicated to nodes B and C, the publication server replicates the pending changes available on primary node B to nodes A and C.
- The primary node C changes are replicated to nodes A and B.

9.6.4 Conflict-resolution strategies

Several built-in conflict-resolution options are available to support automatic conflict resolution. The conflict-resolution options apply to update/update and delete/delete conflicts.

Uniqueness (insert/insert), update/delete, and delete/update conflicts are marked unresolved. You must reconcile them manually.

The following are the built-in, automatic conflict resolution options.

- **Earliest timestamp.** When you select the earliest-timestamp option, the relevant rows involved in an update conflict from the source and target primary nodes are compared based on the timestamp of when the update occurred on that particular node. The row change that occurred earliest is applied. The row changes with the later timestamps are discarded.
- **Latest timestamp.** This option takes the same approach as earliest timestamp except the row change with the latest timestamp is accepted. The row

changes with earlier timestamps are discarded.
- **Node priority.** The row change of the primary node with the highest node priority level is applied while the lower-priority-level primary node changes are discarded. The node priority level is an integer in the range of 1 to 10, where 1 is the highest priority level and 10 is the lowest priority level.
- **Custom.** Custom conflict handling applies to update/update conflicts only. You must supply a PL/pgSQL program to resolve any conflicts that occur resulting from an update/update conflict. See Custom conflict handling for information on using custom conflict handling.

The delete/delete conflict is always resolved implicitly regardless of the conflict resolution option in effect. The net impact of a delete/delete conflict is the removal of a given row, and the row in question was already removed from the source and target nodes.

For the earliest timestamp and latest timestamp conflict resolution strategies, the transaction timestamp is tracked in a column with data type `TIMESTAMP` in the shadow table.

Once you select it, you can later change the conflict resolution strategy for a given table to a different strategy. See Updating the conflict-resolution options).

9.6.5 Conflict prevention – uniqueness case

Since there's no automatic built-in resolution strategy for the uniqueness conflict, you can use strategies to avoid this problem. These strategies are implemented by the DBA. This discussion is based on a realm of numeric values generated by a sequence such as for a unique primary key.

The following are possible strategies:

- **Node-specific sequence range**. A sequence range is reserved for each primary node. For example, primary node A has `MINVALUE = 1` and `MAXVALUE = 1000`, primary node B has `MINVALUE = 1001` and `MAXVALUE = 2000`, and so on for other nodes. This mechanism ensures that a unique ID is always generated across all primary nodes.
- **Start-value variation**. Each node is assigned a different start value. For example, primary node A has a `START` value of `1`, node B has a value of `2`, and node C has a value of `3`. An increment greater than or equal to the number of nodes guarantees unique IDs, as shown in the table that follows.
- **Common sequence**. All nodes share a common sequence object. However this approach has the major disadvantage of slowing down transaction processing due to network round trips associated with each ID generation.
- **MMR-ready sequence**. This technique enhances the use of sequences and provides a more flexible, reliable approach for a distributed, multiple database architecture as is inherent in a multi-master replication system. This approach is recommended over the other sequence techniques. See Conflict prevention with an MMR-ready sequence for more information.

Sequence Clause	Primary node A	Primary node B	Primary node C
START WITH	1	2	3
INCREMENT BY	5	5	5
Generated IDs	1, 6, 11, 16, ...	2, 7, 12, 17, ...	3, 8, 13, 18, ...

9.6.6 Conflict prevention with an MMR-ready sequence

To prevent uniqueness conflicts in a multi-master replication system, you can use an MMR-ready sequence to generate unique identifiers for each row of publication tables that doesn't have an inherent, unique identifier.

An MMR-ready sequence incorporates a function and a sequence to return `BIGINT` data type, integer values. These values combine a user-assigned, unique database identifier for each primary node with a sequence generated within that primary node.

You can modify a publication table requiring an MMR-ready sequence to include a `BIGINT` NOT NULL column with a default value returned by the function.

An MMR-ready sequence satisfies the following characteristics:

- **Uniqueness.** The combination of the unique database identifier with the sequence ensures that each row in a given table has a unique value across all primary nodes.
- **Clustered index support.** An MMR-ready sequence doesn't impair the use of a clustered index to provide retrieval efficiency. MMR-ready sequence values are returned in a typical, ordered sequence, not as random values such as if the universally unique identifier (UUID) were used.
- **Effective migration support.** Yuo can modify tables already using a sequence to use an MMR-ready sequence with minimal impact on existing primary keys and foreign keys.
- **Reliability and maintainability.** An MMR-ready sequence provides a reliable and maintainable method to avoid uniqueness conflicts.

9.6.6.1 Creating an MMR-ready sequence

The following are the steps to create an MMR-ready sequence in a database to participate as a primary node in a multi-master replication system.

Begin these steps with the database to use as the primary definition node.

1. Assign a unique, database identifier as an integer from 1 to 1024, inclusive. You can uniquely identify a maximum of 1024 databases in a multi-master replication system with an MMR-ready sequence.

   Issue the following commands to create and set the database identifier:

   ```
   ALTER DATABASE dbname SET cluster.unique_db_id TO
   db_id;
   SET cluster.unique_db_id TO <db_id>;
   ```

   Use a different `<db_id>` value for each database.

2. Create a sequence to uniquely identify each table row in the database.

   ```
   CREATE SEQUENCE seq_name START WITH 1 INCREMENT BY 1 NO
   CYCLE;
   ```

   You can create multiple sequences if you want to use separate sequences for multiple tables in the publication. Be sure to use the same sequence name across all databases for the same given table.

   A publication table column that uses an MMR-ready sequence includes a `DEFAULT` clause referencing the sequence name in a function call. The publication table definition must be consistent across all primary nodes by referencing the same sequence name in the function call.

3. Create the following function that returns the next MMR-ready sequence value when a row is inserted into the table. This function is referenced by the `DEFAULT` clause of the publication table column.

   ```
   CREATE OR REPLACE FUNCTION MMR_sequence_nextval
   (
       seq_id
   VARCHAR
   )
   RETURNS
   bigint
   LANGUAGE
   sql
   AS
   $function$
   SELECT
   (
    (SELECT
   current_setting('cluster.unique_db_id'))::bigint
     << 52)::bigint +
     nextval($1::regclass);
   $function$;
   ```

The sequence name created in Step 2 is specified as the `seq_id` input argument when the function is added to the `DEFAULT` clause of the publication table column.

This function performs a bitwise shift left operation (`<< 52`) on the database identifier (`cluster.unique_db_id`), thus significantly increasing its numeric value. The next sequence value is then added to this number. Thus, all rows inserted in the table on a given database fall in a numeric range determined by the shifted, database identifier value.

4. (Optional) Create the following function to obtain the current MMR-ready sequence value.

```
CREATE OR REPLACE FUNCTION MMR_sequence_currval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  currval($1::regclass);
$function$;
```

Invoke the `MMR_sequence_nextval` function in the current session before calling the `MMR_sequence_currval` function.

5. Add or modify the publication table column that uses the MMR-ready sequence. The column data type must be `BIGINT` . The `MMR_sequence_nextval` function is specified in the `DEFAULT` clause as shown in the following example for column `id` .

```
CREATE TABLE table_name
(
    id          BIGINT NOT NULL PRIMARY
KEY
                DEFAULT
MMR_sequence_nextval('seq_name'),
    field       VARCHAR2(20)
);
```

The column also typically is the primary key.

6. Repeat steps 1 through 4 for the other databases to add as primary nodes.

> **Note**
>
> Omit Step 5 for the additional primary nodes. Publication table definitions are replicated from the primary definition node to the additional primary nodes when they are created. See Creating additional primary nodes.

7. Create the complete, multi-master replication system as described in Multi-master replication operation.

9.6.6.2 MMR-ready sequence example

The following is an example of a three-primary node system using an MMR-ready sequence. The databases to use as the primary nodes are `MMRnode_a` , `MMRnode_b` , and `MMRnode_c` . A publication table named `MMR_seq_tbl` uses the `MMR-ready` sequence.

The following commands are invoked in database `MMRnode_a`, which is the primary definition node:

```
 ALTER DATABASE MMRnode_a SET cluster.unique_db_id TO 1;
SET cluster.unique_db_id TO 1;

CREATE SEQUENCE MMR_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;

CREATE OR REPLACE FUNCTION MMR_sequence_currval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  currval($1::regclass);
$function$;

CREATE TABLE MMR_seq_tbl
(
    id          BIGINT NOT NULL PRIMARY
KEY
                DEFAULT
MMR_sequence_nextval('MMR_seq'),
    field       VARCHAR2(20)
);
```

On `MMRnode_b` and `MMRnode_c`, run the commands to create different settings for the configuration parameter `cluster.unique_db_id` are run and the commands to create the sequence and the functions.

On `MMRnode_b`, invoke the following commands.

> **Note**
>
> `Cluster.unique_db_id` is set to `2`.

```
 ALTER DATABASE MMRnode_b SET cluster.unique_db_id TO 2;
SET cluster.unique_db_id TO 2;

CREATE SEQUENCE MMR_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;

CREATE OR REPLACE FUNCTION MMR_sequence_currval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  currval($1::regclass);
$function$;
```

On `MMRnode_c`, invoke the following commands.

> **Note**
>
> The `cluster.unique_db_id` is set to 3.

```
 ALTER DATABASE MMRnode_c SET cluster.unique_db_id TO 3;
SET cluster.unique_db_id TO 3;

CREATE SEQUENCE MMR_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
```

```
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;

CREATE OR REPLACE FUNCTION MMR_sequence_currval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  currval($1::regclass);
$function$;
```

Create the multi-master replication system with the **Replicate Publication Schema** and the **Perform Initial Snapshot** options selected when creating the additional primary nodes `MMRnode_b` and `MMRnode_c`.

The resulting primary nodes are shown in the Replication Server console.

> **Note**
>
> The `Default Value` property of the `id` column uses the `MMR_sequence_nextval` function.

Invoke the following `INSERT` commands on `MMRnode_a`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
1');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
2');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
3');
```

Invoke the following `INSERT` on `MMRnode_b`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_b - Row
1');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_b - Row
2');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_b - Row
3');
```

Invoke the following `INSERT` on `MMRnode_c`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_c - Row
1');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_c - Row
2');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_c - Row
3');
```

A synchronization replication is performed.

No uniqueness conflicts occur. A unique value is generated for the `id` primary key column as shown by the following results on `MMRnode_a`:

```
MMRnode_a=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
```

```
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  9007199254740993 | MMRnode_b - Row 1
  9007199254740994 | MMRnode_b - Row 2
  9007199254740995 | MMRnode_b - Row 3
 13510798882111489 | MMRnode_c - Row 1
 13510798882111490 | MMRnode_c - Row 2
 13510798882111491 | MMRnode_c - Row 3
(9 rows)
```

The same query on `MMRnode_b` shows the same set of rows:

```
MMRnode_b=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
```

```
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  9007199254740993 | MMRnode_b - Row 1
  9007199254740994 | MMRnode_b - Row 2
  9007199254740995 | MMRnode_b - Row 3
 13510798882111489 | MMRnode_c - Row 1
 13510798882111490 | MMRnode_c - Row 2
 13510798882111491 | MMRnode_c - Row 3
(9 rows)
```

The same results are present on `MMRnode_c`:

```
MMRnode_c=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
```

```
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  9007199254740993 | MMRnode_b - Row 1
  9007199254740994 | MMRnode_b - Row 2
  9007199254740995 | MMRnode_b - Row 3
 13510798882111489 | MMRnode_c - Row 1
 13510798882111490 | MMRnode_c - Row 2
 13510798882111491 | MMRnode_c - Row 3
(9 rows)
```

9.6.6.3 Converting a standard sequence to an MMR-ready sequence

If you have an existing application with tables that use a standard sequence such as with the `SERIAL` data type, you can modify these tables to use the MMR-ready sequence for incorporation into a multi-master replication system. The basic conversion process is:

1. Update the sequence values in the existing rows with MMR-ready sequence compatible values.
2. Modify the column definition to be compatible with the MMR-ready sequence. These changes include modifying or adding the `DEFAULT` clause to use the MMR-ready sequence function to supply the default values for later inserts.

To perform the conversion of existing sequence values, first create the unique database identifier as described in Step 1 of Creating an MMR-ready sequence.

Use the following function to update the existing primary key and foreign key values that you need to convert.

```
CREATE OR REPLACE FUNCTION MMR_sequence_convert
(
    old_seq_value
bigint
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
$1;
$function$;
```

The function input argument is the old sequence value. The function returns the new MMR-ready sequence value.

The function input and return arguments are data type `BIGINT`, so you must modify the existing sequence columns accordingly before using the function.

Finally, the sequence columns must include the clauses `BIGINT NOT NULL DEFAULT MMR_sequence_nextval('seq_name')` to supply MMR-ready sequence values for future inserts.

See Creating an MMR-ready sequence for information on creating the objects required for an MMR-ready sequence.

9.6.6.4 Conversion to an MMR-ready sequence example

This example shows how you convert two tables with standard sequences used as primary keys and a parent-child relationship by a foreign key constraint to use the MMR-ready sequence, You can then use them in a multi-master replication system.

The tables are defined as follows:

```
CREATE TABLE MMR_seq_tbl
(
    id          SERIAL PRIMARY
KEY,
    field       VARCHAR2(20)
);

CREATE TABLE MMR_seq_child_tbl
(
    id          SERIAL PRIMARY
KEY,
    field       VARCHAR2(20),
```

```
    parent_id   INTEGER CONSTRAINT
MMR_seq_tbl_fk
                REFERENCES MMR_seq_tbl(id)
);
```

> **Note**
>
> Observe the foreign key constraint between columns `MMR_seq_child_tbl.parent_id` and `MMR_seq_tbl.id`.

The tables are populated with an initial set of rows:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
1');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
2');
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
3');

INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 1-1',
1);
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 1-2',
1);
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 2-1',
2);
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 2-2',
2);
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 3-1',
3);
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 3-2',
3);
```

The resulting table content is the following:

```
edb=# SELECT * FROM MMR_seq_tbl;
```

```
 id |       field
----+-------------------
  1 | MMRnode_a - Row 1
  2 | MMRnode_a - Row 2
  3 | MMRnode_a - Row 3
(3 rows)
```

```
edb=# SELECT * FROM MMR_seq_child_tbl;
```

```
 id |        field        | parent_id
----+---------------------+-----------
  1 | MMRnode_a - Row 1-1 |         1
  2 | MMRnode_a - Row 1-2 |         1
  3 | MMRnode_a - Row 2-1 |         2
  4 | MMRnode_a - Row 2-2 |         2
  5 | MMRnode_a - Row 3-1 |         3
  6 | MMRnode_a - Row 3-2 |         3
(6 rows)
```

Prior to performing the conversion, obtain the current, maximum sequence value of the sequence to convert to an MMR-ready sequence. In this example, the value is `6`, as seen in the `id` column of table `MMR_seq_child_tbl`.

This value is needed to set a newly created sequence to use for the MMR-ready sequence. This value must be a large enough starting value to avoid auniqu eness conflict with the converted sequence values of the existing rows.

Converting existing standard sequence values

To convert the existing sequence values in columns `MMR_seq_tbl.id`, `MMR_seq_child_tbl.id`, and `MMR_seq_child_tbl.parent_id`:

Permit deferred updates to the foreign key constraint.

```
ALTER TABLE MMR_seq_child_tbl ALTER CONSTRAINT MMR_seq_tbl_fk DEFERRABLE INITIALLY
DEFERRED;
```

Create the function to perform the sequence conversion.

```
CREATE OR REPLACE FUNCTION MMR_sequence_convert
(
    old_seq_value
bigint
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
$1;
$function$;
```

Change the sequence columns to data type `BIGINT` so they are large enough for the MMR-ready sequence.

```
ALTER TABLE MMR_seq_tbl ALTER COLUMN id SET DATA TYPE
 BIGINT;
ALTER TABLE MMR_seq_child_tbl ALTER COLUMN id SET DATA TYPE BIGINT;
ALTER TABLE MMR_seq_child_tbl ALTER COLUMN parent_id SET DATA TYPE BIGINT;
```

Set the unique database identifier used by the MMR-ready sequence.

```
ALTER DATABASE MMRnode_a SET cluster.unique_db_id TO 1;
SET cluster.unique_db_id TO 1;
```

Update the primary key and foreign key values with the `MMR_sequence_convert` function. Perform the updates affecting the foreign key constraint in the same transaction to avoid a foreign key violation error.

```
BEGIN TRANSACTION;
    UPDATE MMR_seq_tbl SET id = MMR_sequence_convert
(id);
    UPDATE MMR_seq_child_tbl SET parent_id = MMR_sequence_convert (parent_id);
    UPDATE MMR_seq_child_tbl SET id = MMR_sequence_convert
(id);
COMMIT;
```

Reset the foreign key constraint back to its original setting. For example:

```
ALTER TABLE MMR_seq_child_tbl ALTER CONSTRAINT MMR_seq_tbl_fk NOT
DEFERRABLE;
```

After the conversion to the MMR-ready sequence, the table content is as follows:

```
edb=# SELECT * FROM MMR_seq_tbl;
```

```
        id        |       field
------------------+-------------------
 4503599627370497 | MMRnode_a - Row 1
 4503599627370498 | MMRnode_a - Row 2
 4503599627370499 | MMRnode_a - Row 3
(3 rows)

edb=# SELECT * FROM MMR_seq_child_tbl;
        id        |        field         |    parent_id
------------------+----------------------+------------------
 4503599627370497 | MMRnode_a - Row 1-1 | 4503599627370497
 4503599627370498 | MMRnode_a - Row 1-2 | 4503599627370497
 4503599627370499 | MMRnode_a - Row 2-1 | 4503599627370498
 4503599627370500 | MMRnode_a - Row 2-2 | 4503599627370498
 4503599627370501 | MMRnode_a - Row 3-1 | 4503599627370499
 4503599627370502 | MMRnode_a - Row 3-2 | 4503599627370499
(6 rows)
```

The parent-child foreign key relationship between columns `MMR_seq_child_tbl.parent_id` and `MMR_seq_tbl.id` is maintained.

The primary key id values incorporate the old sequence values but are increased by the addition of the 52-bit shifted, database identifier value.

Setting up the MMR-ready sequence

Perform the steps described in Creating an MMR-ready sequence on the databases to use as primary nodes. For database `MMRnode_a` that contains the converted tables, create a new sequence with a starting value of `7` to avoid a primary key uniqueness conflict with the existing rows. In the original tables, the maximum used sequence value was 6. When a sequence number is transformed to an MMR-ready sequence value, the same result is returned if the same database identifier is used with the same original number.

```
CREATE SEQUENCE MMR_seq START WITH 7 INCREMENT BY 1 NO CYCLE;
```

Create the function to return the MMR-ready sequence value.

```
CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;
```

Modify the primary key columns to use the function to return the default value.

```
ALTER TABLE MMR_seq_tbl ALTER COLUMN id SET DEFAULT
 MMR_sequence_nextval('MMR_seq');
ALTER TABLE MMR_seq_child_tbl ALTER COLUMN id SET DEFAULT
MMR_sequence_nextval('MMR_seq');
```

Repeat the MMR-ready sequence setup process for the other primary nodes.

```
 ALTER DATABASE MMRnode_b SET cluster.unique_db_id TO 2;
SET cluster.unique_db_id TO 2;

CREATE SEQUENCE MMR_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
  seq_id
VARCHAR

)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;
```

Repeat the process for `MMRnode_c`.

```
 ALTER DATABASE MMRnode_c SET cluster.unique_db_id TO 3;
SET cluster.unique_db_id TO 3;

CREATE SEQUENCE MMR_seq START WITH 1 INCREMENT BY 1 NO CYCLE;

CREATE OR REPLACE FUNCTION MMR_sequence_nextval
(
    seq_id
VARCHAR
)
RETURNS
bigint
LANGUAGE
sql
AS
$function$
SELECT
(
 (SELECT
current_setting('cluster.unique_db_id'))::bigint
  << 52)::bigint +
  nextval($1::regclass);
$function$;
```

Tables after initial multi-master replication system creation

Create the multi-master replication system using databases `MMRnode_a`, `MMRnode_b`, and `MMRnode_c` in a similar manner as described in MMR-ready sequence example.

After you create the system with the initial snapshot, `MMRnode_a`, `MMRnode_b`, and `MMRnode_c` all contain identical content. The following is the table content:

```
MMRnode_a=# SELECT * FROM MMR_seq_tbl;
        id        |       field
------------------+-------------------
 4503599627370497 | MMRnode_a - Row 1
 4503599627370498 | MMRnode_a - Row 2
 4503599627370499 | MMRnode_a - Row 3
(3 rows)

MMRnode_a=# SELECT * FROM MMR_seq_child_tbl;
        id        |        field        |    parent_id
------------------+---------------------+------------------
 4503599627370497 | MMRnode_a - Row 1-1 | 4503599627370497
 4503599627370498 | MMRnode_a - Row 1-2 | 4503599627370497
 4503599627370499 | MMRnode_a - Row 2-1 | 4503599627370498
 4503599627370500 | MMRnode_a - Row 2-2 | 4503599627370498
 4503599627370501 | MMRnode_a - Row 3-1 | 4503599627370499
 4503599627370502 | MMRnode_a - Row 3-2 | 4503599627370499
(6 rows)
```

Subsequent row insertions and synchronization

The following rows are inserted on `MMRnode_a`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_a - Row
4');
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_a - Row 4-1',
4503599627370503);

MMRnode_a=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
        id        |
field
------------------+-------------------
 4503599627370497 | MMRnode_a - Row
1
 4503599627370498 | MMRnode_a - Row
2
 4503599627370499 | MMRnode_a - Row
3
 4503599627370503 | MMRnode_a - Row
4
(4 rows)

MMRnode_a=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id        |        field        |
parent_id
------------------+---------------------+------------------
 4503599627370497 | MMRnode_a - Row 1-1 |
4503599627370497
 4503599627370498 | MMRnode_a - Row 1-2 |
4503599627370497
 4503599627370499 | MMRnode_a - Row 2-1 |
4503599627370498
 4503599627370500 | MMRnode_a - Row 2-2 |
4503599627370498
 4503599627370501 | MMRnode_a - Row 3-1 |
4503599627370499
 4503599627370502 | MMRnode_a - Row 3-2 |
4503599627370499
 4503599627370504 | MMRnode_a - Row 4-1 |
4503599627370503
```

```
(7 rows)
```

The following rows are inserted on `MMRnode_b`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_b - Row
1');
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_b - Row 1-1',
9007199254740993);

MMRnode_b=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
        id        |
field
------------------+------------------
 4503599627370497 | MMRnode_a - Row
1
 4503599627370498 | MMRnode_a - Row
2
 4503599627370499 | MMRnode_a - Row
3
 9007199254740993 | MMRnode_b - Row
1
(4 rows)

MMRnode_b=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id        |        field        |
parent_id
------------------+---------------------+------------------
 4503599627370497 | MMRnode_a - Row 1-1 |
4503599627370497
 4503599627370498 | MMRnode_a - Row 1-2 |
4503599627370497
 4503599627370499 | MMRnode_a - Row 2-1 |
4503599627370498
 4503599627370500 | MMRnode_a - Row 2-2 |
4503599627370498
 4503599627370501 | MMRnode_a - Row 3-1 |
4503599627370499
 4503599627370502 | MMRnode_a - Row 3-2 |
4503599627370499
 9007199254740994 | MMRnode_b - Row 1-1 |
9007199254740993
(7 rows)
```

The following rows are inserted on `MMRnode_c`:

```
INSERT INTO MMR_seq_tbl (field) VALUES ('MMRnode_c - Row
1');
INSERT INTO MMR_seq_child_tbl (field, parent_id) VALUES ('MMRnode_c - Row 1-1',
135107988821114 89);

MMRnode_c=# SELECT * FROM MMR_seq_tbl ORDER BY
id;
        id         |
field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row
1
  4503599627370498 | MMRnode_a - Row
2
  4503599627370499 | MMRnode_a - Row
3
 135107988821114 89 | MMRnode_c - Row
1
(4 rows)
```

```
MMRnode_c=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id         |        field        |
parent_id
-------------------+---------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1-1 |
4503599627370497
  4503599627370498 | MMRnode_a - Row 1-2 |
4503599627370497
  4503599627370499 | MMRnode_a - Row 2-1 |
4503599627370498
  4503599627370500 | MMRnode_a - Row 2-2 |
4503599627370498
  4503599627370501 | MMRnode_a - Row 3-1 |
4503599627370499
  4503599627370502 | MMRnode_a - Row 3-2 |
4503599627370499
 13510798882111490 | MMRnode_c - Row 1-1 |
13510798882111489
(7 rows)
```

After you perform a synchronization replication, there are no uniqueness conflicts. The following shows the synchronized, consistent tables in the primary nodes:

Content of `MMRnode_a` after synchronization:

```
MMRnode_a=# SELECT * FROM MMR_seq_tbl ORDER BY id;
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  4503599627370503 | MMRnode_a - Row 4
  9007199254740993 | MMRnode_b - Row 1
 13510798882111489 | MMRnode_c - Row 1
(6 rows)

MMRnode_a=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id         |        field        |     parent_id
-------------------+---------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1-1 |  4503599627370497
  4503599627370498 | MMRnode_a - Row 1-2 |  4503599627370497
  4503599627370499 | MMRnode_a - Row 2-1 |  4503599627370498
  4503599627370500 | MMRnode_a - Row 2-2 |  4503599627370498
  4503599627370501 | MMRnode_a - Row 3-1 |  4503599627370499
  4503599627370502 | MMRnode_a - Row 3-2 |  4503599627370499
  4503599627370504 | MMRnode_a - Row 4-1 |  4503599627370503
  9007199254740994 | MMRnode_b - Row 1-1 |  9007199254740993
 13510798882111490 | MMRnode_c - Row 1-1 | 13510798882111489
(9 rows)
```

Content of `MMRnode_b` after synchronization:

```
MMRnode_b=# SELECT * FROM MMR_seq_tbl ORDER BY id;
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  4503599627370503 | MMRnode_a - Row 4
  9007199254740993 | MMRnode_b - Row 1
 13510798882111489 | MMRnode_c - Row 1
(6 rows)
```

```
MMRnode_b=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id         |        field        |     parent_id
-------------------+---------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1-1 |  4503599627370497
  4503599627370498 | MMRnode_a - Row 1-2 |  4503599627370497
  4503599627370499 | MMRnode_a - Row 2-1 |  4503599627370498
  4503599627370500 | MMRnode_a - Row 2-2 |  4503599627370498
  4503599627370501 | MMRnode_a - Row 3-1 |  4503599627370499
  4503599627370502 | MMRnode_a - Row 3-2 |  4503599627370499
  4503599627370504 | MMRnode_a - Row 4-1 |  4503599627370503
  9007199254740994 | MMRnode_b - Row 1-1 |  9007199254740993
 13510798882111490 | MMRnode_c - Row 1-1 | 13510798882111489
(9 rows)
```

Content of `MMRnode_c` after synchronization:

```
MMRnode_c=# SELECT * FROM MMR_seq_tbl ORDER BY id;
        id         |       field
-------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1
  4503599627370498 | MMRnode_a - Row 2
  4503599627370499 | MMRnode_a - Row 3
  4503599627370503 | MMRnode_a - Row 4
  9007199254740993 | MMRnode_b - Row 1
 13510798882111489 | MMRnode_c - Row 1
(6 rows)

MMRnode_c=# SELECT * FROM MMR_seq_child_tbl ORDER BY id;
        id         |        field        |     parent_id
-------------------+---------------------+-------------------
  4503599627370497 | MMRnode_a - Row 1-1 |  4503599627370497
  4503599627370498 | MMRnode_a - Row 1-2 |  4503599627370497
  4503599627370499 | MMRnode_a - Row 2-1 |  4503599627370498
  4503599627370500 | MMRnode_a - Row 2-2 |  4503599627370498
  4503599627370501 | MMRnode_a - Row 3-1 |  4503599627370499
  4503599627370502 | MMRnode_a - Row 3-2 |  4503599627370499
  4503599627370504 | MMRnode_a - Row 4-1 |  4503599627370503
  9007199254740994 | MMRnode_b - Row 1-1 |  9007199254740993
 13510798882111490 | MMRnode_c - Row 1-1 | 13510798882111489
(9 rows)
```

9.6.7 Automatic conflict-resolution example

This example shows a scenario where a transaction change originating from the first primary node is successfully applied to the second primary node but conflicts with the third primary node. The conflict is resolved automatically.

The conflict resolution option is set to latest timestamp.

Timestamp	Action			
`t0`		id = 2, address = 'ADDR'	id = 2, address = 'ADDR'	id = 2, address = 'ADDR'

Timestamp	Action			
t1	**Node A:** UPDATE addrbook SET address = 'ADDR A' WHERE id = 2;	id = 2, address = **'ADDR A'**	id = 2, address = 'ADDR'	id = 2, address = 'ADDR'
t2	**Node C:** UPDATE addrbook SET address = 'ADDR C' WHERE id = 2;	id = 2, address = 'ADDR A'	id = 2, address = 'ADDR'	id = 2, address = **'ADDR C'**
t3	Synchronization pushes Node A changes to Node B. Changes successfully applied.	id = 2, address = 'ADDR A'	id = 2, address = **'ADDR A'**	id = 2, address = 'ADDR C'
t4	Synchronization pushes Node A changes to Node C. Current address on Node C `<>` old value on Node A (`'ADDR C' <> 'ADDR'`) hence conflict detected. Latest change on Node C accepted and Node A change discarded.	id = 2, address = 'ADDR A'	id = 2, address = 'ADDR A'	id = 2, address = **'ADDR C'**
t5	No changes on Node B. Node C changes pushed to Node A that is successfully applied (Node A change already marked as discarded and hence is overwritten.)	id = 2, address = **'ADDR C'**	id = 2, address = 'ADDR A'	id = 2, address = 'ADDR C'
t6	Node C changes pushed to Node B that is successfully applied. All nodes are in sync and have consistent state.	id = 2, address = 'ADDR C'	id = 2, address = **'ADDR C'**	id = 2, address = 'ADDR C'

In a few situations, an update/update conflict might not be properly resolved according to the selected resolution options.

Update/update conflict on column where new value is identical to original value

Suppose there's an update to a publication table where the updated column value is the same as the original column value and then an update/update conflict occurs involving that column. It's possible that the final value of this column isn't set according to the chosen conflict resolution option in one of the conflicting primary nodes. This is a known limitation.

For example, consider the following row in the dept table:

```
deptno |   dname    |  loc
--------+------------+--------
     40 | OPERATIONS | BOSTON
```

First, the following UPDATE statement is given in the primary definition node:

```
edb=# UPDATE dept SET dname = 'OPERATIONS', loc = 'BEDFORD' WHERE deptno =
 40;
UPDATE 1
edb=# SELECT * FROM dept WHERE deptno =
40;
```

```
 deptno |   dname    |   loc
--------+------------+---------
     40 | OPERATIONS | BEDFORD
(1 row)
```

> **Note**

> The original value, `OPERATIONS`, of column `dname` is the same as the value to which it is changed in the `UPDATE` statement.

The following `UPDATE` statement is then given in a second primary node:

```
MMRnode=# UPDATE dept SET dname = 'LOGISTICS', loc = 'CAMBRIDGE' WHERE deptno =
40;
UPDATE 1
MMRnode=# SELECT * FROM dept WHERE deptno =
40;
```

```
 deptno |   dname   |    loc
--------+-----------+-----------
     40 | LOGISTICS | CAMBRIDGE
(1 row)
```

After a synchronization replication using the earliest timestamp conflict-resolution option, the row in the primary definition node retains the update performed on it as expected since the update on the primary definition node occurred first.

```
edb=# SELECT * FROM dept WHERE deptno =
40;
```

```
 deptno |   dname    |   loc
--------+------------+---------
     40 | OPERATIONS | BEDFORD
(1 row)
```

However, the value of column `dname` in the second primary node remains set to `LOGISTICS`. It didn't revert to the value `OPERATIONS` from the primary definition node as would normally be expected on a conflicting column. However, as expected, the value in column `loc` revertes from `CAMBRIDGE` to the primary definition node value of `BEDFORD`.

```
MMRnode=# SELECT * FROM dept WHERE deptno =
40;
```

```
 deptno |   dname   |   loc
--------+-----------+---------
     40 | LOGISTICS | BEDFORD
(1 row)
```

Update/update conflict on primary key columns

An update/update conflict on the primary key column might not resolve consistently resolved according to the selected resolution option. That is, the column values might differ for the same row across multiple primary nodes after the synchronization replication.

In addition, this conflict might not appear under the **Conflict History** tab in the Replication Server console. Even if a conflict resolution entry does appear under the **Conflict History** tab, the actual primary key values might not be consistent across the nodes as implied by the conflict resolution.

9.6.8 Custom conflict handling

For update/update conflicts, custom conflict handling uses a PL/pgSQL function to resolve the conflict. If you're using EDB Postgres Advanced Server, you can use a Stored Procedure Language (SPL) function. When an update/update conflict is detected, the function is called. How you set a certain parameter in the function determines the outcome of the conflict.

You must provide the function and add it to the primary definition node using a utility such as PSQL or pgAdmin (Postgres Enterprise Manager Client in EDB Postgres Advanced Server).

9.6.8.1 Custom conflict-handling function

An update/update conflict occurs with at least one conflicting column in the table.

A column is considered a conflicting column if it's updated on more than one primary node in the same synchronization. Even if the new, updated value for the column is identical in the conflicting update transactions, the fact that the same column was updated on more than one primary node makes it a conflicting column.

Each publication table must have its own custom conflict-handling function to handle custom resolution for update/update conflicts on that publication table.

Custom conflict handling is designed to provide one of the following three outcomes based on how you set the `resolution_code` parameter:

- **Columns are set to the source node.** When you set the `resolution_code` parameter of the function to `1`, the setting of all columns in both conflicting nodes is obtained from the source node of the replication.
- **Columns are set to the target node.** When you set the `resolution_code` parameter of the function to `2`, the setting of all columns in both conflicting nodes is obtained from the target node of the replication.
- **The function logic sets the column.** When you set the `resolution_code` parameter of the function to `3`, the setting of the first conflicting column is obtained from the value returned in the source parameter coded in the function logic. The setting of all other column values is obtained from the source node of the replication.

The following is an example of a custom conflict-handling function where the conflicting columns are set to the target node.

```
CREATE OR REPLACE FUNCTION edb.custom_conflict_dept
(
    INOUT   source
_edb_replicator_pub.rrst_edb_dept,
    IN      target
_edb_replicator_pub.rrst_edb_dept,
    IN      conflict_column     VARCHAR(255),
    OUT     resolution_message  VARCHAR(255),
    OUT     resolution_code     INTEGER
)
AS
$$
DECLARE
BEGIN
    resolution_code := 2;
    resolution_message := 'Custom conflict handling: Target node wins on edb.dept
';
END;
$$
LANGUAGE
plpgsql;
```

If the multi-master replication system is configured with the log-based method of synchronization replication, the shadow tables of the `INOUT` source and `IN` target parameters are replaced with the actual publication tables, as shown by the following:

```
CREATE OR REPLACE FUNCTION edb.custom_conflict_dept
(
    INOUT   source              edb.dept,
    IN      target
edb.dept,
    IN      conflict_column     VARCHAR(255),
    OUT     resolution_message  VARCHAR(255),
```

```
    OUT     resolution_code     INTEGER
)
AS
$$
DECLARE
BEGIN
    resolution_code := 2;
    resolution_message := 'Custom conflict handling: Target node wins on edb.dept
';
END;
$$
LANGUAGE
plpgsql;
```

The following is an example of a custom conflict-handling function where the function logic determines the value of the first conflicting column.

```
CREATE OR REPLACE FUNCTION edb.custom_conflict_emp
(
    INOUT   source              _edb_replicator_pub.rrst_edb_emp,
    IN      target
_edb_replicator_pub.rrst_edb_emp,
    IN      conflict_column     VARCHAR(255),
    OUT     resolution_message  VARCHAR(255),
    OUT     resolution_code     INTEGER
)
AS
$$
DECLARE
BEGIN
    resolution_code := 3;
    source.ename := 'Unknown';
    source.job := 'Unknown';
    source.mgr := 0;
    source.hiredate := '2013-01-01';
    source.sal := 0;
    source.comm := 0;
    resolution_message := 'Custom conflict handling: Defaults set on
edb.emp';
END;
$$
LANGUAGE
plpgsql;
```

In this example, only the first conflicting column (based on the column order in the table) is set to the value coded in the function. All other assignments to the source parameter are ignored. These other columns are set to the source node.

Function parameters

`source`

`INOUT` parameter of the record type of the shadow table in schema `_edb_replicator_pub` of the primary definition node on which to resolve the conflicts. If you use the log-based method of synchronization replication, specify the actual publication table instead of the shadow table. The input values are the column values from the source node. When `resolution_code` is set to `3`, set the columns in this parameter to the values to use for the final outcome.

`target`

`IN` parameter of the record type of the shadow table in schema `_edb_replicator_pub` of the primary definition node on which to resolve conflicts. If you use the log-based method of synchronization replication, specify the actual publication table instead of the shadow table. The input values are the

column values from the target node.

`conflict_column`

`IN` parameter of type `VARCHAR(255)` containing the name of the column on which the update/update conflict occurred. If more than one column is involved in the conflict, the name of the first conflicting column is returned.

`resolution_message`

`OUT` parameter of type `VARCHAR(255)` containing an informative message to write to the publication server log file. Set the publication server configuration option `logging.level` to at least the `INFO` level for the messages to appear in the publication server log file. See Installation details for the location of the publication server log file.

`resolution_code`

`OUT` parameter of type `INTEGER` that you set to one of the following values to determine how to resolve the conflict:

- `1` to use the column values of the source node of the replication for the final outcome
- `2` to use the column values of the target node of the replication for the final outcome
- `3` to use the value set for the source `INOUT` parameter of the first conflicting column as the final outcome for that column.

9.6.8.2 Adding a custom conflict-handling function

To add a custom conflict handling function to the primary definition node:

1. Add the publication under the primary definition node. See Adding a publication.

2. Add the function to the primary definition node. The following example shows adding the function using PSQL.

```
edb=# \i /home/user/custom_conflict_dept.sql
CREATE FUNCTION
```

3. Open the **Conflict Resolution Options** tab. Select **Publication > Update Publication > Conflict Resolution Options**.

4. For the table on which you want to use custom conflict handling, select**Custom** from the corresponding list. In the **Custom Handler** text box, enter the schema and function name used in the `CREATE FUNCTION` statement.

5. Select **Update**, and then select **OK** in response to confirmation message.

> **Note**
>
> If the multi-master replication system uses custom conflict handling and you later switch the role of the primary definition node to another primary node, you must add the functions to the new primary definition node.

> **Note**
>
> If you want to delete the multi-master replication system, before removing the publication, remove all custom conflict-handling functions from the primary definition node.

The following example shows deleting a custom conflict function.

```
DROP FUNCTION
edb.custom_conflict_dept(_edb_replicator_pub.rrst_edb_dept,_edb_replicator_pub.rrst_edb_dept,varchar);
```

9.6.8.3 Custom conflict handling examples

Setting columns from the source or target

The following example shows the effect of custom conflict handling using the custom conflict-handling function named `custom_conflict_dept` shown in Custom conflict handling function. This function sets the target node as the winner of update/update conflicts on the dept table.

The update is made on the primary definition node, `edb`:

```
edb=# UPDATE dept SET loc = 'PORTLAND' WHERE deptno =
50;
UPDATE 1
edb=# SELECT * FROM dept;
```

```
 deptno |    dname     |   loc
--------+--------------+----------
     10 | ACCOUNTING   | NEW YORK
     20 | RESEARCH     | DALLAS
     30 | SALES        | CHICAGO
     40 | OPERATIONS   | BOSTON
     50 | ADVERTISING | PORTLAND
(5 rows)
```

The following update is made on a second primary node, `MMRnode`:

```
MMRnode=# UPDATE dept SET loc = 'LOS ANGELES' WHERE deptno =
50;
UPDATE 1
MMRnode=# SELECT * FROM dept;
```

```
 deptno |    dname     |     loc
--------+--------------+-------------
     10 | ACCOUNTING   | NEW YORK
     20 | RESEARCH     | DALLAS
     30 | SALES        | CHICAGO
     40 | OPERATIONS   | BOSTON
     50 | ADVERTISING | LOS ANGELES
(5 rows)
```

After a synchronization replication, the update/update conflict is detected and resolved as shown in the**Conflict History** tab.

In the source primary node, the `loc` column of department 50 loses the value set in its `UPDATE` statement. The column is reset to the value from the target primary node.

```
edb=# SELECT * FROM dept;
```

```
 deptno |    dname     |     loc
--------+--------------+-------------
     10 | ACCOUNTING   | NEW YORK
     20 | RESEARCH     | DALLAS
     30 | SALES        | CHICAGO
     40 | OPERATIONS   | BOSTON
     50 | ADVERTISING | LOS ANGELES
(5 rows)
```

In the target primary node, the `loc` column of department 50 retains the value set from its `UPDATE` statement.

```
MMRnode=# SELECT * FROM dept;
```

```
 deptno |    dname     |     loc
--------+--------------+-------------
     10 | ACCOUNTING   | NEW YORK
     20 | RESEARCH     | DALLAS
     30 | SALES        | CHICAGO
     40 | OPERATIONS   | BOSTON
     50 | ADVERTISING | LOS ANGELES
(5 rows)
```

The target node wins the conflict as determined by setting the `resolution_code` parameter to `2` in the custom conflict-handling function.

Setting columns from the function logic

The following example shows the effect of custom conflict handling using the custom conflict-handling function `custom_conflict_emp` shown in Custom conflict-handling function. This function sets values coded in the function as the winner of update/update conflicts on the `emp` table.

The following is the row from the emp table prior to the update:

```
edb=# edb=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+-------+----------+------+---------------------+---------+---------+--------
  9001 | SMITH | SALESMAN | 7698 | 31-OCT-13 00:00:00 | 8000.00 | 4000.00 |     30
(1 row)
```

The following update is made in the primary definition node, `edb`:

```
edb=# UPDATE emp SET ename = 'JONES', mgr = 7900, sal = 8500, comm = 5000 WHERE empno =
9001;
UPDATE 1
edb=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+-------+----------+------+---------------------+---------+---------+--------
  9001 | JONES | SALESMAN | 7900 | 31-OCT-13 00:00:00 | 8500.00 | 5000.00 |     30
(1 row)
```

The following update is made in a second primary node, `MMRnode`:

```
MMRnode=# UPDATE emp SET ename = 'ROGERS', mgr = 7788, sal = 9500, comm = 5000 WHERE empno =
9001;
UPDATE 1
MMRnode=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+----------+------+---------------------+---------+---------+--------
  9001 | ROGERS | SALESMAN | 7788 | 31-OCT-13 00:00:00 | 9500.00 | 5000.00 |     30
(1 row)
```

After the synchronization replication of the primary node, `edb` contains the following values for the conflicting row:

```
edb=# SELECT * FROM emp WHERE empno =
9001;
```

```
empno |  ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+---------+----------+------+---------------------+---------+---------+--------
  9001 | Unknown | SALESMAN | 7900 | 31-OCT-31 00:00:00 | 8500.00 | 5000.00 |     30
(1 row)
```

After the synchronization replication of the primary node, `MMRnode`, contains the following values for the conflicting row:

```
MMRnode=# SELECT * FROM emp WHERE empno =
9001;
```

```
empno |  ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+---------+----------+------+---------------------+---------+---------+--------
  9001 | Unknown | SALESMAN | 7900 | 31-OCT-31 00:00:00 | 8500.00 | 5000.00 |     30
(1 row)
```

The value of the first conflicting column is determined by the custom conflict-handling function for both primary nodes.

Setting columns using the source and target shadow tables

The following example shows how to use the values from the source and target shadow tables to set the final values in the conflicting column.

> **Note**
>
> This custom conflict-handling function uses a column (`rrep_old_quantity` in this example) that's a column of the shadow table and not of the actual publication table. So you can't use this solution for a publication that uses the log-based method of synchronization replication.

This example use the following table, which contains product inventory.

```
CREATE TABLE inventory
(
    item_id         NUMERIC PRIMARY KEY,
    name            VARCHAR(20),
    quantity
INTEGER
);
INSERT INTO inventory VALUES (1, 'LaserJet Printer 610',
50);
INSERT INTO inventory VALUES (2, 'Scanner 510',
10);
INSERT INTO inventory VALUES (3, 'LCD',
20);
```

When products are purchased at different locations, resulting in an inventory reduction on several primary nodes, the remaining inventory must be properly updated on all primary nodes to reflect the reduction in all locations. The custom conflict-handling function is coded to properly record the remaining inventory if changes to the same item are made in several locations.

The following example uses the primary definition node, `edb` and a second primary node, `MMRnode`. Initially, the inventory table has the same contents on both primary nodes.

```
edb=# SELECT * FROM inventory;
```

```
 item_id |         name         | quantity
---------+----------------------+----------
       1 | LaserJet Printer 610 |       50
       2 | Scanner 510          |       10
       3 | LCD                  |       20
(3 rows)
```

After creating the primary nodes, the following shows the resulting shadow table structures in the primary definition node:

```
edb=# \d
_edb_replicator_pub.rrst_edb_inventory;
                  Table "_edb_replicator_pub.rrst_edb_inventory"
```

```
         Column          |            Type             |         Modifiers
-------------------------+-----------------------------+---------------------------
 rrep_sync_id            | numeric                     | not null
 rrep_common_id          | numeric                     |
 rrep_operation_type     | character(1)                |
 rrep_tx_timestamp       | timestamp without time zone | default current_timestamp
 item_id                 | numeric                     |
 name                    | character varying(20)       |
 quantity                | integer                     |
 rrep_old_item_id        | numeric                     |
 rrep_old_name           | character varying(20)       |
 rrep_old_quantity       | integer                     |
 rrep_tx_conflict_status | character(1)                |
Indexes:
    "rrst_edb_inventory_pkey" PRIMARY KEY, btree (rrep_sync_id)
```

Similarly, in the second primary node the same shadow table is created.

```
MMRnode=# \d _edb_replicator_pub.rrst_edb_inventory
                  Table "_edb_replicator_pub.rrst_edb_inventory"
```

```
         Column          |            Type             |         Modifiers
-------------------------+-----------------------------+---------------------------
 rrep_sync_id            | numeric                     | not null
 rrep_common_id          | numeric                     |
 rrep_operation_type     | character(1)                |
 rrep_tx_timestamp       | timestamp without time zone | default current_timestamp
 item_id                 | numeric                     |
 name                    | character varying(20)       |
 quantity                | integer                     |
 rrep_old_item_id        | numeric                     |
 rrep_old_name           | character varying(20)       |
 rrep_old_quantity       | integer                     |
 rrep_tx_conflict_status | character(1)                |
Indexes:
    "rrst_edb_inventory_pkey" PRIMARY KEY, btree (rrep_sync_id)
```

For an update transaction, the shadow table contains the column values both:

- Before the update was made on the publication table (columns with names `rrep_old_column_name`)
- After the update was applied (columns named identically to the publication table column names)

The custom conflict-handling function uses both the current and old values of the quantity columns from the source and target shadow tables as shown by the following.

```
CREATE OR REPLACE FUNCTION edb.custom_conflict_inventory
(
    INOUT   source
_edb_replicator_pub.rrst_edb_inventory,
    IN      target
_edb_replicator_pub.rrst_edb_inventory,
    IN      conflict_column     VARCHAR(255),
    OUT     resolution_message  VARCHAR(255),
    OUT     resolution_code     INTEGER
)
AS
$$
DECLARE
BEGIN
    source.quantity := source.rrep_old_quantity
      - ((source.rrep_old_quantity - source.quantity)
       + (target.rrep_old_quantity - target.quantity));
    resolution_code := 3;
    resolution_message := 'Custom conflict handling: Quantity
adjusted';
END;
$$
LANGUAGE
plpgsql;
```

Assume two items with `item_id` of `1` are purchased on the primary definition node:

```
edb=# UPDATE inventory SET quantity = quantity - 2 WHERE item_id =
1;
UPDATE 1
edb=# SELECT * FROM inventory WHERE item_id = 1;
```

```
 item_id |         name         | quantity
---------+----------------------+----------
       1 | LaserJet Printer 610 |       48
(1 row)
```

Also assume one item with `item_id` of `1` is purchased from the second primary node:

```
MMRnode=# UPDATE inventory SET quantity = quantity - 1 WHERE item_id =
1;
UPDATE 1
MMRnode=# SELECT * FROM inventory WHERE item_id = 1;
```

```
 item_id |         name         | quantity
---------+----------------------+----------
       1 | LaserJet Printer 610 |       49
(1 row)
```

After the synchronization replication and invocation of the custom conflict handling function, the quantity column for `item_id` 1 is correctly set to `47` in both primary nodes:

```
edb=# SELECT * FROM inventory WHERE item_id = 1;
```

```
 item_id |         name         | quantity
---------+----------------------+----------
       1 | LaserJet Printer 610 |       47
(1 row)
```

```
edb=# \c MMRnode MMRuser
```

```
Password for user
MMRuser:
You are now connected to database "MMRnode" as user
"MMRuser".
MMRnode=# SET search_path TO
edb;
SET
MMRnode=# SELECT * FROM inventory WHERE item_id = 1;
```

```
 item_id |         name         | quantity
---------+----------------------+----------
       1 | LaserJet Printer 610 |       47
(1 row)
```

9.6.9 Manual conflict resolution for the trigger-based method

> **Note**
>
> See Manual Conflict Resolution for the Log-Based Method for information on manual conflict resolution for multi-master replication systems configured with the log-based method of synchronization replication.

As discussed in Conflict prevention – uniqueness case, there's no built-in, automatic conflict-resolution strategy for the uniqueness (insert/insert) conflict. If a uniqueness conflict occurs, then you must modify rows in the publication tables containing the conflict and modify rows in the control schema tables in the primary nodes to resolve the conflict.

Similarly, you must manually correct update/delete and delete/update conflicts. In addition, if the conflict resolution option is set to **Manual** (see Updating the conflict resolution options) and a conflict occurs, you must also resolve this conflict in a manual fashion.

Make updates to the publication tables and the control schema tables in the primary nodes.

9.6.9.1 Finding conflicts

You can find conflicts using the **Conflict History** tab in the Replication Server console, described in Viewing conflict history.

On the **Conflict History** tab, the Source DB and Target DB columns provide the IP address and database names of the source and target primary nodes involved in the conflict.

Select **View Data** to show the details of the transactions involved in a conflict.

You can also get this information from a SQL query. Run the following query from a primary node to display information regarding pending (unresolved) conflicts:

```
SELECT DISTINCT

conflict_type,
  t.table_name,
  pk_value,
  d1.db_host AS src_db_host,
  d1.db_port AS src_db_port,
  d1.db_name AS src_db_name,
  src_rrep_sync_id,
  d2.db_host AS target_db_host,
  d2.db_port AS target_db_port,
```

```
  d2.db_name AS target_db_name,
  target_rrep_sync_id,
  c.notes
FROM _edb_replicator_pub.xdb_conflicts
c
  JOIN _edb_replicator_pub.xdb_pub_database d1 ON c.src_db_id =
d1.pub_db_id
  JOIN _edb_replicator_pub.xdb_pub_database d2 ON c.target_db_id =
d2.pub_db_id
  JOIN _edb_replicator_pub.rrep_tables t ON c.table_id =
t.table_id
WHERE resolution_status = 'P'
ORDER BY t.table_name;
```

The following is example output from the query:

```
-[ RECORD 1 ]-------+-----------------------------------------------------------
-
conflict_type       |
II
table_name          |
dept
pk_value            |
deptno=50
src_db_host         |
192.168.2.22
src_db_port         |
5444
src_db_name         |
edb
src_rrep_sync_id    |
2
target_db_host      |
192.168.2.22
target_db_port      |
5444
target_db_name      |
MMRnode
target_rrep_sync_id |
0
notes               | ERROR: duplicate key value violates unique constraint
"dept_pk"
                    |   Detail: Key (deptno)=(50) already
exists.
-[ RECORD 2 ]-------+-----------------------------------------------------------
-
conflict_type       |
II
table_name          |
dept
pk_value            |
deptno=50
src_db_host         |
192.168.2.22
src_db_port         |
5444
src_db_name         |
MMRnode
src_rrep_sync_id    |
1
target_db_host      |
192.168.2.22
target_db_port      |
5444
target_db_name      |
edb
target_rrep_sync_id |
0
```

```
notes                | ERROR: duplicate key value violates unique constraint
"dept_pk"
                     |   Detail: Key (deptno)=(50) already
exists.
-[ RECORD 3 ]--------+-----------------------------------------------------------
-
conflict_type        |
DU
table_name           |
emp
pk_value
|
src_db_host          |
192.168.2.22
src_db_port          |
5444
src_db_name          |
edb
src_rrep_sync_id     |
5
target_db_host       |
192.168.2.22
target_db_port       |
5444
target_db_name       |
MMRnode
target_rrep_sync_id |
4
```

9.6.9.2 Conflict resolution preparation

Set up a database user with certain properties so that you can modify the publication table rows to resolve conflicts manually.

Manual conflict resolution typically requires modifying rows in one or more publication tables to correct wrong entries. You can perform these changes using a utility such as `PSQL` or `pgAdmin` (Postgres Enterprise Manager Client in Advanced Server).

Usually, manual publication table corrections are isolated. Limit modifications to the publication tables you're directly changing. Don't replicate them to the other primary nodes as normally occurs in the multi-master replication system.

To prevent replication of changes to a publication table, prevent the Replication Server insert, update, and delete triggers on the publication tables from activating when you make these corrections to the publication table rows. If any of the insert, update, or delete triggers antivate, an entry is added to the publication table's shadow table. This entry results in a transaction replicated to the other primary nodes the next time synchronization replication occurs.

To prevent the triggers on the publication tables from activating, during the session in which you modify the publication table rows, set the database server configuration parameter `session_replication_role` to `replica`. (The default setting of `session_replication_role` is `origin`, in which case the triggers activate.)

The suggested method to ensure the `replica` setting is in effect is to create a database user with a default session setting of `replica` for this parameter. Whenever you connect to a database with this database user, the `replica` setting is in effect during that session.

Connect to a primary node with this database user whenever you plan to make manual corrections to the publication tables in that node that you don't want to replicate to the other primary nodes.

In the following example, database superuser `MMRmaint` is created and modified for this purpose:

```
MMRnode_a=# CREATE ROLE MMRmaint WITH LOGIN SUPERUSER PASSWORD
 'password';
CREATE ROLE
MMRnode_a=# ALTER ROLE MMRmaint SET session_replication_role TO
replica;
```

```
ALTER ROLE
```

When connected to a database with this database user, you can confirm this setting is in effect during the session using the following command:

```
MMRnode_a=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 replica
(1 row)
```

9.6.9.3 Correction strategies

Before you begin manual resolution correction, first determine the extent of the inconsistencies that occurred in the publication tables across the primary nodes of the replication system.

The **Conflict History** tab in the Replication Server console and the SQL query described in Finding conflicts can help determine the source of an initial conflict.

However, once this conflict occurs, your replication system might have processed and replicated more transactions during that synchronization operation. Some of these later replications might have succeeded, but others might have failed or produced unexpected results because of the prior conflict. With a replication schedule in effect, more synchronization operations can occur, which can create more conflicts.

Therefore, when you learn that a conflict occurred, we recommend that you stop the publication server. Use the stop option of the Linux scripts or Windows services described in Registering a publication server.

In this way, you can carefully analyze the content of the publication tables in question as well as any pending transactions in the shadow tables. This approach helps you to determine the best course of action without continued updates by the running replication system.

When analyzing your tables. you must determine the following:

- The publication tables that contain inconsistent rows across primary nodes (that is, missing rows on some primary nodes or rows with different column values for the same primary key on different primary nodes).
- Pending transactions in the shadow tables not applied to the publication tables across all primary nodes. Pending transactions are denoted by a value of `P` in the `rrep_tx_conflict_status` column of the shadow table.
- Transactions on the publication tables that have occurred and are recorded in the shadow tables following the initial conflict. Also determine whether these transactions were applied completely and correctly to the publication tables across all primary nodes. These transactions might not be marked as pending. Instead, their `rrep_tx_conflict_status` column might be set to null, meaning that no conflict was detected during replication or the transaction wasn't replicated yet. You can identify these transactions because they have a later `rrep_tx_timestamp` value than the transactions causing the initial conflict.

The general steps to resolving the problem following this analysis are the following:

1. Make the needed manual corrections to the rows in the publication tables across all primary nodes to get them into an initial, consistent state so each publication table has the same set of identical rows across primary nodes. This might be to a state before the conflicting transactions occurred, depending on what you determine to be the easiest course of action for fully resolving the conflict.

2. Apply or reapply transactions (either from your application or from the shadow tables) so that all publication tables across all primary nodes are updated consistently according to the desired, expected result of what was recorded in the shadow tables.

3. In the shadow tables, update certain indicators for conflicting entries to verify that they were resolved.

4. In the control schema, update certain indicators for the conflicting entries to show that these conflicts were resolved. This update changes the Resolution Status of these entries to Resolved in the **Conflict History** tab. These entries no longer appear in the SQL query described in Finding conflicts.

Perform these updates to the control schema of the controller database. You can determine the currently designated controller database from the content of the Replication Server configuration file (see Replication Server configuration file). The publication server ensures that the control schema changes made on the controller database are replicated to the control schemas of all publication databases. This mechanism maintains metadata consistency across all publication databases.

5. Resume operating your replication system. Start the publication server and recreate the replication schedule if you were using one.

To acheive the first two steps, use some combination of the following methods. The methods you use depend on the state of your publication tables and the extent of pending transactions that need to be applied from the shadow tables.

- **Manual publication table correction.** Use a utility such as PSQL or pgAdmin (Postgres Enterprise Manager Client in Advanced Server) to manually correct the rows in the publication tables across all primary nodes without replicating these changes. Use the database user with `session_replication_role` set to `replica` for this purpose.
- **Correction using new transactions.** Rerun your application on one primary node to create new transactions that can replicate to all other primary nodes. Use this method after you ensure that all publication tables are in a consistent state across all primary nodes.
- **Correction using shadow table transactions.** Force the synchronization of transactions already recorded in the shadow tables. Use this method if many shadow table transactions need to be applied and it's simpler to force the synchronization of these transactions rather than reissuing the transactions from your application.

Example replication environment

In the examples that follow, the following replication environment is used.

- A three-node multi-master replication system is established. The primary node names are `MMRnode_a` (the primary definition node and the controller database), `MMRnode_b` , and `MMRnode_c` .
- The publication is named `emp_pub` and uses dept and emp tables.
- The conflict used to show the first two conflict resolution methods is a uniqueness conflict occurring on the dept table on primary key column `deptno` on value 50. This conflict resulted from the `INSERT` statements shown by the following:

On `MMRnode_a` , the following statement is run:

```
INSERT INTO dept VALUES (50, 'FINANCE',
'CHICAGO');
```

On `MMRnode_b` , the following statement is run:

```
INSERT INTO dept VALUES (50, 'MARKETING', 'LOS
ANGELES');
```

A synchronization replication is then performed.

The following is the content of table dept on `MMRnode_a` :

```
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | FINANCE    | CHICAGO
(5 rows)
```

The following is the content of table dept on `MMRnode_b` :

```
MMRnode_b=# SELECT * FROM dept;
```

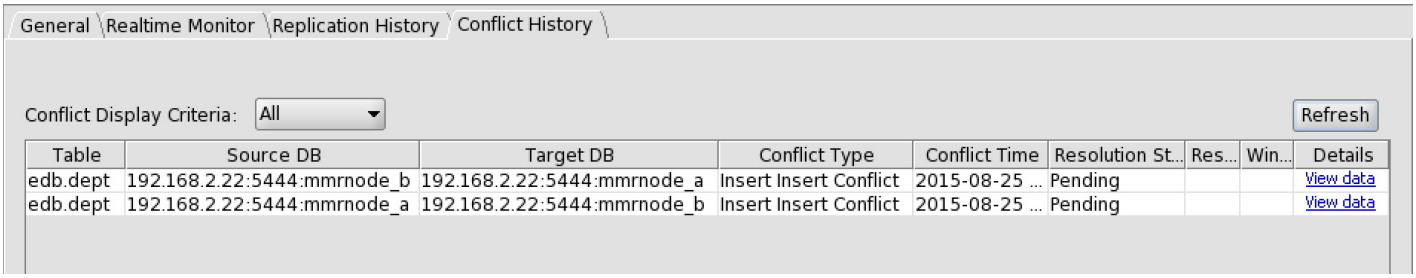
```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

The following is the content of table dept on `MMRnode_c` :

```
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

The **Conflict History** tab shows the following entry:



The following is the output from the SQL query described in Finding conflicts.

```
-[ RECORD 1 ]-------+-----------------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_a
src_rrep_sync_id    | 2
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode_b
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
-[ RECORD 2 ]-------+-----------------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_b
src_rrep_sync_id    | 1
target_db_host      | 192.168.2.22
target_db_port      | 5444
```

```
target_db_name      | MMRnode_a
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
```

Manual publication table correction

The first step in all manual conflict resolutions is to ensure all publication tables are consistent across all primary nodes. That is, all corresponding tables have the same rows with the same column values.

After that, you can then reapply transactions that failed to replicate successfully.

Using the example from Correction strategies, the inconsistencies are the following:

- Primary nodes `MMRnode_a` and `MMRnode_b` each contain a row with primary key value `50`, but the other column values in the row are different.
- Primary node `MMRnode_c` doesn't have a row with primary key value `50`.

Assuming that the correct state of the dept table is the one in `MMRnode_b`, the following options are available to correct the state of all primary nodes:

- Manually correct the dept table in `MMRnode_a` and `MMRnode_c`. That is, update the row in `MMRnode_a` so it has the correct values, and insert the missing row in `MMRnode_c`. The dept table on all nodes is now consistent and up to date.
- Manually delete the row with primary key value `50` from the table on both `MMRnode_a` and `MMRnode_b`. This brings the dept table on all primary nodes back to a prior, consistent state. Then, with the multi-master replication system running, perform the insert transaction again using the correct column values on any one of the primary nodes.
- Manually delete the incorrect row with primary key value `50` from the table on `MMRnode_a`. Leave the correct row in the table in `MMRnode_b`. This simulates the state where the correct transaction was run on `MMRnode_b`, is recorded in the shadow table, but has not yet been replicated, and the incorrect transaction was never run on `MMRnode_a`. Update the shadow table entry in `MMRnode_a` to indicate that it is discarded and to ensure it isn't included in any future synchronizations. Update the metadata for the shadow table entry in `MMRnode_b` to force its inclusion in the next synchronization. Perform a synchronization replication so the accepted shadow table entry in `MMRnode_b` is replicated to `MMRnode_a` and `MMRnode_c`.

After the publication table rows are corrected, update the appropriate control schema table in the publication database currently designated as the controller database to indicate that the conflict has been resolved.

Step 1

Manually correct the rows in the publication tables with `session_replication_role` set to `replica`. On `MMRnode_a`, correct the erroneous row:

```
MMRnode_a=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
replica
(1 row)
```

```
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
```

```
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | FINANCE    | CHICAGO
(5 rows)
```

```
MMRnode_a=# UPDATE dept SET dname = 'MARKETING', loc = 'LOS ANGELES' WHERE deptno =
50;
UPDATE 1
MMRnode_a=# SELECT * FROM dept ORDER BY deptno;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_c`, insert the missing row:

```
MMRnode_c=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 replica
(1 row)
```

```
MMRnode_c=# INSERT INTO dept VALUES (50, 'MARKETING', 'LOS
ANGELES');
INSERT 0 1
MMRnode_c=# SELECT * FROM dept ORDER BY deptno;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

The dept table on `MMRnode_a` and `MMRnode_c` now match the content of the table on `MMRnode_b`:

```
MMRnode_b=# SELECT * FROM dept ORDER BY deptno;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

Step 2

Update the shadow table entries for the conflicting transactions in the primary nodes to indicate that the conflict was resolved. In each primary node where a transaction occurred that's involved in the conflict, inspect the shadow table for the publication table in question. Shadow tables are located in each primary node in schema `_edb_replicator_pub`. Shadow tables follow the naming convention `rrst_schema_table` where `schema` is the name of the schema containing the publication table and `table` is the name of the publication table.

Note the following points regarding shadow tables:

- A row in a shadow table corresponds to an `INSERT, UPDATE`, or `DELETE` statement that's applied to the corresponding publication tables in the other primary nodes. A shadow table row doesn't necessarily correspond to the SQL statement issued by the user application. For example, a SQL statement issued by a user application that includes a `WHERE` clause using a range such as greater than or less than results in multiple, individual entries in the shadow table for each row in the result set of the application's SQL statement.
- The primary key of a shadow table is a program-generated positive integer in `column rrep_sync_id`. The `rrep_sync_id` values are unique among all shadow tables in a given primary node. Therefore, the `rrep_sync_id` values for conflicting transactions might not have the same value across primary nodes. This depends on how many prior transactions were recorded in the shadow tables of each primary node.
- A shadow table entry for a transaction involved in a conflict that wasn't yet resolved contains a value of `P` (pending) in column `rrep_tx_conflict_status`. If a transaction isn't involved in a conflict, this column is set to null. (Most shadow table entries have null in this column.) If a transaction was involved in a conflict that was resolved automatically by the publication server, and this transaction was accepted as correct, this column contains `C` (complete/accepted). If a transaction was involved in a conflict that was resolved automatically, and this transaction was deemed incorrect, this column contains `D` (discarded).

To find the shadow table entries involved in a conflict, use the **Conflict History** tab in the Replication Server console or the SQL query described in Finding conflicts and shown by the following output:

```
-[ RECORD 1 ]-------+-----------------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_a
src_rrep_sync_id    | 2
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode_b
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
-[ RECORD 2 ]-------+-----------------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_b
src_rrep_sync_id    | 1
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode_a
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
```

You can then query the shadow table in the desired primary node on its `rrep_sync_id` value.

The following query is performed on the shadow table for the dept table in `MMRnode_a` on `rrep_sync_id` value 2 obtained from field `src_rrep_sync_id` of RECORD 1 in the preceding output.

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.rrst_edb_dept WHERE rrep_sync_id = 2;
```

```
-[ RECORD 1 ]-----------+--------------------------
rrep_sync_id            | 2
rrep_common_id          |
rrep_operation_type     | I
rrep_tx_timestamp       | 25-AUG-15 11:39:35.590648
deptno                  | 50
dname                   | FINANCE
loc                     | CHICAGO
rrep_old_deptno         |
rrep_old_dname          |
rrep_old_loc            |
rrep_tx_conflict_status | P
```

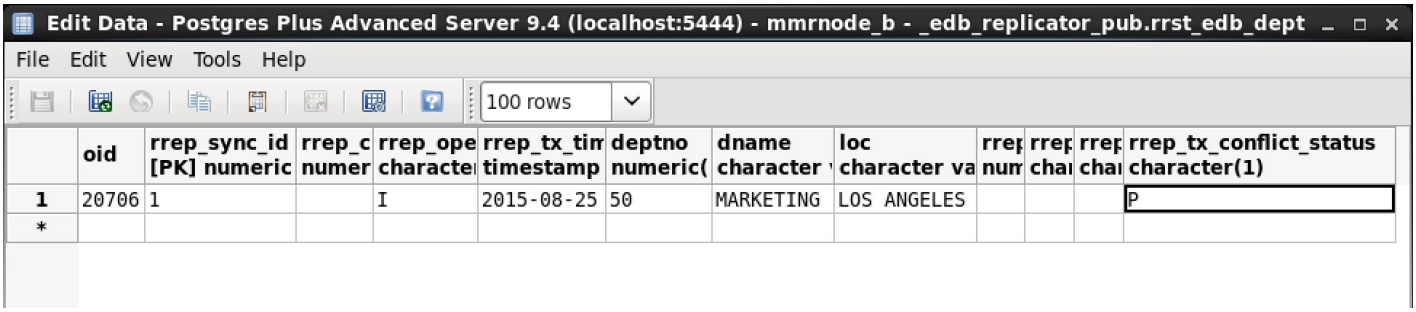
A similar query can locate the pending shadow table entry in `MMRnode_b` by querying on the key value obtained from field `src_rep_sync_id: of RECORD 2:`

```
MMRnode_b=# SELECT * FROM _edb_replicator_pub.rrst_edb_dept WHERE rrep_sync_id = 1;
```

```
-[ RECORD 1 ]-----------+--------------------------
rrep_sync_id            | 1
rrep_common_id          |
rrep_operation_type     | I
rrep_tx_timestamp       | 25-AUG-15 11:39:57.980469
deptno                  | 50
dname                   | MARKETING
loc                     | LOS ANGELES
rrep_old_deptno         |
rrep_old_dname          |
rrep_old_loc            |
rrep_tx_conflict_status | P
```

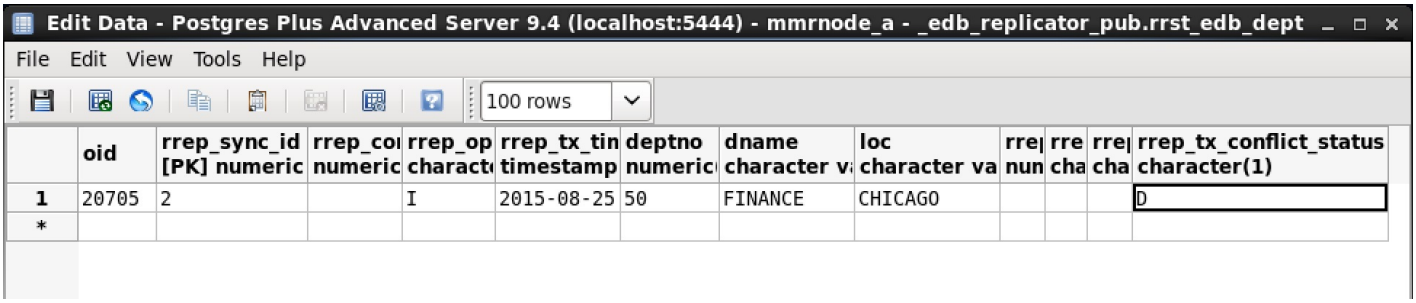
> **Note**
>
> To be certain no pending transactions are overlooked, examine the shadow tables in all primary nodes that might be involved in the conflict, and search for entries where `rrep_tx_conflict_status` is set to `P`.

The following shows the `rrep_tx_conflict_status` column marked `P` (pending) in the Postgres Enterprise Manager Client.



Modify column `rrep_tx_conflict_status` by changing the value to `D` (discarded) to show that the pending conflict was resolved. A value of `D` also ensures that the shadow table entry isn't replicated during any future synchronization replications.

Make this change to the shadow tables in both `MMRnode_a` and `MMRnode_b`.



Be sure to qualify the row with the correct `rrep_sync_id` value if you perform the update using a SQL statement such as in the following:

```
UPDATE _edb_replicator_pub.rrst_edb_dept SET rrep_tx_conflict_status = 'D' WHERE rrep_sync_id = 1;
```

There's no shadow table entry in `MMRnode_c`, since an insert transaction wasn't performed in that primary node by the application.

Step 3

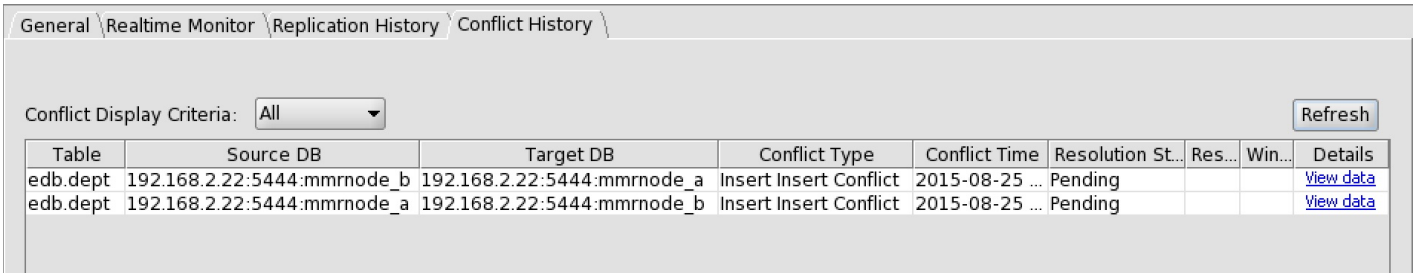
In the control schema of the publication database currently designated as the controller database, modify the entries in the `xdb_conflicts table` to indicate the conflict was resolved. Table `xdb_conflicts` is located in schema `_edb_replicator_pub`.

> **Note**
>
> The entries in table xdb_conflicts affect only the data that appears in the**Conflict History** tab and the SQL query described in Finding conflicts. Changing entries in `xdb_conflicts` has no effect on future replication operations but provides a way to keep a record of how past conflicts were resolved.

Note the following points regarding the `xdb_conflicts` table:

- A row in the `xdb_conflicts` table appears as an entry in the **Conflict History** tab.
- The primary key of the `xdb_conflicts table` is made up of columns `src_db_id`, `target_db_id`, `src_rrep_sync_id`, and `target_rrep_sync_id`. Column `src_db_id` contains a unique identifier for the primary node in which a transaction occurred that results in a conflict when replicated to the primary node identified by `target_db_id`. `src_rrep_sync_id` is the shadow table identifier of the transaction on the source primary node involved in the conflict, while `target_rrep_sync_id` is the shadow table identifier of the transaction on the target primary node involved in the conflict. For uniqueness (insert/insert) conflicts, the `target_rrep_sync_id` value is always set to `0`. For a given uniqueness conflict, there are two entries in the `xdb_conflicts table`. The `src_rrep_sync_id` value in each of the two entries corresponds to the shadow table identifiers. One is for the shadow table identifier associated with the source primary node. The other is for the shadow table identifier associated with the target primary node.
- Table `xdb_pub_database` in the control schema associates the database identifiers `src_db_id` and `target_db_id` with the primary node attributes such as the database name, IP address, and port.
- Column `table_id` is the identifier of the publication table on which the conflict occurred. Association of the `table_id` value with the publication table attributes such as its name, schema, and shadow table is found in each primary node in `_edb_replicator_pub.rrep_tables`.
- For uniqueness (insert/insert) conflicts only, column `pk_value` contains text indicating the primary key value that resulted in the conflict. The text is formatted as `column_name=value`. If the primary key is made up of two or more columns, each column and value pair is separated by the keyword AND, such as `column_1=value_1 AND column_2=value_2`. This provides the primary key of the row in the publication table designated by `table_id` that resulted in the conflict. Only uniqueness (insert/insert) conflicts contain the `column_name=value` text in the `pk_value` column. The `pk_value` column is null for all other conflict types (that is, update/update, delete/update, update/delete, and delete/delete conflicts).
- Column `resolution_status` indicates the status of the conflict. Possible values are `P` (pending) or `C` (completed, that is the conflict was resolved). This status appears in the Resolution Status column of the **Conflict History** tab.
- You can use the column `win_db_id` to record the database identifier of the primary node that contains the winning (accepted) transaction. This information appears in the Winning DB column of the **Conflict History** tab.
- You can use column `win_rrep_sync_id` to record the shadow table identifier of the winning transaction.

The following shows the **Conflict History** tab prior to updating the `xdb_conflicts` table.



You can find the conflict entry for synchronization from `MMRnode_a` to `MMRnode_b` in `xdb_conflicts` with the following query for this example:

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.xdb_conflicts
MMRnode_a-#   WHERE src_db_id = 1
MMRnode_a-#     AND target_db_id = 4
MMRnode_a-#     AND src_rrep_sync_id =
2
MMRnode_a-#     AND target_rrep_sync_id =
0;
```

```
-[ RECORD 1 ]-------+-----------------------------------------------------------------
src_db_id           | 1
target_db_id        | 4
src_rrep_sync_id    | 2
target_rrep_sync_id | 0
table_id            | 31
conflict_time       | 25-AUG-15 10:40:23.685738
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
win_rrep_sync_id    | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
 pk_value            | deptno=50
```

You can find the conflict entry for synchronization from `MMRnode_b` to `MMRnode_a` in xdb_conflicts with the following query for this example:

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.xdb_conflicts
MMRnode_a-#   WHERE src_db_id = 4
MMRnode_a-#     AND target_db_id = 1
MMRnode_a-#     AND src_rrep_sync_id =
1
MMRnode_a-#     AND target_rrep_sync_id =
0;
```

```
-[ RECORD 1 ]-------+-----------------------------------------------------------------
src_db_id           | 4
target_db_id        | 1
src_rrep_sync_id    | 1
target_rrep_sync_id | 0
table_id            | 31
conflict_time       | 25-AUG-15 10:40:23.726889
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
```

```
win_rrep_sync_id    | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
pk_value            | deptno=50
```

For uniqueness (insert/insert) conflicts only, you can use the following query to display both of the preceding entries:

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.xdb_conflicts
MMRnode_a-#   WHERE pk_value =
'deptno=50'
MMRnode_a-#     AND conflict_type = 'II'
MMRnode_a-#     AND resolution_status = 'P';
```

```
-[ RECORD 1 ]-------+-----------------------------------------------------------------
src_db_id           | 1
target_db_id        | 4
src_rrep_sync_id    | 2
target_rrep_sync_id | 0
table_id            | 31
conflict_time       | 25-AUG-15 10:40:23.685738
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
win_rrep_sync_id    | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
pk_value            | deptno=50
-[ RECORD 2 ]-------+-----------------------------------------------------------------
src_db_id           | 4
target_db_id        | 1
src_rrep_sync_id    | 1
target_rrep_sync_id | 0
   table_id            | 31
   conflict_time       | 25-AUG-15 10:40:23.726889
   resolution_status   | P
   resolution_strategy |
   resolution_time     |
   alert_status        |
   conflict_type       | II
   win_db_id           | 0
   win_rrep_sync_id    | 0
   notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                       |   Detail: Key (deptno)=(50) already exists.
   pk_value            | deptno=50
```

These entries appear in the Postgres Enterprise Manager Client.

Change the value in column `resolution_status` from `P` (pending) to `C` (completed) to indicate this conflict was resolved. The value in `winning_db_id` changes to `4` to indicate primary node `MMRnode_b` contains the winning transaction. The value in `winning_rrep_sync_id` changes to the value of `rrep_sync_id` for the shadow table entry of the transaction in `MMRnode_b`, since this is the one deemed correct.

The SQL statement to perform this update for the `MMRnode_a` to the `MMRnode_b` synchronization conflict is the following:

```
UPDATE _edb_replicator_pub.xdb_conflicts SET
  resolution_status = 'C',
  win_db_id = 4,
```

```
  win_rrep_sync_id =
1
WHERE src_db_id = 1
  AND target_db_id = 4
  AND src_rrep_sync_id =
2
  AND target_rrep_sync_id =
0;
```

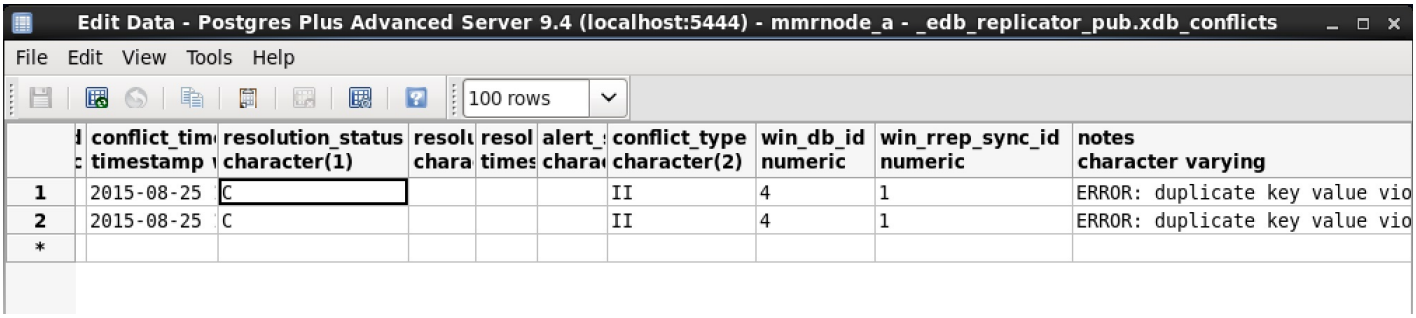
The SQL statement to perform this update for the `MMRnode_b` to the `MMRnode_a` synchronization conflict is the following:

```
UPDATE _edb_replicator_pub.xdb_conflicts SET
  resolution_status = 'C',
  win_db_id = 4,
  win_rrep_sync_id =
1
WHERE src_db_id = 4
  AND target_db_id = 1
  AND src_rrep_sync_id =
1
  AND target_rrep_sync_id =
0;
```

For uniqueness (insert/insert) conflicts only, you can use the following SQL statement to update both of the preceding entries at the same time:

```
UPDATE _edb_replicator_pub.xdb_conflicts SET
  resolution_status = 'C',
  win_db_id = 4,
  win_rrep_sync_id =
1
WHERE pk_value =
'deptno=50'
  AND conflict_type = 'II'
  AND resolution_status = 'P';
```

The following are the updated xdb_conflicts entries:



When viewed in the **Conflict History** tab, the entries now show Resolved instead of Pending in the Resolution Status column, and the Winning DB column shows the address of primary node `MMRnode_b` .

Correction using new transactions

You can bring all the publication tables to a consistent state by removing any changes caused by the conflicting transactions. Then issue new, corrected transactions at one primary node, which you allow the multi-master replication system to synchronize to all other primary nodes.

Referring back to the uniqueness conflict on the dept table, instead of correcting the erroneous row and inserting the row into the primary node where it is missing as described in Manual publication table correction, you can delete the conflicting rows from all primary nodes and then insert the correct row in one primary node and let the multi-master replication system synchronize the correct row to all primary nodes.

Step 1

Manually delete the inserted row from the publication tables in all primary nodes with `session_replication_role` set to `replica`.

On `MMRnode_a`, delete the erroneous row:

```
MMRnode_a=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 replica
(1 row)


 MMRnode_a=# SELECT * FROM dept;
  deptno |   dname    |   loc
 --------+------------+----------
      10 | ACCOUNTING | NEW YORK
      20 | RESEARCH   | DALLAS
      30 | SALES      | CHICAGO
      40 | OPERATIONS | BOSTON
      50 | FINANCE    | CHICAGO
 (5 rows)
```

```
MMRnode_a=# DELETE FROM dept WHERE deptno =
50;
DELETE 1
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

On `MMRnode_b`, delete the row even though the transaction created the correct result:

```
MMRnode_b=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 replica
(1 row)
```

```
MMRnode_b=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

```
MMRnode_b=# DELETE FROM dept WHERE deptno =
50;
DELETE 1
MMRnode_b=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
   10 | ACCOUNTING | NEW YORK
   20 | RESEARCH   | DALLAS
   30 | SALES      | CHICAGO
   40 | OPERATIONS | BOSTON
(4 rows)
```

On `MMRnode_c`, no changes are required as the conflicting transaction didn't insert a new row into the table on this node:

```
MMRnode_c=# SET search_path TO
edb;
SET
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

Step 2

Rerun the transaction on one primary node with the multi-master replication system running and with `session_replication_role` set to the default (`origin`).

For this example, the correct `INSERT` statement is executed on `MMRnode_a`:

```
MMRnode_a=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 origin
(1 row)
```

```
MMRnode_a=# INSERT INTO dept VALUES (50, 'MARKETING', 'LOS
ANGELES');
INSERT 0 1
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

Step 3

Perform synchronization replication.

The same rows now appear in the publication table on all primary nodes. On `MMRnode_a`:

```
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_b`:

```
MMRnode_b=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_c`:

```
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

Step 4

Update the shadow table entries for the conflicting transactions in the primary nodes to indicate that the conflict was resolved as in Step 2 ofManual publication table correction.

Change the `rrep_tx_conflict_status` column from `P` (pending) to `D` (discarded) on all primary nodes.

> **Note**
>
> The second entry for the accepted transaction you ran in Step 2, where `rrep_tx_conflict_status` is set to null, indicates there was no conflict.

There's no shadow table entry in `MMRnode_c`, since an insert transaction wasn't performed in that primary node by the application.

Step 5

In the control schema of the publication database currently designated as the controller database, modify the entries in the `xdb_conflicts` table to indicate the conflict was resolved as in Step 3 of Manual publication table correction.

Correction using shadow table transactions

You can bring all publication tables to a consistent state is by removing changes caused by the conflicting transactions. You then modify the publication table's metadata in such a way that the next synchronization results in the replication of transactions already stored in the shadow tables.

Such transactions might not have successfully replicated to all the other primary nodes in a prior synchronization for various reasons.

The following is an example of such a case:

- Applications on two primary nodes insert rows with the same primary key value. This results in a uniqueness conflict when synchronization replication occurs.
- Following the insert on one primary node, the application continues to apply updates to the newly inserted row. These updates are successfully applied to the row on this primary node and are recorded in the shadow table on this node.
- Synchronization replication is performed.
- Since there is a uniqueness conflict, the rows with the conflicting primary key value aren't replicated into the publication tables on the other primary nodes.
- However, the conflicting row on the primary node that wasn't directly updated receives those update transactions by the replication, resulting in possibly inconsistent, updated rows on the two primary nodes.

Two options are:

- Manually insert the missing row into the other primary nodes and manually change the incorrect row.
- Rerun the application to reapply the correct insert and updates.

However, the following option provides a way to reapply the transactions already recorded in the shadow table of the winning primary node.

The example used to illustrate this method is based upon the following transactions on the emp table.

In `MMRnode_b`, the following row is inserted:

```
INSERT INTO emp (empno,ename,job,deptno) VALUES
(9001,'SMITH','ANALYST',20);
```

In `MMRnode_c`, the following row is inserted with the same primary key value 9001 in the `empno` column:

```
INSERT INTO emp (empno,ename,job,deptno) VALUES
(9001,'JONES','SALESMAN',30);
In MMRnode_c, this is followed by a series of updates to the newly inserted
row:
UPDATE emp SET mgr = 7698 WHERE empno =
9001;
UPDATE emp SET sal = 9500 WHERE empno =
9001;
UPDATE emp SET comm = 5000 WHERE empno =
9001;
```

Synchronization replication is performed. The resulting content of the `emp` table is as follows:

On `MMRnode_a` the conflicting row has not been replicated:

```
MMRnode_a=# SELECT * FROM emp;
```

```
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00  |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00  | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00  | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00  | 2975.00 |         |     20
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00  | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00  | 2850.00 |         |     30
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00  | 2450.00 |         |     10
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00  | 3000.00 |         |     20
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00  | 5000.00 |         |     10
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00  | 1500.00 |    0.00 |     30
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00  | 1100.00 |         |     20
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00  |  950.00 |         |     30
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00  | 3000.00 |         |     20
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00  | 1300.00 |         |     10
(14 rows)
```

On `MMRnode_b` the conflicting row inserted on this node remains, but is updated with the transactions replicated from `MMRnode_c` :

```
MMRnode_b=# SELECT * FROM emp;
```

```
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00  |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00  | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00  | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00  | 2975.00 |         |     20
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00  | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00  | 2850.00 |         |     30
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00  | 2450.00 |         |     10
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00  | 3000.00 |         |     20
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00  | 5000.00 |         |     10
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00  | 1500.00 |    0.00 |     30
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00  | 1100.00 |         |     20
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00  |  950.00 |         |     30
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00  | 3000.00 |         |     20
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00  | 1300.00 |         |     10
  9001 | SMITH  | ANALYST   | 7698 |                     | 9500.00 | 5000.00 |     20
(15 rows)
```

On `MMRnode_c` the conflicting row inserted on this node remains along with the updates performed on this node:

```
MMRnode_c=# SELECT * FROM emp;
```

```
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00  |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00  | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00  | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00  | 2975.00 |         |     20
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00  | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00  | 2850.00 |         |     30
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00  | 2450.00 |         |     10
```

```
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00 | 3000.00 |         |     20
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00 | 5000.00 |         |     10
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00 | 1500.00 |    0.00 |     30
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00 | 1100.00 |         |     20
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00 |  950.00 |         |     30
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00 | 3000.00 |         |     20
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00 | 1300.00 |         |     10
  9001 | JONES  | SALESMAN  | 7698 |                    | 9500.00 | 5000.00 |     30
(15 rows)
```

In this example, it is assumed that the desired, correct row is on `MMRnode_c`.

The following are the steps to reproduce the correct row, currently on `MMRnode_c`, to the other primary nodes by synchronizing the shadow table entries that resulted from the original insert and updates to this row on `MMRnode_c`.

Step 1

Manually delete the inserted row from the publication tables on all primary nodes except for `MMRnode_c`, which has the correct row. Be sure `session_replication_role` is set to `replica`.

On `MMRnode_a`, this row does not exist:

```
MMRnode_a=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-------+-------+-----+-----+----------+-----+------+--------
(0 rows)
```

On `MMRnode_b`, delete the erroneous row:

```
MMRnode_a=# SHOW session_replication_role;
```

```
 session_replication_role
--------------------------
 replica
(1 row)
```

```
MMRnode_b=# DELETE FROM emp WHERE empno =
9001;
DELETE 1
```

On `MMRnode_c`, leave the correct, accepted row intact:

```
MMRnode_c=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  | hiredate |   sal   |  comm   | deptno
-------+-------+----------+------+----------+---------+---------+--------
  9001 | JONES | SALESMAN | 7698 |          | 9500.00 | 5000.00 |     30
(1 row)
```

Step 2

On the primary nodes containing the conflicting row to discard, mark the shadow table entry for that row as discarded. This mark indicates the conflict on this row was resolved and ensures this shadow table entry isn't replicated in the future.

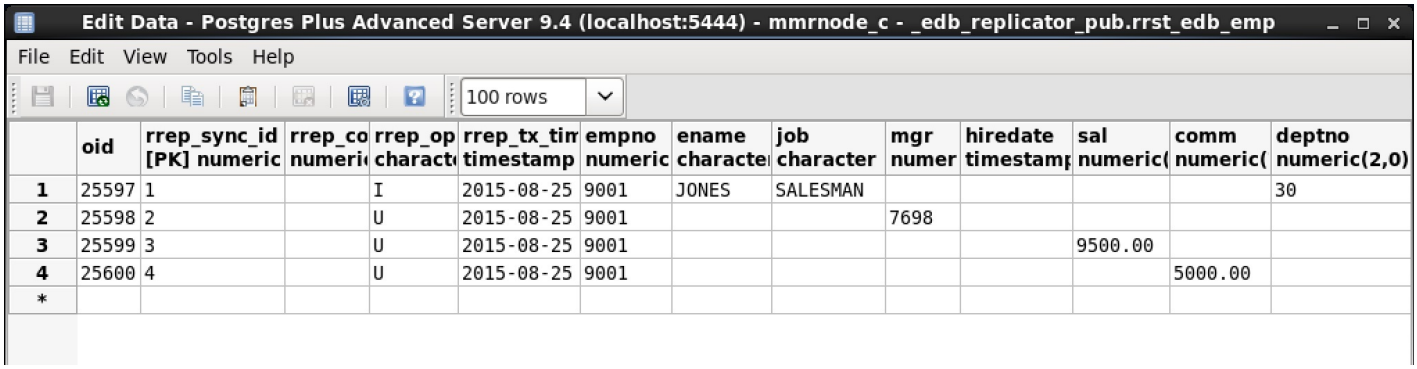
Change the `rrep_tx_conflict_status` column from `P` (pending) to `D` (discarded) on the losing node, `MMRnode_b`.

Step 3

On winning node `MMRnode_c`, inspect the shadow table for the emp publication table.

The objective is to use the shadow table entries for the insert and three update transactions that were previously run on this node to replicate to the other primary nodes during the next synchronization.

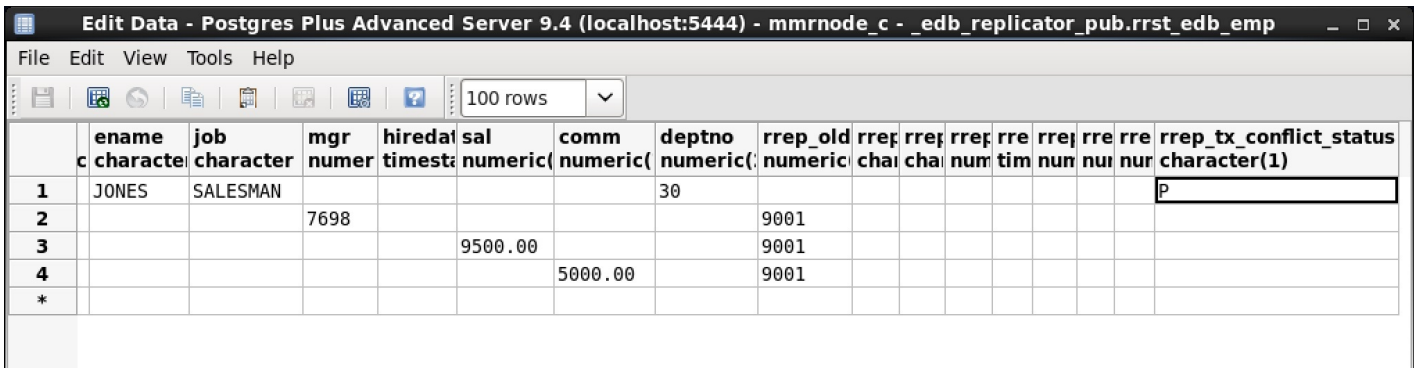
The left-most columns of the shadow table appear as follows:



	oid	rrep_sync_id [PK] numeric	rrep_co numeri	rrep_op charact	rrep_tx_tir timestamp	empno numeric	ename characte	job character	mgr numer	hiredate timestamp	sal numeric(	comm numeric(	deptno numeric(2,0)
1	25597	1		I	2015-08-25	9001	JONES	SALESMAN					30
2	25598	2		U	2015-08-25	9001			7698				
3	25599	3		U	2015-08-25	9001					9500.00		
4	25600	4		U	2015-08-25	9001						5000.00	
*													

Make note of the `rrep_sync_id` values for these four entries, which are `1`, `2`, `3`, and `4` in this example.

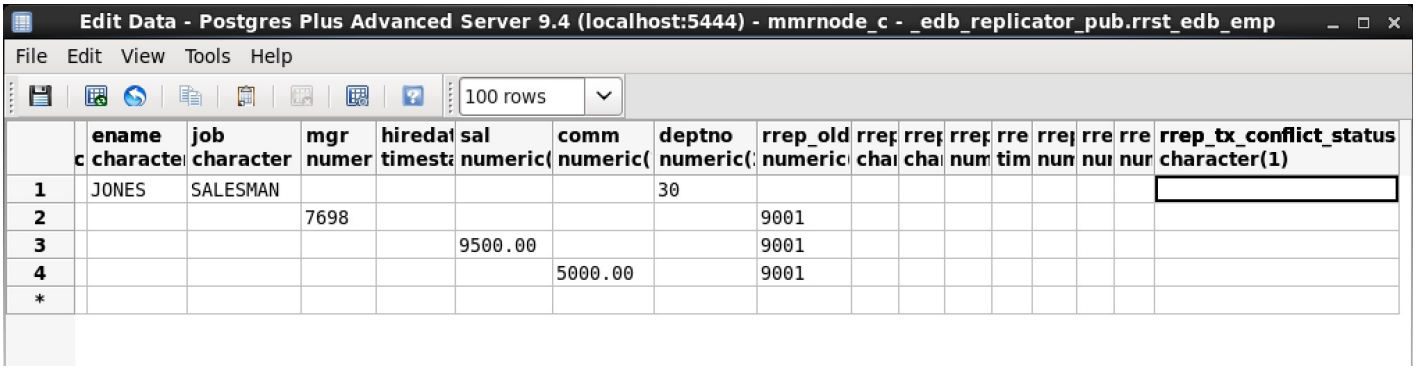
The following shows the right-most columns of the shadow table from the figure. Note the contents of column `rrep_tx_conflict_status` furthest to the right.



	c	ename characte	job character	mgr numer	hireda timesta	sal numeric(	comm numeric(	deptno numeric(	rrep_old numeric	rrep cha	rrep cha	rrep num	rre tim	rrep num	rre nu	rre nur	rrep_tx_conflict_status character(1)
1		JONES	SALESMAN					30									P
2				7698					9001								
3						9500.00			9001								
4							5000.00		9001								
*																	

Make sure the `rrep_tx_conflict_status` column is null for these four entries. In this case, for the insert transaction, you need to change the `P` (pending) value to null.

The resulting change for the `rrep_tx_conflict_status` column in the shadow table on `MMRnode_c` is shown by the following:



Step 4

To replicate these four shadow table entries during the next synchronization, you must add one or more entries to the control schema table `_edb_replicator_pub.rrep_MMR_txset` on `MMRnode_c` to indicate pending status for synchronization to the target primary nodes (`MMRnode_a` and `MMRnode_b`) of the four shadow table entries. These shadow table entries are identified by the `rrep_sync_id` values of `1` , `2` , `3` , and `4` noted in Step 3.

First, you must identify the `pub_id` and target `db_id` values to associate with the pending transactions. To do so, invoke the following query, substituting the `rrep_sync_id` values for `sync_id` in the query:

```
SELECT pub_id, db_id AS target_db_id
  FROM
_edb_replicator_pub.rrep_MMR_txset
  WHERE start_rrep_sync_id <= sync_id
    AND end_rrep_sync_id >=
sync_id;
```

In this example, there are four values to substitute for `sync_id` , which are `1` , `2` , `3` , and `4` .

The results are the following:

```
MMRnode_c=# SELECT pub_id, db_id AS target_db_id
MMRnode_c-#   FROM
_edb_replicator_pub.rrep_MMR_txset
MMRnode_c-#   WHERE start_rrep_sync_id <= 1 AND end_rrep_sync_id >=
1;
```

```
 pub_id | target_db_id
--------+--------------
      3 |            1
      3 |            4
(2 rows)
```

```
MMRnode_c=# SELECT pub_id, db_id AS target_db_id
MMRnode_c-#   FROM
_edb_replicator_pub.rrep_MMR_txset
MMRnode_c-#   WHERE start_rrep_sync_id <= 2 AND end_rrep_sync_id >=
2;
```

```
 pub_id | target_db_id
--------+--------------
      3 |            1
      3 |            4
(2 rows)
```

```
MMRnode_c=# SELECT pub_id, db_id AS target_db_id
MMRnode_c-#   FROM
_edb_replicator_pub.rrep_MMR_txset
MMRnode_c-#   WHERE start_rrep_sync_id <= 3 AND end_rrep_sync_id >=
3;
```

```
 pub_id | target_db_id
--------+--------------
      3 |            1
      3 |            4
(2 rows)
```

```
MMRnode_c=# SELECT pub_id, db_id AS target_db_id
MMRnode_c-#   FROM
_edb_replicator_pub.rrep_MMR_txset
MMRnode_c-#   WHERE start_rrep_sync_id <= 4 AND end_rrep_sync_id >=
4;
```

```
 pub_id | target_db_id
--------+--------------
      3 |            1
      3 |            4
(2 rows)
```

The results indicate that the previously executed synchronization that attempted to apply the shadow table transactions identified by the `rrep_sync_id` values of `1`, `2`, `3`, and `4` were all for the publication identified by `pub_id` of `3`. The target primary nodes were identified by `db_id` of `1` (for `MMRnode_a`) and `db_id` of `4` (for `MMRnode_b`).

Thus, you must insert at least two entries into the control schema table `_edb_replicator_pub.rrep_MMR_txset` on `MMRnode_c`. At least one entry is required for the target `db_id` of `1` and at least one entry for the target `db_id` of `4`.

Each entry in `_edb_replicator_pub.rrep_MMR_txset` consists of a range of `rrep_sync_id` values (identified by columns `start_rrep_sync_id` and `end_rrep_sync_id`). The desired shadow table `rrep_sync_id` values happen to be contiguous (1 through 4). Thus a single entry can encompass the four `rrep_sync_id` values for a single target database.

In this example, you can add a total of two entries to `_edb_replicator_pub.rrep_MMR_txset`, one for each target database.

> **Note**
>
> When multiple, noncontiguous `rrep_sync_id` values are required for synchronization (for example, `1`, `2`, `5`, and `6`), multiple entries are required for each target database. The entries specify `rrep_sync_id` ranges to collectively cover all of the noncontiguous values but omitting `rrep_sync_id` values not included in the synchronization (for example, one entry for 1 through 2 and a second entry for 5 through 6).

Step 5

Insert the entries into the `_edb_replicator_pub.rrep_MMR_txset` control schema table identified in the preceding step.

The two `INSERT` statements invoked on `MMRnode_c` are the following:

```
INSERT INTO _edb_replicator_pub.rrep_MMR_txset (set_id, pub_id, db_id, status, start_rrep_sync_id,
end_rrep_sync_id)

  values (nextval('_edb_replicator_pub.rrep_txset_seq'),3,1,'P',1,4);

INSERT INTO _edb_replicator_pub.rrep_MMR_txset (set_id, pub_id, db_id, status, start_rrep_sync_id,
end_rrep_sync_id)
  values (nextval('_edb_replicator_pub.rrep_txset_seq'),3,4,'P',1,4);
```

A query of the _edb_replicator_pub.rrep_MMR_txset metadata table displays the following:

```
MMRnode_c=# SELECT set_id, pub_id, db_id AS target_db_id, status,
MMRnode_c-#     start_rrep_sync_id,
end_rrep_sync_id
MMRnode_c-#   FROM _edb_replicator_pub.rrep_MMR_txset;
```

```
 set_id | pub_id | target_db_id | status | start_rrep_sync_id | end_rrep_sync_id
--------+--------+--------------+--------+--------------------+------------------
      1 |      3 |            1 | C      |                  1 |                4
      1 |      3 |            4 | C      |                  1 |                4
      2 |      3 |            1 | P      |                  1 |                4
      3 |      3 |            4 | P      |                  1 |                4
(4 rows)
```

There are now two new entries with pending status: one for target `db_id` 1, the other for target `db_id` 4. Both entries cover the `rrep_sync_id` range of 1 through 4.

The two entries with completed status are from the synchronization attempt that first produced the conflict.

Step 6

Perform synchronization replication.

The insert and three update transactions recorded in the `rrst_edb_emp` shadow table on `MMRnode_c` are replicated to the other primary nodes.

```
On MMRnode_a:
MMRnode_a=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  | hiredate |   sal   |  comm   | deptno
-------+-------+----------+------+----------+---------+---------+--------
  9001 | JONES | SALESMAN | 7698 |          | 9500.00 | 5000.00 |     30
(1 row)
```

```
On MMRnode_b:
MMRnode_b=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  | hiredate |   sal   |  comm   | deptno
-------+-------+----------+------+----------+---------+---------+--------
  9001 | JONES | SALESMAN | 7698 |          | 9500.00 | 5000.00 |     30
(1 row)
```

These rows now match the row created by the original transactions on `MMRnode_c`:

```
MMRnode_c=# SELECT * FROM emp WHERE empno =
9001;
```

```
 empno | ename |   job    | mgr  | hiredate |   sal   |  comm   | deptno
-------+-------+----------+------+----------+---------+---------+--------
  9001 | JONES | SALESMAN | 7698 |          | 9500.00 | 5000.00 |     30
(1 row)
```

Step 7

In the control schema of the publication database currently designated as the controller database, modify the entries in the xdb_conflicts table to indicate the conflict was resolved as in Step 3 of Manual publication table correction.

For a uniqueness (insert/insert) conflict only, the following query on the `xdb_conflicts` table in the controller database can display the conflicts:

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.xdb_conflicts
MMRnode_a-#   WHERE pk_value =
'empno=9001'
MMRnode_a-#     AND conflict_type = 'II'
MMRnode_a-#     AND resolution_status = 'P';
```

```
-[ RECORD 1 ]-------+-------------------------------------------------------------
src_db_id           | 4
target_db_id        | 56
src_rrep_sync_id    | 1
target_rrep_sync_id | 0
table_id            | 32
conflict_time       | 25-AUG-15 15:27:20.928679
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
win_rrep_sync_id    | 0
notes               | ERROR: duplicate key value violates unique constraint "emp_pk"
                    |   Detail: Key (empno)=(9001) already exists.
pk_value            | empno=9001
-[ RECORD 2 ]-------+-------------------------------------------------------------
src_db_id           | 56
target_db_id        | 4
src_rrep_sync_id    | 1
target_rrep_sync_id | 0
table_id            | 32
conflict_time       | 25-AUG-15 15:27:20.970959
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
win_rrep_sync_id    | 0
notes               | ERROR: duplicate key value violates unique constraint "emp_pk"
                    |   Detail: Key (empno)=(9001) already exists.
pk_value            | empno=9001
```

The following SQL statement changes the value in column `resolution_status` from `P` (pending) to `C` (completed) to indicate this conflict was resolved. The value in `winning_db_id` changes to `56` to indicate primary node `MMRnode_c` contains the winning transaction. The value in `winning_rrep_sync_id` is changed to the value of `rrep_sync_id` for the shadow table entry of the `INSERT` transaction in `MMRnode_c`, since this is the correct one.

```
UPDATE _edb_replicator_pub.xdb_conflicts SET
  resolution_status = 'C',
  win_db_id = 56,
  win_rrep_sync_id =
1
WHERE pk_value =
'empno=9001'
  AND conflict_type = 'II'
  AND resolution_status = 'P';
```

When viewed in the **Conflict History** tab, the entry now shows Resolved in the Resolution Status column, and the Winning DB column shows the address of primary node `MMRnode_c` .

9.6.10 Manual conflict resolution for the log-based method

> **Note**
>
> See Manual conflict resolution for the trigger-based method for information on manual conflict resolution for multi-master replication systems configured with the trigger-based method of synchronization replication.

As discussed in Conflict prevention - uniqueness case there's no built-in, automatic conflict-resolution strategy for the uniqueness (insert/insert) conflict. If a uniqueness conflict occurs, to resolve this conflict you must modify rows in the publication tables containing the conflict and modify rows in the control schema tables in the primary nodes.

Similarly, you must use manual correction for update/delete and delete/update conflicts. In addition, if the conflict resolution option is set to **Manual** (see Updating the conflict-resolution options) and a conflict occurs, you must also resolve this conflict manually.

9.6.10.1 Finding conflicts

You can find conflicts on the **Conflict History** tab as described in Viewing conflict history. In the **Conflict History** tab, select **Refresh** to show all the latest conflicts.

> **Note**
>
> The **View Data** link and Conflict Details window appear only for multi-master replication systems configured with the trigger-based method of synchronization replication. They aren't available for multi-master replication systems configured with the log-based method of synchronization replication.

The Source DB and Target DB columns provide the IP address and database names of the source and target primary nodes involved in the conflict.

You can get this same information from a SQL query. You can run the following query from a primary node to display information regarding pending (unresolved) conflicts:

```
 SELECT DISTINCT

conflict_type,
  t.table_name,
  pk_value,
  d1.db_host AS src_db_host,
  d1.db_port AS src_db_port,
  d1.db_name AS src_db_name,
  src_rrep_sync_id,
  d2.db_host AS target_db_host,
  d2.db_port AS target_db_port,
  d2.db_name AS target_db_name,
  target_rrep_sync_id,
  c.notes
FROM _edb_replicator_pub.xdb_conflicts
c
  JOIN _edb_replicator_pub.xdb_pub_database d1 ON c.src_db_id =
d1.pub_db_id
  JOIN _edb_replicator_pub.xdb_pub_database d2 ON c.target_db_id =
d2.pub_db_id
```

```
  JOIN _edb_replicator_pub.rrep_tables t ON c.table_id =
t.table_id
WHERE resolution_status = 'P'
ORDER BY t.table_name;
```

The following shows example output from the query:

```
 -[ RECORD 1 ]-------+-------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | edb
src_rrep_sync_id    | 41939160
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode
target_rrep_sync_id | 42289824
notes               |
-[ RECORD 2 ]-------+-------------
conflict_type       | DU
table_name          | emp
pk_value            | empno=9003
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | edb
src_rrep_sync_id    | 41940704
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode
target_rrep_sync_id | 42292848
notes               |
```

9.6.10.2 Conflict resolution concept for the log-based method

Manual conflict resolution typically requires modifying rows in one or more publication tables to correct erroneous entries. You can perform these changes using a utility such as `PSQL` or `pgAdmin` (Postgres Enterprise Manager Client in Advanced Server).

Usually, manual publication table corrections are isolated. Limit modifications to the publication tables you're directly changing. Don't replicate them to the other primary nodes as normally occurs in the multi-master replication system.

To prevent Replication Server from replicating changes to one or more publication tables during a synchronization operation, make the changes to the publication tables in a transaction block that includes a reference to an Replication Server control schema table. This reference causes Replication Server to skip the transaction block when performing a synchronization replication.

Not every Replication Server control schema table prevents this replication of a transaction block. Include the SQL `UPDATE` statement shown in the following transaction block to prevent replication of other publication table changes from appearing in the same transaction block:

```
 BEGIN;
UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp WHERE key = 'last_mcr_timestamp';

One or more SQL statements to correct publication tables

END;
```

When you execute such a transaction block in a primary node, inserting, updating, or deleting rows of any publication table in the transaction block aren't replicated to any other primary node at the next synchronization replication.

You can run as many such transaction blocks on any primary node as needed to change the publication table rows to resolve the conflicts. The resulting changes are isolated to the primary node on which the transaction block is run. Thus you can correct each primary node independently.

9.6.10.3 Correction strategies

Before you begin manual resolution correction, it's important to determine the extent of the inconsistencies that occurred in the publication tables across the primary nodes of the replication system.

The **Conflict History** tab and the SQL query described in Finding conflicts can help determine the source of an initial conflict.

However, once this conflict occurs, your replication system might have processed and replicated more transactions during that synchronization operation. Some of these later replications might have succeeded, but others might have failed or produced unexpected results because of the prior conflict. With a replication schedule in effect, more synchronization operations can occur, which can create more conflicts.

Therefore, when you discovered that a conflict occurred, we recommended that you stop the publication server. Use the stop option of the Linux scripts or Windows services described in Step 1 of Registering a publication server.

In this way, you can carefully analyze the content of the publication tables in question as well as any pending transactions in the shadow tables. This approach helps you to determine the best course of action without continued updates by the running replication system.

When analyzing your tables, you must determine which publication tables contain inconsistent rows across primary nodes (that is, missing rows on some primary nodes or rows with different column values for the same primary key on different primary nodes).

The general steps to resolving the problem following this analysis are:

1. Make the needed manual corrections to the rows in the publication tables across all primary nodes to get them into an initial, consistent state so each publication table has the same set of identical rows across primary nodes. This might be to a state before the conflicting transactions occurred, depending on what you determine is the easiest course of action for fully resolving the conflict.

2. Apply transactions (either from your application or from transaction blocks as defined inConflict resolution concept for the log-based method) so that all publication tables across all primary nodes are updated consistently according to the desired, expected result.

3. In the control schema, update certain indicators for the conflicting entries to show that these conflicts are resolved. This update changes the Resolution Status of these entries to Resolved in the **Conflict History** tab. These entries no longer appear in the SQL query described in Finding conflicts.

   Perform these updates to the control schema of the controller database. You can determine the currently designated controller database from the content of the Replication Server configuration file (see xDB replication configuration rile). The publication server ensures that the control schema changes made on the controller database are replicated to the control schemas of all publication databases to maintain metadata consistency across all publication databases.

4. Resume operating your replication system. Start the publication server and re-create the replication schedule if you were using one.

For accomplishing steps 1 and 2, use some combination of the following methods. The methods you use depends upon the state of your publication tables.

- **Manual publication table correction.** Use a utility such as `PSQL` or `pgAdmin` (Postgres Enterprise Manager Client in Advanced Server) to manually correct the rows in the publication tables across all primary nodes without replicating these changes. Apply these manual corrections in the transaction block described in Conflict resolution concept for the log-based method.
- **Correction using new transactions.** To create new transactions that you allow to replicate to all other primary nodes, rerun your application on one primary node. Use this method after you ensure that all publication tables are in a consistent state across all primary nodes.

In the description of these methods, the following replication environment is used.

- A three-node multi-master replication system is established. The primary node names are `MMRnode_a` (the primary definition node and the controller database), `MMRnode_b` , and `MMRnode_c` .
- The publication is named `emp_pub` and uses the dept and emp tables.
- The conflict used to show the conflict resolution methods is a uniqueness conflict occurring on the dept table on primary key column `deptno` on value 50. This conflict is a result of the `INSERT` statements shown by the following:

On `MMRnode_a` , the following statement is run:

```
INSERT INTO dept VALUES (50, 'FINANCE',
'CHICAGO');
On MMRnode_b, the following statement is run:
INSERT INTO dept VALUES (50, 'MARKETING', 'LOS
ANGELES');
```

A synchronization replication is then performed.

The following is the content of table dept on `MMRnode_a` :

```
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | FINANCE    | CHICAGO
(5 rows)
```

The following is the content of table dept on `MMRnode_b` :

```
MMRnode_b=# SELECT * FROM dept;
```

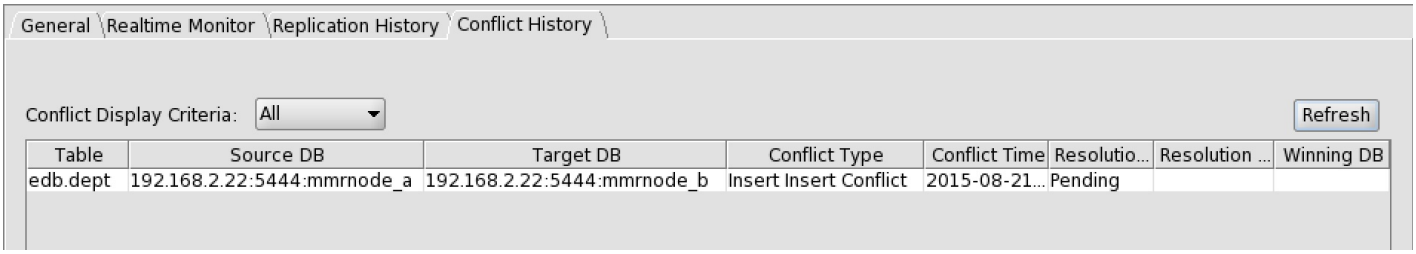
```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

The following is the content of table dept on `MMRnode_c` :

```
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

The **Conflict History** tab shows the following entry:



The following is the output from the SQL query described in Finding conflicts.

```
-[ RECORD 1 ]-------+-----------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_a
src_rrep_sync_id    | 2
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode_b
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
-[ RECORD 2 ]-------+-----------------------------------------------------------
conflict_type       | II
table_name          | dept
pk_value            | deptno=50
src_db_host         | 192.168.2.22
src_db_port         | 5444
src_db_name         | MMRnode_b
src_rrep_sync_id    | 1
target_db_host      | 192.168.2.22
target_db_port      | 5444
target_db_name      | MMRnode_a
target_rrep_sync_id | 0
notes               | ERROR: duplicate key value violates unique constraint "dept_pk"
                    |   Detail: Key (deptno)=(50) already exists.
```

Manual publication table correction

The first step required in all manual conflict resolutions is to ensure all publication tables are consistent across all primary nodes. All corresponding tables must have the same rows with the same column values.

Once this state is achieved, you can then reapply transactions that failed to replicate successfully.

In the example shown in Correction strategies, the inconsistencies are the following:

- Primary nodes `MMRnode_a` and `MMRnode_b` each contain a row with primary key value `50`, but the other column values in the row are different.
- Primary node `MMRnode_c` doesn't have a row with primary key value `50`.

Assuming that the correct state of the dept table is the one in `MMRnode_b`, you can use one of the following options to correct the state of all primary nodes:

- Manually correct the dept table in `MMRnode_a` and `MMRnode_c`. That is, update the row in `MMRnode_a` so it has the correct values, and insert the missing row in `MMRnode_c`. The dept table on all nodes is then consistent and up to date.
- Manually delete the row with primary key value `50` from the table on both `MMRnode_a` and `MMRnode_b`. This brings the dept table on all primary nodes back to a prior, consistent state. Then, with the multi-master replication system running, perform the insert transaction again using the correct column values on any one of the primary nodes.

After the publication table rows are corrected, update the appropriate control schema table in the publication database currently designated as the controller database to indicate that the conflict was resolved.

The method outlined by the first bullet point is accomplished as follows.

Step 1

Manually correct the rows in the publication tables with SQL statements incorporated in a transaction block as described in Conflict resolution concept for the log-based method.

On `MMRnode_a`, correct the erroneous row by running the following transaction block:

```
BEGIN;
UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
  WHERE key = 'last_mcr_timestamp';
UPDATE edb.dept SET dname = 'MARKETING', loc = 'LOS
ANGELES'
  WHERE deptno =
50;
COMMIT;
```

This is shown by the following:

```
MMRnode_a=# BEGIN;
BEGIN
MMRnode_a=# UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
MMRnode_a-#   WHERE key = 'last_mcr_timestamp';
UPDATE 1
MMRnode_a=# UPDATE edb.dept SET dname = 'MARKETING', loc = 'LOS
ANGELES'
MMRnode_a-#   WHERE deptno =
50;
UPDATE 1
MMRnode_a=# COMMIT;
COMMIT
MMRnode_a=# SELECT * FROM edb.dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_c`, insert the missing row with the following transaction block:

```
BEGIN;
UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
  WHERE key = 'last_mcr_timestamp';
```

```
INSERT INTO edb.dept VALUES (50,'MARKETING','LOS
ANGELES');
COMMIT;
```

This is shown by the following:

```
 MMRnode_c=# BEGIN;
BEGIN
MMRnode_c=# UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
MMRnode_c-#   WHERE key = 'last_mcr_timestamp';
UPDATE 1
MMRnode_c=# INSERT INTO edb.dept VALUES (50,'MARKETING','LOS
ANGELES');
INSERT 0 1
MMRnode_c=# COMMIT;
COMMIT
MMRnode_c=# SELECT * FROM edb.dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

The dept table on `MMRnode_a` and `MMRnode_c` now match the content of the table on `MMRnode_b`:

Step 2

In the control schema of the publication database currently designated as the controller database, modify the entry in the xdb_conflicts table to indicate the conflict was resolved. Table xdb_conflicts is located in schema `_edb_replicator_pub`.

> **Note**
>
> The entries in table xdb_conflicts affect only the data that appears in the **Conflict History** tab and the SQL query described in Finding conflicts. Changing entries in xdb_conflicts doesn't affect future replication operations but provides a way to keep a record of how past conflicts were resolved.

Note the following regarding the xdb_conflicts table:

- A row in the xdb_conflicts table appears as an entry in the **Conflict History** tab.
- The primary key of the xdb_conflicts table is made up of columns `src_db_id`, `target_db_id`, `src_rrep_sync_id`, and `target_rrep_sync_id`. `Column src_db_id` contains a unique identifier for the primary node in which a transaction occurred that results in a conflict when replicated to the primary node identified by `target_db_id`. `src_rrep_sync_id` is the identifier of the transaction on the source primary node involved in the conflict. `target_rrep_sync_id` is the identifier of the transaction on the target primary node involved in the conflict.

  > **Note**
  >
  > The `src_rrep_sync_id` and `target_rrep_sync_id` values are used internally by Replication Server. You don't need them for the manual conflict resolution process.

- Table xdb_pub_database in the control schema associates the database identifiers `src_db_id` and `target_db_id` with the primary node attributes such as the database name, IP address, and port.

- Column `table_id` is the identifier of the publication table where the conflict occurred. Association of the `table_id` value with the publication table attributes, such as its name and schema, is found in each primary node in `_edb_replicator_pub.rrep_tables`.

- Column `pk_value` contains text indicating the primary key value that resulted in the conflict. The text is formatted as `column_name=value`. If the primary key is composed of two or more columns, each column and value pair is separated by the keyword AND, such as `column_1=value_1 AND column_2=value_2`. This syntax provides the primary key of the row in the publication table designated by `table_id` that resulted in the conflict.

- Column `resolution_status` indicates the status of the conflict. Possible values are `P` (pending) or `C` (completed: the conflict was resolved). This status appears in the Resolution Status column of the **Conflict History** tab.

- You can use column `win_db_id` to record the database identifier of the primary node that contains the winning (accepted) transaction. This information appears in the Winning DB column of the **Conflict History** tab.

Pending uniqueness conflict

For this example, you can find the entry for the pending insert/insert conflict on the `deptno` primary key value of `50` in xdb_conflicts with the following query:

```
MMRnode_a=# SELECT * FROM _edb_replicator_pub.xdb_conflicts
MMRnode_a-#   WHERE pk_value =
'deptno=50'
MMRnode_a-#     AND conflict_type = 'II'
MMRnode_a-#     AND resolution_status = 'P';
```

```
-[ RECORD 1 ]-------+--------------------------
src_db_id           | 1
target_db_id        | 22
src_rrep_sync_id    | 44713808
target_rrep_sync_id | 44718040
table_id            | 31
conflict_time       | 21-AUG-15 15:34:55.134171
resolution_status   | P
resolution_strategy |
resolution_time     |
alert_status        |
conflict_type       | II
win_db_id           | 0
win_rrep_sync_id    | 0
notes               |
pk_value            | deptno=50
```

This entry appears in the Postgres Enterprise Manager Client.

Pending conflict in xdb_conflicts

Change the value in column `resolution_status` from `P` (pending) to `C` (completed) to indicate this conflict was resolved. Change the value in `winning_db_id` to `22` to indicate primary node `MMRnode_b` contains the winning transaction.

The SQL statement to perform this update for the `MMRnode_a` to the `MMRnode_b` synchronization conflict is:

```
UPDATE _edb_replicator_pub.xdb_conflicts SET
  resolution_status = 'C',
  win_db_id = 22
```

```
WHERE pk_value =
'deptno=50'
    AND conflict_type = 'II'
    AND resolution_status = 'P';
```

When viewed in the **Conflict History** tab, the entry now shows `Resolved` instead of `Pending` in the Resolution Status column, and the Winning DB column shows the address of primary node `MMRnode_b`.

Correction using new transactions

Another method for bringing all the publication tables to a consistent state is by removing any changes caused by the conflicting transactions and then issuing new, corrected transactions at one primary node. You allow the multi-master replication system to synchronize this node to all other primary nodes.

Referring back to the uniqueness conflict on the dept table, instead of correcting the erroneous row and inserting the row into the primary node where it's missing, as described in Manual publication table correction, you can delete the conflicting rows from all primary nodes and then insert the correct row in one primary node. Then let the multi-master replication system synchronize the correct row to all primary nodes.

1. Manually delete the inserted row from the publication tables in all primary nodes using the transaction block described inConflict resolution concept for the log-based method.

   On `MMRnode_a`, delete the erroneous row with the following transaction block:

```
 BEGIN;
UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
  WHERE key = 'last_mcr_timestamp';
DELETE FROM edb.dept WHERE deptno =
50;
COMMIT;
This is shown by the following:
MMRnode_a=# BEGIN;
BEGIN
MMRnode_a=# UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
MMRnode_a-#   WHERE key = 'last_mcr_timestamp';
UPDATE 1
MMRnode_a=# DELETE FROM edb.dept WHERE deptno =
50;
DELETE 1
MMRnode_a=# COMMIT;
COMMIT
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

   On `MMRnode_b`, delete the row even though the transaction created the correct result:

```
 MMRnode_b=# BEGIN;
BEGIN
MMRnode_b=# UPDATE _edb_replicator_pub.rrep_properties SET value = current_timestamp
MMRnode_b-#   WHERE key = 'last_mcr_timestamp';
UPDATE 1
```

```
MMRnode_b=# DELETE FROM edb.dept WHERE deptno =
50;
DELETE 1
MMRnode_b=# COMMIT;
COMMIT
MMRnode_b=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

On `MMRnode_c`, no changes are required, as the conflicting transaction didn't insert a new row into the table on this node:

```
MMRnode_c=# SET search_path TO
edb;
SET
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

2. Rerun the correct transaction on one primary node with the multi-master replication system running. Don't run this in the transaction block described in Conflict resolution concept for the log-based method, as the objective is to synchronize it to all primary nodes.

   For this example, execute the correct INSERT statement on `MMRnode_a`:

   On `MMRnode_a`:

```
MMRnode_a=# INSERT INTO dept VALUES (50, 'MARKETING', 'LOS
ANGELES');
INSERT 0 1
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

3. Perform synchronization replication.

   The same rows now appear in the publication table on all primary nodes.

   On `MMRnode_a`;

```
MMRnode_a=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_b`;

```
MMRnode_b=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

On `MMRnode_c`;

```
MMRnode_c=# SELECT * FROM dept;
```

```
 deptno |   dname    |     loc
--------+------------+-------------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
     50 | MARKETING  | LOS ANGELES
(5 rows)
```

4. In the control schema of the publication database currently designated as the controller database, modify the entry in the xdb_conflicts table to indicate the conflict was resolved as in Step 2 of Conflict resolution concept for the log-based method.

9.6.11 Viewing conflict history

Conflict history shows the following types of events that occurred during synchronization replication:

- Uniqueness conflicts where two or more primary nodes attempted to insert a row with the same primary key value or unique column value
- Update/update conflicts where two or more primary nodes attempted to update the same column of the same row
- Update/delete and delete/update conflicts where one primary node attempted to update a row that was deleted by another primary node

See Conflict resolution for more information on conflict resolution.

> **Note**
>
> You can view the conflict history from the Publication node under any primary node in the multi-primary replication system. The history shows conflicts on all publication tables of all primary nodes that occurred during synchronization. Thus the history appears the same regardless of the primary node under which you view it.

> **Note**
>
> For uniqueness (insert/insert) conflicts, the number of entries appearing under the **Conflict History** tab differs when you use the trigger-based method of synchronization replication versus the log-based method. If you use the trigger-based method, a single insert/insert conflict appears as two entries in the conflict history. Each entry differs in that the source and target database fields for the two conflicting primary nodes are interchanged. If the same conflict occurs when you use the log-based method, only one entry appears in the conflict history.

To view the conflict history:

1. In the Replication Server console, select any Publication node under a Database node representing a primary node. Tabs labeled **General**, **Realtime Monitor**, **Replication History**, and **Conflict History** appear.
2. Select the **Conflict History** tab. Select **Refresh** to ensure all the conflicts are listed.
3. Use the **Conflict Display Criteria** list to display only conflicts of the chosen status: pending, resolved, failed, or all.
4. Select **View Data** to show the details of a particular conflict.

> **Note**
>
> **View Data** and the Conflict Details window are available only for multi-primary replication systems configured with the trigger-based method of synchronization replication. These elements don't appear for multi-primary replication systems configured with the log-based method of synchronization replication.

9.6.12 Updating the conflict resolution options

You can change conflict resolution options on a publication table. (See Conflict resolution for information on conflict resolution.)

1. In the Replication Server console, make sure the publication server whose node is the parent of the publication you want to change is running and registered in the console you're using. See Registering a publication server to learn how to start and register a publication server.
2. Select the Publication node under the Publication Database node representing the primary definition node.
3. Select **Publication > Update Publication > Conflict Resolution**.
4. In the Conflict Resolution Options dialog box, for each table, select the primary conflict resolution strategy and a standby strategy. Select the corresponding box to view the list of options.
5. Select **Update**.
6. When Conflict Resolution Options Updated Successfully appears, select **OK**.

9.7 Enabling and disabling table filters on a primary node

You must first define table filters in a set of available table filters in the publication before you can enable them on a primary node. See Adding a publication for information on defining table filters in a multi-master replication system. See Table settings and restrictions for table filters for table setup requirements for a log-based replication system as well as general restrictions on the use of table filters.

To enable or disable table filters on an existing primary node:

1. In the Replication Server console, make sure the publication server whose node is the parent of the primary nodes of the replication system is running and registered in the console you're using. See Registering a publication server to learn ho to start and register a publication server.

2. Select the Publication Database node corresponding to the primary node on which you want to enable or disable individual filter rules.

3. On the Publication Database node, from the context menu, select **Update Filter Rule**.

> **Note**
>
> If you want to enable or disable filter rules on the current primary definition node, you must first switch the role of the primary definition node to another primary node. Switching the role causes **Update Filter Rule** to appear on the primary node context menu. See Switching the primary definition node to learn how to switch the primary definition node.

The primary node you choose as the new primary definition node must contain a superset, or at least an equivalent set, of data as the current primary definition node. This ensures that the former primary definition node contains the complete set of data to satisfy the filtering criteria. This applies after you take a snapshot from the new primary definition node to the former primary definition node on which you just enabled the table filters.

4. In the **Filter Rules** tab, select and clear boxes to specify the filter rules to enable or disable on the primary node. You can enable at most one filter rule on any table.

5. Select **Save**. A confirmation box shows a warning message and a recommendation to perform a snapshot replication to any primary node on which you changed the filtering criteria.

6. Select **OK** to proceed with the update to the filter rule selections.

Filter Rules Updated Successfully appears after the successful update.

We strongly recommend that you perform a snapshot replication to the primary node that contains tables on which the filtering criteria changed. A snapshot ensures that the content of the primary node tables is consistent with the updated filtering criteria. See Performing snapshot peplication for more information.

> **Note**
>
> The primary definition node, which provides the source of the table content for a snapshot, must contain a superset of all the data contained in the other primary nodes of the multi-master replication system. This ensures that the target of the snapshot receives all of the data that satisfies the updated filtering criteria.

If the primary definition node contains only a subset of all the data contained in the other primary nodes, then a snapshot to another primary node might not result in the complete set of data that is required for that target primary node.

9.8 Switching the primary definition node

After you create the multi-master replication system, you can switch the role of the primary definition node with another primary node.

1. In the Replication Server console, make sure the publication server whose node is the parent of the primary nodes of the replication system is running and registered in the console you're using. See Registering a publication server to learn how to start and registe a publication server.

2. Select the Publication Database node corresponding to the primary node that you want to set as the primary definition node.

3. From the context menu, select **Set as MDN**.

4. In the confirmation box, select **Yes**. The selected master node is now the master definition node.

The value **Yes** in the **MDN** field of the Property window indicates this database is the primary definition node.

The new primary definition node moves to the top of the replication tree.

> **Note**

> Perform a synchronization replication to ensure that the new primary definition node is synchronized with the other primary nodes. See Performing synchronization replication for details.

9.9 Ensuring high availability

In a multi-master replication system, the primary nodes participating in replication can reside on separate physical hosts. If any primary node goes offline, the primary nodes on the other hosts continue to synchronize transactions among themselves, which ensures consistency of the publication tables on the remaining active primary nodes. When an offline primary node is brought back online, pending transactions involving that primary node are synchronized with the other primary nodes of the replication system. No transaction data is lost between the primary nodes.

Thus, an inherent characteristic of multi-master replication systems is that each primary node serves as a backup for the other nodes, and any such node can provide consistent publication data to applications.

Similarly, the complete, multi-master replication system configuration information (that is, the control schema and its control schema objects) is stored in each publication database (that is, every primary node) of the multi-master replication system.

If any primary node goes offline, the configuration information stored in the control schema is always available to the publication server so that the replication system operation can continue.

Although every publication database contains a copy of the control schema, the publication database designated as the controller database has special significance to the operation of the replication system.

Significance of the controller database

Throughout operation of the replication system, one of the publication databases of the primary nodes is designated as the controller database.

You can identify the controller database in either of these ways:

- In the Replication Server console, when you select a primary node, the **Controller database** field in the Property window is set to **Yes** if this primary node is the current controller database.
- In the Replication Server configuration file, the authentication and connection parameters are set to the controller database. See Replication Server configuration file for details.

When a replication system is in use, the Replication Server, and particularly the publication server component, accesses the currently designated controller database for configuration information.

Any changes that you make to the replication system configuration using the Replication Server console or CLI first update the control schema of the controller database. Then they are replicated by the Replication Server to the other publication databases to keep such information consistent.

> **Note**
>
> Replication history can take a longer time to replicate from the controller database to the other publication databases, It is possible that some replication history can be lost if access to the controller database fails and a switchover is made to another publication database to act as the controller database. See Viewing replication history for more information.

It's important that the controller database be accessible whenever the replication system is use.

Automatic switchover of the controller database

If the publication server is currently running with its connection to the controller database and that database suddenly becomes inaccessible such as with

a network or system problem, the Replication Server automatically performs a connection to another online publication database to act as the controller database There is no apparent disruption in the operation of the Replication Server.

Modify the controller database authentication and connection information accordingly in the Replication Server configuration file (see Replication Server configuration file). Any later startups of the publication and subscription servers use this newly designated controller database.

You can later change the controller database to use another publication database as described in Switching an active controller database and Restarting with an alternate controller database.

Switching an active controller database

If you must take the database server hosting the controller database offline for maintenance or some other reasons, you can switch the role of the controller database to another publication database.

If the publication server is currently running, you can make this switch using the Replication Server console (seeSwitching the controller database) or the Replication Server CLI (see Setting the controller (setascontroller)).

After the switch, you can take the former controller database offline. Any pending transactions involving the former controller database are applied after it is brought back online.

If the publication server isn't running, you can still change the controller database so that the publication server connects to a newly designated controller database when the publication server starts. See Restarting with an alternate controller database for information on this method.

Restarting with an alternate controller database

If the publication serveran can't access the currently designated controller database, symptoms such as the following might occur:

- The Replication Server console is unresponsive or the Replication Server CLI commands fail unpredictably.
- The publication server isn't running and you can't successfully start it.

If you can't access the controller database, then you can switch the controller database role to another publication database. Edit the Replication Server configuration file so it contains the connection information of another primary node in the replication system. See Replication Server configuration file for more information.

After you modify the Replication Server configuration file, restart the publication server and the subscription server if you are using that. SeeRegistering a publication server for instructions on starting the publication server. See Registering a subscription server for instructions on starting the subscription server.

9.10 Optimizing performance

Publication server configuration options are available to optimize the performance of multi-master replication systems.

Almost all publication server performance-related configuration options for single-master replication systems apply to multi-master replication systems (except when they are database-product specific, such as for Oracle).

Set the publication server configuration options in the publication server configuration file. See Publication and subscription server configuration options for a detailed explanation of how to set the configuration options in this file.

In addition, for configuration options that apply specifically to publication databases configured with the log-based method of synchronization replication, see Log-based method of synchronization options.

The following are some other configuration options that apply only to multi-master replication systems.

`uniquenessConflictDetection`

The `uniquenessConflictDetection` option determines whether to detect uniqueness conflict at data load time or when data is applied against a target primary node. Possible values are `EAGER` and `LAZY`. Set it to `EAGER` if there is a high probability of duplicate inserts across primary nodes.

When the number of primary nodes is equal to two, then the conflict detection is performed in the default `LAZY` mode.

When the number of primary nodes is greater than two, then the conflict detection is always performed in EAGER mode. (A `LAZY` mode setting is ignored.) This is primarily required to avoid removing the already replicated conflicted changes from a target node, which otherwise is an expensive option.

`uniquenessConflictDetection={EAGER | LAZY}`

The default value is `LAZY` when the number of primary nodes is two.

`skipConflictDetection`

The `skipConflictDetection` option controls whether to skip conflict detection during synchronization replication. The default is `false`. Change this value only when the chance of data conflict across primary nodes is zero. For example, if each primary node operates on an independent set of data, then turning on this option improves the replication time.

`skipConflictDetection={true | false}`

The default value is `false`.

`deadlockRetryCount`

This option applies in a multi-master replication system if a deadlock is detected on a target primary node. The `deadlockRetryCount` option controls the number of times the publication server attempts to retry applying the changes in the current replication cycle after waiting for the number of milliseconds specified by `deadlockWaitTime`. Set `deadlockRetryCount` to `0` to turn off this option. When this option is off, the failed changes are tried in the next replication cycle.

`deadlockRetryCount=n`

The default value for `n` is `1`.

`deadlockWaitTime`

Use the `deadlockWaitTime` option with the `deadlockRetryCount` option to set the wait time before the publication server attempts to retry applying the changes on the target primary node. Specify a value in milliseconds.

`deadlockRetryCount=n` `

The default value for `n` is `1000`.

10 Common operations

These configuration and maintenance operations of Replication Server are common to both single-master and multi-master replication systems.

For configuration and management of your replication system, the Replication Server console is used to illustrate steps and examples. You can perform the same steps from the operating system command line using the Replication Server command line interface (CLI). The commands of the Replication Server CLI utility are described in Replication Server command line interface.

10.1 Selecting tables with the wildcard selector

When selecting tables for creating a publication for a single-master replication system (see Adding a publication) or a multi-master replication system (see Adding a publication), there might be cases when the number of available tables for selection is so large that selecting the ones you want is difficult and time consuming.

You might also encounter this issue when adding tables to an existing publication (seeAdding tables to a publication) or deleting tables from an existing publication (see Removing tables from a publication).

In such cases, the wildcard selector lets you choose a set of tables by using pattern matching similar to the technique used by the SQL statement `LIKE` clause.

Wildcard selector patterns

Pattern matching, as performed by the wildcard selector, is the process in which the eligible tables for an operation are returned in a filtered list if their schema and table name combination match a character string, called a pattern. Matching a pattern means that the schema and table name combined in a string formatted as `schema_name.table_name` matches the pattern, character by character, according to the rules designated for the characters appearing in the pattern.

If the `schema_name.table_name` string matches the pattern, then the schema and table are displayed in the filtered list for that pattern, which is the **Available Tables** field of the Wildcard Selector dialog box. You can then selectively choose the tables from the filtered list to add to a local list, which contains the candidate tables for the operation for which you're using the wildcard selector.

Similarly, you can remove tables from the local list that were previously selected if you decide that you don't want those tables applied for the operation.

With the exception of wildcards characters, characters appearing in a pattern require that the character in the corresponding position in the `schema_name.table_name` string must match the pattern character. This pattern matching is not case sensitive.

Wildcards characters are interpreted as follows:

- `?` – Single-character wildcard specifies that any single character can exist in its position of the pattern. (The `SQL LIKE` clause uses the underscore character (_) for this purpose.)
- `%` - Multi-character wildcard specifies that any combination of multiple characters, including the absence of any character, can exist in its position of the pattern.
- `[abc...]` – List wildcard specifies that any one of the characters listed in the brackets can exist in its position of the pattern.
- `[a-d]` – Range wildcard specifies that any one character that is greater than or equal to the character preceding the hyphen (-) and less than or equal to the character following the hyphen can exist in its position of the pattern.
- `[abcd-f...]` – List and range combination wildcard specifies that any character that matches any of the list or range wildcard descriptions can exist in its position of the pattern.
- Any character specified in the pattern other than `?`, `%`, `[`, `]`, and the characters enclosed in the square brackets of a list or range wildcard must exist in its position of the pattern. Pattern matching of such characters is not case sensitive.
- `NOT pattern`, `!pattern`, `! pattern` – Exclusive pattern specifies that tables that match the pattern string indicated by pattern are omitted from the filtered list. Tables that don't match pattern are included in the filtered list. The keyword `NOT` can be in upper case, lower case, or mixed case, but it must be followed by a single space character preceding pattern. `!pattern` specifies that pattern immediately follows the exclamation point (!) with no intervening space character. `! pattern` specifies that a single space character exists between pattern and the exclamation point (!).
- `pattern*` - Specify the asterisk (*) immediately following the pattern with no intervening space character if you want to include tables in the filtered list that match pattern and were previously selected (that is, the local list tables) along with tables that weren't selected. In the filtered list, each previously selected, local list table is displayed with its check box selected. Each filtered list table that wasn't previously selected has its check box cleared. By default, when you omit the asterisk, only tables that were not previously selected are returned in the filtered list. Using the asterisk is useful for removing currently selected tables from the local list.

You can see the wildcard pattern definitions and examples from a help screen. Right-click the **Filter Pattern** text field of the Wildcard Selector dialog box to access the help screen.

Using the wildcard selector

The following terms are used in the Wildcard Selector dialog box and the description of the wildcard selector feature:

- **Calling dialog box**. This is the dialog box of the operation from which you invoke the Wildcard Selector dialog box. The final set of tables from the wildcard selector is applied to the operation managed by the calling dialog box.
- **Table list.** This is the list of currently selected tables displayed in the calling dialog box. Each selected table has its check box selected.
- **Local list.** This is a temporary, internal copy of the table list managed by the wildcard selector. The wildcard selector allows you to add tables to the local list and to remove tables from the local list. When you select **Done** in the Wildcard Selector dialog box, the local list becomes the table list. In other words, the local list tables appear as the selected tables of the calling dialog box.
- **Unselected tables.** These are the tables eligible, but not selected, for the operation with which you're using the wildcard selector. When you select **Filter List**, the unselected tables that match the filter pattern are listed in the **Available Tables** field of the Wildcard Selector dialog box. To list all unselected tables, use the percent sign (`%`) for the filter pattern.
- **Selected tables.** These are the tables you selected for the operation with which you're using the wildcard selector. That is, these tables make up the local list. To display selected tables that match a filter pattern, add the asterisk character (`*`) immediately after the filter pattern. Each selected table has its check box selected.

To use the wildcard selector:

1. Prior to opening the Wildcard Selector dialog box, you can start selecting tables from the list of available tables of the calling dialog box.

2. From the calling dialog box, select **Use Wildcard Selection**.

   In the Wildcard Selector dialog box, any tables that you preselected are included in the local list used by the wildcard selector to manage the addition or removal of tables. When you first open the Wildcard Selector, the default filter pattern is the percent sign (`%`), which returns all eligible, unselected tables.

   The **Available Tables** field displays the filtered list matching the pattern used in the Filter Pattern text field.

3. Enter a pattern in the **Filter Pattern** text field to narrow your desired table selection. Select **Filter List** to display the tables that match the pattern.

4. Select tables from the **Available Tables** list that you want to add to the local list by selecting each table's check box. Select **Select All** check box to select all tables and then deselect certain tables by clearing the check box.

5. Select **Apply Selections to Local List** button to add the selected tables to the local list.

6. Apply as many filter patterns as needed to add all of your desired tables to the local list.

   Select all the tables from this filtered list by selecting **Select All**.

   Click **Apply Selections to Local List** to add all tables to the local list. After applying the selections, there are no unselected tables remaining that match the filter pattern.

   By using the asterisk after the pattern, you can display previously selected tables comprising the local list. Each selected table has a check mark its check box.

   You can remove selected tables from the local list by clearing each such table's check box.

   Removal of the deselected tables from the local list occurs along with the addition of any newly selected tables when you select **Apply Selections to Local List**.

   The deselected tables still appear in the Available Tables list since they still match the pattern but as unselected tables.

7. When the local list contains all of the tables you want, select **Done**. The local list becomes the list of selected tables displayed in the calling dialog box.

8. When the calling dialog box contains the complete list of your desired tables, select the appropriate action in the calling dialog box to complete the operation with the selected tables.

10.2 Creating a schedule

A schedule establishes recurring times for replication.

> **Note**
>
> (For MMR only): Be sure an initial snapshot replication was performed from the primary definition node to every other primary node in the multi-master replication system. If a newly added primary node didn't undergo an initial snapshot, any subsequent synchronization replication started by a schedule might fail to apply the transactions to that primary node. You can take the initial snapshot when you first add the primary node (see Creating more primary nodes) or by performing an on-demand snapshot (see Performing snapshot replication).

In a single-master replication system, once a schedule is created, the subscription server starts replications according to the schedule until you either change or remove the schedule. In a multi-master replication system, the publication server handles this process.

See Managing a schedule for changing or removing a schedule.

When a scheduled replication occurs, all components of the replication system must be running:

- Publication database server
- Subscription database server (applies only to single-master replication systems)
- Publication server
- Subscription server (applies only to single-master replication systems)

If any of these components isn't running at the time of a scheduled replication, then the scheduled replication doesn't occur. The replication occurs at the next scheduled replication time when all required replication system components are running.

For synchronization replications with the trigger-based method, changes that occurred on the source tables that weren't replicated due to a skipped scheduled replication are maintained as pending transactions in the shadow tables of the source database.

For synchronization replications with the log-based method, changes that were extracted from the WAL files to in-memory structures but weren't applied are persisted using Java object serialization to files on the host running the publication server.

All changes since the last successful replication are applied whenever the next scheduled replication occurs. Thus, accumulated changes are never lost due to a missed replication.

For snapshot replications, skipped scheduled replications present no problem since a snapshot replication replaces all of the data in the target tables with the current source data.

1. For SMR: Select the Subscription node of the subscription for which you want to create a schedule.

   For MMR: Select the Publication Database node designated as the controller database. (The **Controller database** field in the Property window is set to **Yes** for the controller database.)

2. For SMR: Select **Subscription > Schedule > Configure Schedule**.

   For MMR: On the Publication Database node and select **Configure Schedule** from the context menu.

3. In the Scheduled Task Wizard dialog box, select either synchronization replication or snapshot replication.

   > **Note**
   >
   > If the publication associated with this subscription is a snapshot-only publication, then you can select only **Snapshot**. In a multi-master replication system, you can select only **Synchronize**.

4. Select the scheduled replication frequency, or select **Cron Expression** to write your own cron expression. The frequency choices have the following meanings:

- **Continuously**. Schedules replication to run continuously at an interval in seconds that you specify. Select this option if the source tables change frequently during the day and the target tables must be kept up to date throughout the course of the day.
- **Daily**. Schedules replication to run once a day at the time you choose. Select this option if the target tables need to be refreshed daily.
- **Weekly**. Schedules replication to run once a day at the time you choose, but only on the specific days of the week you choose. Select this option if you need more flexibility than a daily schedule, and the target tables don't have to be refreshed every day.
- **Monthly**. Schedules replication to run one day per month on the day of the month and time you choose, but only in the months you choose. Select this option if updates to the source tables aren't very frequent, and the target tables can be out of date by a month or more. The **Monthly** option allows you to schedule replication for as often as once a month or as infrequently as once a year.
- **Cron Expression**. Provides additional flexibility for specifying a schedule. See Writing a cron expression for details.

5. Select **Next**.

6. Review the schedule and select **Finish** to accept it.

Select **Refresh** to see the schedule properties in the **General** tab.

10.3 Managing a schedule

After you create a schedule, Replication Server performs replications according to the schedule until you update or remove the schedule.

Updating or removing a schedule doesn't affect a replication that already started. If a replication is in progress when the schedule is updated or removed, the in-progress replication continues until completion.

Update a schedule

To change an existing schedule for SMR:

1. Make sure the subscription server whose node is the parent of the subscription you want to change is running and registered in the Replication Server console you're using. See Registering a subscription server to learn how to start and register a subscription server.

2. Select the Subscription node of the subscription whose schedule you want to update.

3. Select **Subscription > Schedule > Configure Schedule**.

4. In the Configure Scheduler confirmation box, select **Yes**.

5. In the Scheduled Task Wizard dialog box, create the new schedule. See Creating a schedule for details.

To change an existing schedle for MMR:

1. Make sure the publication server whose node is the parent of the controller database you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. Select the Publication Database node designated as the controller database whose schedule you want to update. From the context menu, select **Configure Schedule**.

3. In the Configure Scheduler confirmation box, select **Yes**.

4. In the Scheduled Task Wizard dialog box, create the new schedule. See Creating a schedule for details.

Remove a schedule

If you no longer want replication to take place on a schedule, remove the schedule.

To remove a schedule for SMR:

1. Make sure the subscription server whose node is the parent of the subscription you want to change is running and registered in the Replication Server console you're using. See Registering a subscription Server to learn how to start and register a subscription server.
2. Select the Subscription node of the subscription whose schedule you want to remove.
3. Select **Subscription > Schedule > Remove Schedule**.
4. In the Removing Schedule confirmation box, select **Yes**.

Select **Refresh** in the tool bar to refresh the information window. The schedule information no longer appears.

To remove a schedule for MMR:

1. Make sure the publication server whose node is the parent of the controller database you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.
2. Select the Publication Database node designated as the controller database whose schedule you want to remove. From the context menu, select **Remove Schedule**.
3. In the Removing Schedule confirmation box, select **Yes**.

Select **Refresh** in the tool bar to refresh the information window. The schedule information no longer appears.

10.4 Viewing replication history

You can view a summary of replications performed on each subscription or primary node in the Replication Server console. You can also view a detailed replication history showing each insert, update, and deletion made against each target table. See Synchronization replication with the trigger-based method for a discussion on how changes are applied to target tables for the target-based method of synchronization replication. See Synchronization replication with the log-based method for information on the log-based method of synchronization replication.

> **Note**
>
> (For SMR Only): You can view the replication history from the Publication node as well as from the Subscription node. The history shown for a Publication node is the same set of inserts, updates, and deletions made on the subscription tables by the publication server during synchronization. The history shown for a Publication node doesn't show the actual SQL statements processed on the publication tables that originated from user applications.

> **Note**
>
> (For MMR only): You can view the replication history from the Publication node under any primary node in the multi-master replication system. The history shown includes inserts, updates, and deletions made on all publication tables of all primary nodes by the publication server during synchronization. The history appears the same regardless of the primary node under which you view the history.

View all replication history

Replication history shows the following types of events that occur on a given subscription or primary node:

- Snapshot replications
- Synchronization replications where at least one change (insert, update, or deletion) was applied to a target table
- Synchronization replications where no updates were applied to any of the target tables since the last restart of the publication server

1. For SMR: Select the node under the Subscription node. Tabs labeled **General**, **Realtime Monitor**, and **Replication History** appear.

   For MMR: Select any Publication node under a Database node representing a primary node. Tabs labeled **General**, **Realtime Monitor**, **Replication History**, and **Conflict History** appear.

2. Select the **Replication History** tab to show a history of replications.

> **Note**
>
> Every snapshot replication and each synchronization replication with at least one update produces a history record that is maintained in replication history tables in the control schema. Over time the size of the replication history tables will grow. You can clean up replication history records as needed. See Clean up replication History for information.

Hide synchronizations with zero transaction counts

You might notice synchronization replications with transaction counts of zero. These records indicate that there were no changes to synchronize when the replication occurred. For scheduled replications that occur frequently, this can result in a large number of lines in the **Replication History** tab, obscuring the more meaningful replications with non-zero transaction counts.

While viewing the **Replication History** tab, you can hide the records with zero transaction counts.

1. On the **Replication History** tab, select **Show History With Transactions Count > 0**.

The next time the **Replication History** tab refreshes, only the replications with non-zero transaction counts appear.

> **Note**
>
> Zero transaction count replication records are kept in the publication server memory. By default, they aren't permanently stored on disk. Therefore, when you shut down the publication server, the in-memory zero transaction count replication records are no longer available.

When the publication server starts running again, zero transaction count replication records appear on the **Replication History** tab as zero transaction count replications occur.

To permanently store zero transaction count replication records to disk, set the publication server configuration option `persistZeroTxRepEvent` to `true`. See Replacing null characters for more information.

Shadow table history

Expanding the nodes under the Subscription node of a single-master replication system or the Publication node of a multi-master replication system provides more information about the subscription or publication.

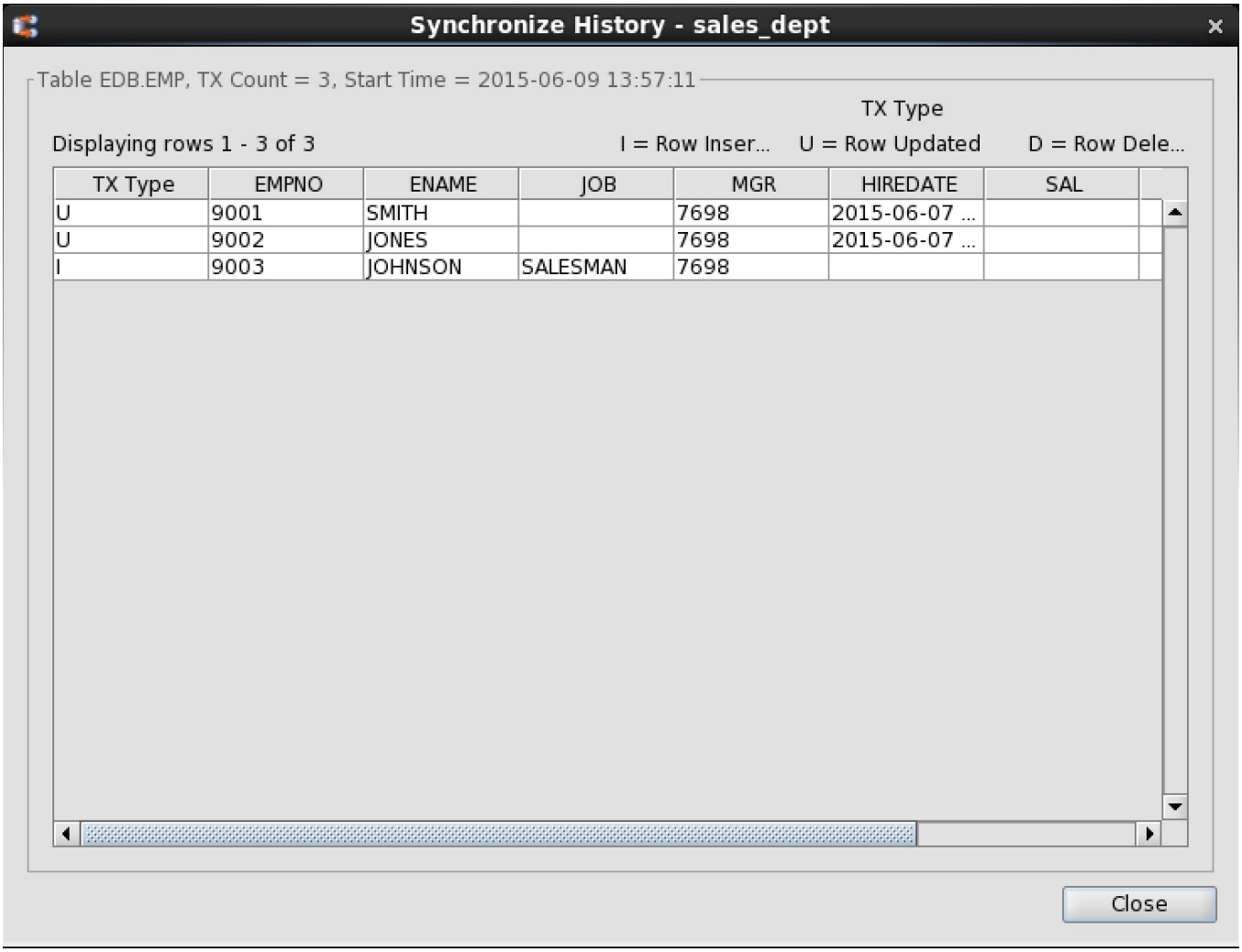
1. Select a table to reveal tabs that contain general information about the table and the replication history of the table. Expand a Table node to reveal the columns in the table.

2. Select the **Replication History** tab to show a history of replications for this table.

3. Select **View Data** to show a list of each change made to the table during the synchronization replication.

The Synchronize History window shows two update operations followed by one insert operation against the emp target table that correspond to the following set of SQL statements executed on the emp source table:

```
UPDATE emp SET hiredate = TO_DATE('07-JUN-15'), mgr = 7698 WHERE empno IN (9001,
9002);
INSERT INTO emp (empno, ename, job, mgr, deptno) VALUES (9003, 'JOHNSON', 'SALESMAN', 7698,
30);
```



Note

Since all insert, update, and delete operations on all source tables are recorded in shadow tables, the size of the shadow tables can grow considerably over time for volatile source tables. The rows shown in the Synchronize History window are obtained from these shadow tables. You can delete rows in the shadow tables as needed. See Cleaning up shadow table history for details.

10.5 Managing history

Replication Server maintains three types of history:

- Shadow table history. Records of each change (insert, update, or delete) that was applied to each target table during synchronization replications using the trigger-based method. The log-based method doesn't keep a shadow table history for synchronization replications.
- Replication history. Summary records of each replication.
- Event history. Records of each change that was applied to various control schema tables.

The size of the control schema tables that store these history records grows over time. You can use a number of methods to clean up these history records.

Shadow table history cleanup information is described in Scheduling shadow table history cleanup and Cleaning up shadow table history. For replication history cleanup, see Cleaning up replication history.

For event history cleanup, see Cleaning up event history.

Scheduling shadow table history cleanup

You can set a preference for each publication database definition to specify the schedule for shadow table history cleanup for all publications appearing under its corresponding Publication Database node. Shadow table history cleanup has no benefit for snapshot-only publications, so if all your publications under a Publication Database node are snapshot-only publications, then disable scheduled shadow table history cleanup.

Replication history isn't deleted by scheduling shadow table history cleanup. New publication database definitions include a scheduled default setting of every Sunday at 12:00 a.m. for shadow table history cleanup.

> **Note**
>
> A configuration option is available to force shadow table history cleanup after every synchronization replication. See Forcing immediate shadow table cleanup for information on this option.

> **Note**
>
> The cleanup of certain processed rows in the shadow tables might be delayed beyond the next scheduled cleanup. These roles are eventually removed in later cleanup events.

For Oracle only: For scheduling shadow table history cleanup on an Oracle publication database, use the Oracle `DBMS_JOB` package on the Oracle database server. The time you specify in the schedule for cleanup is passed and stored in `DBMS_JOB` without time zone translation.

For example, assume the publication server is running on a host in New York and the Oracle publication database is on a server in California, which has a three-hour time difference. Suppose you set shadow table history cleanup to run at 12:00 a.m. according to the New York-based publication server. The cleanup job on the California-based Oracle database starts at 12:00 a.m. Pacific Time, which is 3:00 a.m. Eastern Time.

For SQL Server only: For scheduling shadow table history cleanup on a SQL Server publication database, use SQL Server Agent on the host running SQL Server. The time you specify in the schedule for cleanup is passed to SQL Server Agent without time zone translation.

For Postgres only: For scheduling shadow table history cleanup on a Postgres publication database, use the Quartz scheduler on the host running the publication server based on the location of the controller database.

For example, assume the publication server is running on a host in New York and the Postgres publication database on which cleanup is scheduled is also the controller database and is on a host in California. If you set shadow table history cleanup to run at 12:00 a.m. according to the New York-based publication server, the cleanup job on the California-based Postgres database starts at 12:00 a.m. Pacific Time, which is 3:00 AM Eastern Time in New York.

By contrast, assume the publication server is running on a host in New York along with the controller database, and the Postgres publication database on which to schedule cleanup is on a host in California. If you set shadow table history cleanup to run at 12:00 a.m. according to the New York-based publication server and controller database, the cleanup job on the California-based Postgres database starts at 12:00 a.m. Eastern Time, which is 9:00 p.m. in California.

For Oracle only: The cleanup job on an Oracle publication database runs independently of the publication server, so the cleanup job runs regardless of whether the publication server is running.

For Postgres only: The publication server must be running for the cleanup job to run on a Postgres publication database.

> **Note**
>
> An alternative to using the Quartz scheduler when Postgres is the publication database is to use pgAgent job scheduling instead. See Using pgAgent job scheduling for information on how to use pgAgent job scheduling and its advantages.

To change the default setting:

1. Make sure the publication server whose node is the parent of the publication database definition whose cleanup scheduling preference you want to set is running and registered in the Replication Server console you are using. See Registering a publication server to learn how to start and register a publication server.

2. Select the Publication Database node whose cleanup scheduling preference you want to set.

3. Select **Publication > Preferences**.

4. In the Publication Server Preferences dialog box, clear the check box if you don't want to run a scheduled shadow table history cleanup job.

5. If you want to schedule shadow table history cleanup, instead make sure the**Run Cleanup Job** check box is selected. Select a radio button for the cleanup frequency:

    - **Every number of minutes/hours.** Schedules shadow table history cleanup to run continuously at an interval in either minutes or hours that you specify. Select this option for huge volumes of updates to the publication tables during the course of the day, every day.
    - **Every Day at hour of day.** Schedules shadow table history cleanup to run once a day at the hour you choose. Select this option if updates to the publication tables are frequent enough to require more than once-a-week cleanup but aren't needed more than once a day.
    - **Every selected day of week at hour of day.** Schedules shadow table history cleanup to run once a week on the day and at the hour you choose. Select this option if updates to the publication tables are infrequent and you don't want to run cleanup manually.
    - **Cron Expression.** Provides more flexibility for specifying a schedule. See Writing a cron expression.

    > **Note**
    >
    > A configuration option is available to force shadow table history cleanup after every synchronization replication. SeeForcing immediate shadow table cleanup for information on this option.

6. To accept the schedule, select**OK**.

Cleaning up the shadow table history

For publications that aren't snapshot-only (that is, publications on which synchronization replications occur) whose tables experience frequent changes, clean their shadow table history periodically. Otherwise the amount of disk space consumed by the shadow tables in the publication database can grow too rapidly.

Cleaning up shadow table history deletes the rows in the following Replication Server metadata tables:

- RREP_TXSET
- RREP_TXSET_LOG
- RRST_schema_table

For Oracle only: When Oracle is the publication database, these tables are located in the publication database in the schema of the publication database user.

For SQL Server only: When SQL Server is the publication database, these tables are located in the publication database in the schema you chose inSQL server publication catabase.

For Postgres only: When Postgres is the publication database, these tables are located in the publication database in schema `_edb_replicator_pub`. You can schedule shadow table history cleanup to run periodically (see Scheduling shadow table history cleanup) or on demand.

> **Note**
>
> The cleanup of certain processed rows in the shadow tables might be delayed beyond the next scheduled cleanup. When Oracle or SQL Server is the publication database for clusters created using versions 7.5 and earlier, the cleanup delay can occur due to uncommitted changes. These rows are eventually removed in later cleanup events. For new clusters created using version 7.5.1 and later, the shadow cleanup operation is

> accurate, and the delayed cleanup issue doesn't occur.

To run shadow table history cleanup on demand for a chosen publication:

1. Make sure the publication server whose node is the parent of the publication whose shadow table history you want to clean up is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.
2. Select the Publication node of the publication whose the shadow table history you want to clean up.
3. Select **Publication > Cleanup Shadow Table History**.
4. In the Cleanup Synchronization History confirmation box, select **Yes**.
5. Select **Yes** in response to Shadow Table's Transaction History Removed Successfully.

After shadow table history cleanup, if you select **View Data** in the **Replication History** tab, an information message appears stating that there's no synchronization history to view.

Cleaning up replication history

Cleaning up replication history deletes rows from the following tables in the control schema:

- xdb_pub_replog
- xdb_pub_table_replog

To run replication history cleanup for a chosen publication:

1. Make sure the publication server whose node is the parent of the publication whose replication history you want to clean up is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.
2. Select the Publication node of the publication whose replication history you want to clean up.
3. Select **Publication > Cleanup Replication History**.
4. In the Cleanup Replication History confirmation box, select **Yes**.
5. Select **Yes** in response to Replication History Has Been Removed.

After replication history cleanup, no history records appear in the **Replication History** tab.

Cleaning up event history

Unlike shadow table history (Cleaning up shadow table history) and replication history (Cleaning up replication history), you can't view or remove event history using the Replication Server console.

Event history is a recording of various updates to the control schema tables that occur during Replication Server processing. Event history content grows significantly over time.

The tables containing event history to clean up are the following:

- xdb_events

- xdb_events_status

In addition, clean up the replication history tables. You can manually clean up these tables as described in Cleaning up replication history.

- xdb_pub_replog
- xdb_pub_table_replog

For Oracle, these tables are located in the schema of the publication database user. For SQL Server and Postgres, these tables are located in schema `_edb_replicator_pub`.

The event history and replication history data in the control schema are deleted on a scheduled, daily basis at 12 a.m., thus reducing the number of rows in tables xdb_events, xdb_events_status, xdb_pub_replog, and xdb_pub_table_replog.

Publication server configuration option `historyCleanupDaysThreshold` enables you to designate how old the completed data must be before its removal. The default setting is that completed data must be older than seven days before it is deleted during the daily 12 a.m. cleanup process.

To clean up all completed event and replication history, set `historyCleanupDaysThreshold` to a value of `0`, and then restart the publication server. The cleanup occurs during the next scheduled 12 a.m. cleanup.

See Setting event history cleanup threshold for the `historyCleanupDaysThreshold` option.

10.6 Managing a publication

After you create a publication, certain aspects of the underlying replication system environment might change for any number of reasons. Attributes that might change include the network location of the publication database server, the network location of the host running the publication server, database or operating system user names and passwords, and so forth.

This information is saved in the replication system metadata when you create a publication. Changes to these attributes result in inaccurate replication system metadata, which in turn can result in errors during subsequent replication attempts or replication system administration.

You can update the metadata stored for the publication server, the publication database definition, and publications to keep the information consistent with the actual replication system environment.

10.6.1 Updating a publication server

Two aspects of metadata relate to the publication server that you might need update, depending on the change in the host environment:

- If the network location (IP address or port number), admin user name, or password of the publication server changes, and if you have saved this information in the server login file, you need to update the server login file with the new information.
- If the network location (IP address or port number) of a subscription server changes, then for each publication server managing a publication associated with a subscription of the changed subscription server, you must update the publication server's metadata with the subscription server's new network location. This type of update applies only to single-master replication systems.

Update the publication server login file

When you register a publication server in the Replication Server console, you can save the publication server's network location, admin user name, and encrypted password in a server login file on the computer on which you are running the Replication Server console. See Saving server login information for details.

Open the Replication Server console. To perform the update, the publication server whose login information you want to modify in the server login file must

appear as a Publication Server node in the replication tree.

You can perform the following actions on the server login file:

- Change the publication server's login information (host IP address, port number, admin user name, and password) that you last saved in the server login file.
- Delete the publication server's login information that is currently saved in the server login file. This is the default action, which requires you to register the publication server again the next time you open the Replication Server console.
- Resave the publication server's login information in the server login file. Each time you open the Update Publication Server dialog box, you must choose to save the login information if you want it recorded in the server login file.

Updating the publication server login file changes only the content of the server login file residing on the host under the current Replication Server console user's home directory. These changes don't change any characteristic of the actual publication server daemon (on Linux) or service (on Windows). These changes affect only how a publication server is viewed through the Replication Server console on this host by this user.

1. Make sure the publication server whose login information you want to save, change, or delete in the server login file is running. See Registering a publication server to learn how to start the publication server.

2. On the Publication Server node, select **Update** from the context menu.

3. In the Update Publication Server dialog box, complete the fields according to your needs:

    - If the publication server now runs on a host with a different IP address or port number than what is shown in the dialog box, enter the correct information. You must also enter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address you entered in the **Host** field. Select **Save Login Information** if you want the new login information saved in the server login file. If you don't select this option, access to the publication server is available for the current session: future sessions will require you to register the publication server again.
    - If you want to delete previously saved login information, make sure the network location shown in the dialog box is still correct. Reenter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address in the **Host** field. Leave the **Save Login Information** box cleared. Access to the publication server is available for this session, but future sessions will require you to register the publication server again.
    - If you want to save the current login information shown in the dialog box, make sure the network location shown in the dialog box is correct. Reenter the admin user name and password saved in the Replication Server configuration file that resides on the host identified by the IP address in the **Host** field. Select **Save Login Information**.

4. Select **Update**. When the dialog box closes, then the update to the server login file was successful. Select the **Refresh** icon in the tool bar to show the updated Publication Server node.

Update subscription server network location

> **Note**
>
> Updating subscription server network location applies only to single-master replication systems.

Part of the metadata stored for each publication server is the network location of subscription servers that manage subscriptions associated with the publication server's publications. This network information enables the publication server to communicate with the subscription server.

If the network location of a subscription server changes after subscriptions were created, you must update the publication server metadata with the new network location. Otherwise a communication failure occurs between the publication server and the subscription server and an error message appears.

To update the subscription server network location in a publication server's metadata:

1. Make sure the publication server whose metadata you want to change is running. See Registering a publication server for more information.

2. On the Publication Server node, select **Update Subscription Servers** from the context menu.

3. In the Update Subscription Servers dialog box, enter the new network location for each subscription server in the list whose network location has

changed.

4. Select **Update**. When the dialog box closes, then the update to the publication server's metadata was successful.

5. If the subscription server with the new network location manages subscriptions associated with publications in other publication servers, update those subscription servers as well.

10.6.2 Updating a publication database

When you create a publication database definition, you save the publication database server's network location (IP address and port number), the database identifier, a database login user name, and the user's password in the control schema accessed by the publication server. This login information is used whenever you need to establish a session with the publication database. See Adding a publication database for information on creating a publication database definition for a single-master replication system. See Adding the primary definition node and Creating more primary nodes for a multi-master replication system.

You can update the publication database login information stored in the control schema if any of these attributes of the physical database change.

> **Note**
>
> Depending on the database type (Oracle, SQL Server, or Postgres), you must not change certain attributes. Don't change any attribute that modifies access to the schema where the control schema objects were created when you originally added this publication database definition. See Control schema objects created for a publication for the location of the control schema objects.

Don't change the following attributes:

- The Oracle login user name of an Oracle publication database as the control schema objects already reside in this Oracle user's schema
- The database server network location if the new network location references a database server that doesn't access the publication database where the control schema objects are stored
- The database identifier if the new database identifier references a different physical database than where the control schema objects are stored

You can change the following attributes:

- The login user name's password to match a changed database user password
- The database server network location if the corresponding location change was made to the database server that accesses the publication database
- The database identifier such as the Oracle service name, SQL Server database name, or Postgres database name if the corresponding name change was made on the database server

To update a publication database:

1. Make sure the database server that you want to save as the publication database definition is running and accepting client connections.

2. Make sure the publication server whose node is the parent of the publication database definition you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

3. Select the Publication Database node corresponding to the publication database definition that you want to update.

4. Select **Publication > Publication Database > Update Database**.

5. In the Update Database Source dialog box, enter the desired changes. See Adding a publication database for the meanings of the fields for a single-master replication system. See Adding the primary definition node and Creating more primary nodes for a multi-master replication system.

6. Select **Test**. When Test Result: Success appears, select **OK**, and then select **Save**.

7. Restart the publication server. See Registering a publication server for information on restarting the publication server.

8. Select the **Refresh** icon in the tool bar to show the updated Publication Database node and any of its publications.

10.6.3 Updating a publication

You can update existing publications in the following ways:

- Add tables to the publication
- Remove tables from the publication
- Update filter rules on publication tables

Add tables to a publication

For a single-master replication system, you can add tables to a publication, even while existing subscriptions are associated with the publication. Similarly, for a multi-master replication system, you can add tables to a publication while there are more primary nodes in the replication system.

1. Make sure the publication server whose node is the parent of the publication you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. (For SMR only): Select the Publication node of the publication to which you want to add tables.

   (For MMR only): Select the Publication node under the Publication Database node representing the primary definition node.

3. Select **Publication > Update Publication > Add Tables**.

4. Fill in the following fields in the **Add Tables** tab of the Add Tables dialog box:

   - **Add**. Select the boxes next to the table names from the **Available Tables** list to add to the publication. If the publication is a snapshot-only publication, then views appear in the **Available Tables** list as well. The **Available Tables** list contains only tables and views that aren't already members of other publications under the same Publication Database node. You can also select **Use Wildcard Selection** to use wildcard pattern matching for selecting tables to add to the publication.
   - **Select All**. Select this option if you want to include all tables and views in the **Available Tables** list in the publication.
   - **Use Wildcard Selection**. Select to use the wildcard selector to choose tables for the publication. See Selecting tables with the wildcard selector for information on the wildcard selector.

5. If you want to filter the rows of the publication tables or views, select the **Table Filters** tab. Define filter rules by entering a unique, descriptive filter name and an appropriate `SQL WHERE` clause in the Filter dialog box to select the rows you want to replicate.

   For a single-master replication system, see Adding a publication for information on defining table filters on a publication table. For a multi-master replication system, see Adding a publication.

6. (For SMR only): Select **Add Tables**. When Publication Updated Successfully appears, select **OK**.

   (For MMR only): Select **Add Tables**. The `Data Sync Check` dialog box appears warning you that synchronization replication is performed before the table is added. To proceed, select **Yes**. When Publication Updated Successfully appears, select **OK**.

7. The replication tree appears with the newly added table under the Publication node. Select the **Refresh** icon. The newly added table appears under the Subscription nodes of a single-master replication system or the added primary nodes of a multi-master replication system.

8. (For MMR only): If you want to modify or see the default conflict resolution options assigned to the newly added table, follow the instructions in Updating the conflict resolution options.

9. If you defined table filters on the newly added table and you want to use these filters on any subscriptions or primary nodes, you must enable the filters on the table in the desired subscriptions or primary nodes.

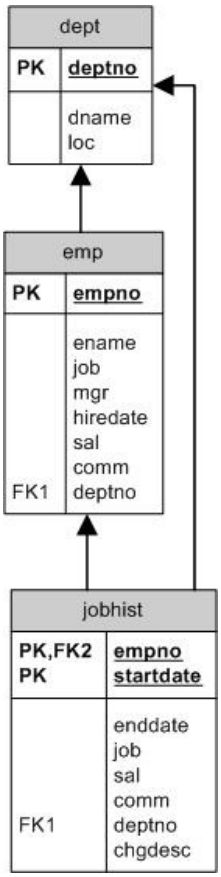
For a single-master replication system, see Enabling and disabling table filters on a subscription.

For a multi-master replication system, see Enabling and disabling table filters on a primary node.

Remove tables from a publication

You can remove one or more tables from a publication but only if the following condition is true:

- The tables to remove aren't parent tables referenced by foreign key constraints of child tables that aren't selected for removal as well.



In this entity relationship diagram, the emp table has a foreign key constraint referencing the dept table, and the jobhist table has two foreign key constraints. One constraint references the emp table and the other references the dept table.

If all three tables are in the publication, then you can remove the following combinations of tables:

- The jobhist table only
- Both the jobhist table and the emp table

1. Make sure the publication server whose node is the parent of the publication you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. (For SMR only): Select the Publication node of the publication from which you want to remove tables.

   (For MMR only): Select the Publication node under the Publication Database node representing the primary definition node.

3. Select **Publication > Update Publication > Remove Tables**.

4. Use the Remove Tables dialog box as follows:

- **Remove**. Select the boxes next to the table names from the **Available Tables** to remove from the publication. If the publication is a snapshot-only publication, then views appear in the **Available Tables** list as well. You can also select **Use Wildcard Selection** to use wildcard pattern matching for selecting tables to remove from the publication.
- **Use Wildcard Selection**. Select to use the wildcard selector to choose tables to remove from the publication. See Selecting tables with the wildcard Selector for more information.

5. Select **Remove** and then select **Yes** in response to the confirmation.

6. When Tables Removed Successfully appears, select **OK**.

The replication tree appears without the removed table under the Publication node.

10.6.4 Updating the set of available table filters in a publication

After you define a set of available table filters in the publication of a single-master replication system or a multi-master replication system, you can update the set by adding filter rules, removing filter rules, or modifying filter rules.

For background on table filters, see:

- Table settings and restrictions for table filters for table setup requirements for a log-based replication system as well as general restrictions on the use of table filters.

- Adding a publication for information on using table filters in a single-master replication system and Adding a publication for a multi-master replication system.

Updating the set of available table filters in a publication has the following implications:

- After you add filter rules to a publication, you must then enable these filter rules on the subscriptions or primary nodes that you want these filter rules to affect. See Enabling and disabling table filters on a subscription and Enabling and disabling table filters on a primary node for details.
- Removing filter rules from a publication also deletes the rules from any associated subscription or primary node on which they were enabled. You don't need to modify the subscriptions or primary nodes to disable the filter rules after you remove them from the publication.
- Modifying filter rules (changing the filter name or filter clause) applies the changes to any subscriptions or primary nodes on which the filter rules are enabled. You don't need to make any changes in the associated subscriptions or primary nodes.

After you complete your updates to the set of available table filters in the publication and enable or disable the filter rules on the target subscriptions or primary nodes, perform a snapshot replication. Perform this snapshot on any subscription or primary node affected by an updated filter rule to ensure that the content of the targeted subscription tables or primary node tables is consistent with the current set of filter rules enabled on those tables.

To update the set of available filter rules in a publication:

1. Make sure the publication server whose node is the parent of the publication you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. For SMR: Select the Publication node of the publication whose set of available filters you want to update.

   For MMR: Select the Publication node under the Publication Database node representing the primary definition node.

3. Select **Publication > Update Publication > Update Filters**.

   In the Update Filters dialog box, the set of all available filter rules defined in the publication are listed on the **Table Filters** tab.

4. To add a filter rule, from the **Table/View** list, select the table or view to which you want to add a filter and select **Add Filter**.

5. In the Add Filter dialog box, fill in the information as needed. See Adding a publication for more details on adding filter rules in a single-master replication system. See Adding a publication for a multi-master replication system.

To remove a filter rule, select the filter rule you want to remove and select **Remove Filter**.

To modify the filter name or filter clause of a filter rule listed in the **Table Filters** tab, double-click the cell of the filter name or filter clause you want to change and enter the new text.

6. When you are satisfied with the updated set of available table filters, select **Update**.

7. A confirmation appears recommending that you perform a snapshot replication to any subscription or primary node on which you intend to enable the change in filtering criteria. Select **OK**.

8. You can selectively enable any new filter rules to the corresponding tables of the associated subscriptions or primary nodes. See Enabling and disabling table filters on a subscription and Enabling and disabling table filters on a primary node for details.

10.6.5 Validating a publication

After you create a publication, don't directly change the definitions of the tables belonging to the publication. Doing so can cause a failure during the replication process. Examples of table definitions that you must not change include:

- Adding or removing columns to a table
- Renaming columns
- Changing the data types of columns
- Changing the lengths of columns
- Changing a not-nullable column to nullable or a nullable column to not-nullable
- Adding or removing uniqueness constraints
- Adding or removing check constraints

In a single-master replication system, Replication Server doesn't propagate table definition changes to the subscription tables after you create the subscription tables. Rows that might be allowed in a modified publication table might not be allowed in the subscription table you didn't change. This condition causes an error during replication.

Similarly, in a multi-master replication system, table definition changes aren't propagated from one primary node to another. The exception is when you add a new primary node and you choose to replicate the schema definition from the primary definition node.

In addition, for synchronization replication with the trigger-based method, triggers are generated on the publication tables that use certain attributes of these tables. If you change the table definition, the trigger might not function properly.

> **Note**
>
> Don't change the triggers generated by Replication Server. If you need to regenerate the triggers, you must remove the associated publication and then re-create the publication.

> **Note**
>
> You can make certain table definition changes that are propagated by Replication Server by using the DDL change replication feature. See Replicating DDL changes for details.

If you don't use the DDL change replication feature, then take the following general steps if you make table definition changes.

In a single-master replication system, if you made changes to the definitions of one or more publication tables, the resolution to the problem must be handled on a case-by-case basis as it depends upon the type of changes that were made. In the worst-case scenario, you must remove and re-create the subscription and publication as follows:

- Remove the subscription that is associated with the publication. See Removing a subscription.
- Remove the subscription tables from the subscription database. Use `SQL DROP TABLE` statements in the database system.
- Remove the publication. See Removing a publication.
- Add the publication again. See Adding a publication.

- Add the subscription again. See Adding a subscription.

In a multi-master replication system, if you made changes to the definitions of one or more publication tables on one or more primary nodes, the resolution to the problem involves:

- Making sure the table definitions are updated on all primary nodes so that they are identical, or updating the table definition on the primary definition node so it can be replicated to the other primary nodes.
- Re-creating the publication database definitions of the primary nodes.

The general steps are the following:

- Remove the publication database definitions of all primary nodes except for the primary definition node. See Removing a publication database.
- Remove the publication. See Removing a publication.
- Remove the publication database definition of the primary definition node. See Removing a publication database. At this point, all of the triggers, shadow tables, and metadata are removed from the primary nodes.
- With respect to the publication table definitions, you can either:
  - Update the table definitions on all primary nodes so that they are identical.
  - Assume the table definitions on the primary definition node are up to date, and delete the out-of-date table definitions on all other primary nodes.

- Add the publication database definition for the primary definition node again. See Adding the primary definition node.
- Add the publication again. See Adding a publication.
- Add more primary nodes. See Creating more primary nodes. When adding a primary node, clear the **Replicate Publication Schema** check box if you already created the table definitions on all primary nodes. Select the **Replicate Publication Schema** check box if you want to propagate the table definitions from the primary definition node to all other primary nodes. A snapshot reloads the primary node tables from the primary definition node.

Validate a single publication

Replication Server provides a way to verify that certain characteristics of publication tables haven't changed since you created the publication.

> **Note**
>
> This validation feature is available only for publications using the trigger-based method of synchronization replication. This validation feature isn't available for publications using the log-based method of synchronization replication.

This validation operation and those in Validating all publications can check for the following types of table modifications:

- Adding columns to a table
- Removing columns from a table
- Renaming columns

> **Note**
>
> In a multi-master replication system, publication tables in only the primary definition node are validated. The validation operation doesn't check if table definitions have changed in other primary nodes.

To validate a single publication:

1. Make sure the publication server whose node is the parent of the publication you want to validate is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. For SMR: Select the Publication node of the publication you want to validate.

   For MMR: Select the Publication node under the Publication Database node representing the primary definition node.

3. Select **Publication > Validate Publication**.

4. When All Schema of Published Tables in Publication 'publication_name' Are Up-To-Date appears, select **OK**.

If errors occur, determine which tables were changed and the changes that were made to the table definitions. Resolve these issues on a case-by-case basis as discussed earlier.

Validate all publications

You can validate all publications under a single Publication Database node in one operation.

> **Note**
>
> This validation feature is available only for publications using the trigger-based method of synchronization replication. This validation feature isn't available for publications using the log-based method of synchronization replication.

> **Note**
>
> In a multi-master replication system, publication tables in only the primary definition node are validated. The validation operation doesn't check if table definitions changed in other primary nodes.

1. Make sure the publication server whose node is the parent of the publications you want to validate is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. For SMR: Select the Publication Database node under which you want to validate all publications.

   For MMR: Select the Publication Database node representing the primary definition node.

3. Select **Publication > Validate All Publications**.

4. If there were no modified tables, select **OK**.

   If there were modified tables, a list of publications that contain the modified tables appears. Determine which tables changed and the changes made to the table definitions. Resolve these issues on a case-by-case basis as discussed earlier.

10.6.6 Removing a publication

In a single-master replication system, you can remove a publication before removing its associated subscriptions. See Removing a subscription for more information.

In a multi-master replication system, remove the publication from under the Publication Database node representing the primary definition node. Before you can remove a publication, you must remove all non-MDN nodes. See Removing a publication database for information about removing a publication database definition of a primary node.

Removing a publication doesn't delete the publication tables in the publication database. It removes the identity and association of these tables to Replication Server so the tables remain in the database until the DBA deletes them with the `DROP TABLE SQL` statement.

The publication database user name is also left intact along with some of the Replication Server metadata database objects. Shadow tables and triggers associated with the publication tables that were created by the publication server are deleted when you remove the publication. The remaining metadata database objects are deleted when you remove the publication database definition.

To remove a publication:

1. Make sure the publication server whose node is the parent of the publication you want to remove is running and registered in the Replication Server

console you're using. See Registering a publication server to learn how to start and register a publication server.

2. For SMR: Select the Publication node of the publication that you want to remove.

   For MMR: Select the Publication node under the Publication Database node representing the primary definition node.

3. Select **Publication > Remove Publication**.

4. Respond **Yes** in the Remove Publication confirmation box.

   The Publication node no longer appears under the Publication Database node.

10.6.7 Removing a publication database

Removing a publication database definition from Replication Server is equivalent to removing its Publication Database node.

If the Publication Database node you want to remove is currently designated as the controller database and there are additional publication databases in other single-master or multi-master replication systems, then you must first switch the controller database role to another publication database. See Switching the controller database for details.

If the Publication Database node you want to remove is the only remaining publication database (that is, there are no other single-master or multi-master replication systems), then this database can remain as the controller database. No other publication database is available to designate as the controller database. However, you must remove any existing subscription database definition before you remove the last Publication Database node.

In a single-master replication system, before you can remove a Publication Database node, you must remove all publications under that Publication Database node. See Removing a publication.

In a multi-master replication system, removing a Publication Database node representing a primary node (other than the primary definition node), eliminates that node's future participation in the replication system. Synchronization replications no longer involve tables in the removed primary node.

In a multi-master replication system, removing the Publication Database node representing the primary definition node removes the remaining metadata database objects of that particular multi-master replication system, effectively removing the multi-master replication system (except for the database objects comprising the publication tables).

Removing the Publication Database node representing the primary definition node entails the following steps:

- If the multi-master replication system is the only Replication Server replication system (that is, there are no single-master replication systems), then switch the controller database to the primary definition node if the designated controller database isn't currently the same database as the primary definition node.
- If there are one or more single-master replication systems in addition to the multi-master replication system, switch the controller database to a Postgres publication database of a single-master replication system. If none of the single-master publication databases is of type Postgres, and there are more than one Oracle or SQL Server publication databases, then you must create a Postgres publication database for a single-master replication system just for the purpose of designating it as the controller database.
- You must remove all Publication Database nodes representing non-MDN nodes. Repeat the process for each such primary node.
- You must remove the publication under the Publication Database node representing the primary definition node. See Removing a publication for details.
- Remove the Publication Database node representing the primary definition node.

Removing a Publication Database node doesn't delete the physical database from the database server. It removes the identity and association of the database to Replication Server. No further replications can originate from tables in the database unless there are other publication database definitions in Replication Server with the same host and database identifier. You can remove the physical database only by using the database management system's database removal procedures.

The publication database user name is also left intact.

All Replication Server metadata database objects created for that publication database definition are deleted.

For Oracle and SQL Server: All metadata database objects under the publication database user's schema are deleted.

For Postgres only: The schema `_edb_replicator_pub` and all of its database objects are deleted from the publication database.

To remove the Publication Database node and, equivalently, the publication database definition:

1. Make sure the publication server whose node is the parent of the publication database definition you want to remove is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. Select the Publication Database node that you want to remove.

3. Select **Publication > Publication Database > Remove Database**.

4. In the Remove Publication Database confirmation box, select **Yes**.

   The Publication Database node no longer appears under the Publication Server node.

10.7 Switching the controller database

The controller database is designated in the Replication Server Configuration file and determines the publication database to which the publication server and subscription server first connect on startup. See Controller database and Replication Server configuration file for more information.

You can't remove the current controller database from a replication system unless it's the last publication database remaining subordinate to the publication server. In other words, no other publication databases are managed by the publication server.

With more than one publication databases, if you want to remove the publication database currently designated as the controller database, you must first set another publication database as the controller. Then you can remove the publication database previously designated as the controller.

The publication database used as the controller can be the primary database of any single-master replication system or any primary node of a multi-master replication system. Any database type (Oracle, SQL Server, or Postgres) is acceptable as the controller database.

> **Note**
>
> If the controller database is an Oracle or a SQL Server publication database, you can't add a second Oracle or SQL Server publication database to create a second single-master replication system. For Replication Server to run more than one single-master replication systems consisting of Oracle or SQL Server publication databases, you must designate a Postgres publication database as the controller database.

Once you have multiple Oracle or SQL Server publication databases set up in single-master replication systems with a Postgres controller database, don't switch the controller database to an Oracle or SQL Server publication database.

Upon switching the controller database, the publication server updates the Replication Server configuration file so the parameters `user`, `password`, `host`, `port`, `database`, and `type` are set to the connection and authentication settings for the selected publication database.

To set another publication database as the controller database:

1. Make sure the publication server whose node is the parent of the publication databases is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

2. Select the Publication Database node corresponding to the publication database that you want to set as the controller database.

3. From the context menu, select **Set as Controller Database**.

4. In the Set as Controller Database confirmation box, select **Yes**.

The selected publication database is now set as the controller database.

The value **Yes** in the **Controller Database** field of the Property window indicates this database is the controller database.

The following shows the Replication Server configuration file after the controller database was switched to the primary node database `MMRnode_b`.

```
#xDB Replication Server Configuration
Properties
#Thu Oct 15 14:42:35 GMT-05:00
2015
port=5444
admin_password=ygJ9AxoJEX854elcVIJPTw\=\=
user=MMRuser
admin_user=admin
type=enterprisedb
database=MMRnode_b
password=ygJ9AxoJEX854elcVIJPTw\=\=
host=192.168.2.22
```

10.8 Replicating DDL changes

Once a replication system is created and in operation, there might be times when you need to change the publication table definitions. These data definition language (DDL) changes can include the following:

- Adding new columns to a table
- Renaming existing columns
- Modifying a column data type
- Modifying a column constraint
- Removing columns

> **Note**
>
> See Validating a publication for information on making other types of table definition changes.

Table definition changes are generally implemented using the `SQL ALTER TABLE` statement, which you issue in an SQL command line utility program such as PSQL.

The DDL change replication feature accepts one or more `ALTER TABLE` statements. You can provide the statements in a text file or by entering them into the Alter Publication Table dialog box. The DDL change replication feature then performs the following actions:

- Applies the `ALTER TABLE` statements to the appropriate target table in the publication and subscription databases of a single-master replication system or in all primary nodes (including the primary definition node) of a multi-master replication system
- For the trigger-based method of synchronization replication, modifies the insert/update/delete triggers that add data into the shadow table whenever a transaction occurs on the target table
- For the trigger-based method of synchronization replication, modifies the shadow table to accommodate the target table changes

The DDL change replication feature is supported for Oracle and SQL Server subscription databases as well as Postgres subscription databases. However, the publication database must always be a Postgres database.

The syntax of the `ALTER TABLE` statement accepted by the DDL change replication features is as follows:

```
ALTER TABLE schema.table_name <action>
```

<action> can be any of the following:

Rename an existing column:

```
RENAME [ COLUMN ] column_name TO
new_column_name
```

Add a column to the table:

```
ADD [ COLUMN ] column_name
data_type
[ DEFAULT dflt_expr
]
[ column_constraint_1 [ column_constraint_2 ]
...]
```

Drop a column from the table:

```
DROP [ COLUMN ] column_name [ RESTRICT
]
```

Change the data type of a column:

```
ALTER [ COLUMN ] column_name [ SET DATA ] TYPE
data_type
[ COLLATE "collation"
]
[ USING data_type_expr
]
```

Set the DEFAULT value of a column:

```
ALTER [ COLUMN ] column_name SET DEFAULT
dflt_expr
```

> **Note**
>
> The `SET DEFAULT` clause isn't supported when Oracle or SQL Server is the subscription database.

Drop the `DEFAULT` value of a column:

```
ALTER [ COLUMN ] column_name DROP
DEFAULT
```

> **Note**
>
> The `DROP DEFAULT` clause isn't supported when Oracle or SQL Server is the subscription database.

Set the column to reject null values:

```
ALTER [ COLUMN ] column_name SET NOT
NULL
```

> **Note**
>
> The `SET NOT NULL` clause isn't supported when SQL Server is the subscription database.

Allow the column to accept null values:

```
ALTER [ COLUMN ] column_name DROP NOT
NULL
```

> **Note**

> The `DROP NOT NULL` clause isn't supported when SQL Server is the subscription database.

The following restrictions apply to the manner in which you specify the `ALTER TABLE` statements. These restrictions apply whether you're entering the statements in a text file or in the dialog box.

- You must terminate each `ALTER TABLE` statement with a semicolon and begin each statement on a separate line.
- Although the Postgres `ALTER TABLE` statement allows multiple actions per statement, the Replication Server DDL change replication feature permits only one action per `ALTER TABLE` statement.
- The target table of all `ALTER TABLE` statements must be the same.
- You can't specify the `DROP COLUMN` action for a column that's part of the table's primary key.

Parameters

`schema`

The name of the schema containing `table_name`. This value is case sensitive.

`table_name`

The name of the table containing the column to add, modify, or drop. This value is case sensitive.

`column_name`

The name of the column to add, modify, or drop.

`new_column_name`

The new name of the column specified in the `RENAME COLUMN` clause.

`data_type`

The data type of the column.

`dflt_expr`

An expression for the default value of the column.

`column_constraint_n`

A column constraint such as a `UNIQUE` or `CHECK` constraint. For more information on column constraints see the `CREATE TABLE SQL` command in the PostgreSQL Core Documentation.

`RESTRICT`

In the `DROP COLUMN` clause, don't drop the column if objects depend on it. This is the default.

> **Note**
>
> You can't specify the CASCADE option as it isn't supported by the DDL change replication feature.

`collation`

Collation assigned to the column. If omitted, the column data type's default collation is used. Examples of collation are `default`, `C`, `POSIX`, `en_US`, `en_GB`, or `de_DE`.

`data_type_expr`

An expression specifying how to convert the column value with the new data type from the column value with the old data type. This expression can reference other columns in the same table. If omitted, the default conversion is an assignment cast from the old data type to the new data type.

Examples of ALTER TABLE statements

The following set of `ALTER TABLE` statements adds columns to the edb.emp table.

```
ALTER TABLE edb.emp ADD COLUMN gender CHAR(1) CHECK(gender IN
('M','F'));
ALTER TABLE edb.emp ADD COLUMN gradelevel VARCHAR2(4);
ALTER TABLE edb.emp ADD COLUMN title VARCHAR2(10);
```

The following `ALTER TABLE` statement changes the data type length of the title column and sets its values with the `USING data_type_expr` clause.

```
ALTER TABLE edb.emp
  ALTER COLUMN title SET DATA TYPE VARCHAR(25) USING
    CASE
job
      WHEN 'CLERK' THEN 'ADMINISTRATIVE ASSISTANT'
      WHEN 'ANALYST' THEN 'R & D
SPECIALIST'
      WHEN 'SALESMAN' THEN 'MARKETING REPRESENTATIVE'
      WHEN 'MANAGER' THEN 'SUPERVISOR'
      WHEN 'PRESIDENT' THEN 'CHIEF EXECUTIVE
OFFICER'
    END;
```

The following query shows the values assigned to the title column after the DDL change replication feature applies the preceding `ALTER TABLE` statement to the edb.emp table. This change to the title column and assignment of values occurs in all the subscription databases of a single-master replication system or in all the primary nodes of a multi-master replication system.

```
edb=# SELECT empno, ename, job, title FROM
emp;
```

```
 empno | ename  |    job    |          title
-------+--------+-----------+--------------------------
  7369 | SMITH  | CLERK     | ADMINISTRATIVE ASSISTANT
  7499 | ALLEN  | SALESMAN  | MARKETING REPRESENTATIVE
  7521 | WARD   | SALESMAN  | MARKETING REPRESENTATIVE
  7566 | JONES  | MANAGER   | SUPERVISOR
  7654 | MARTIN | SALESMAN  | MARKETING REPRESENTATIVE
  7698 | BLAKE  | MANAGER   | SUPERVISOR
  7782 | CLARK  | MANAGER   | SUPERVISOR
  7788 | SCOTT  | ANALYST   | R & D SPECIALIST
  7839 | KING   | PRESIDENT | CHIEF EXECUTIVE OFFICER
  7844 | TURNER | SALESMAN  | MARKETING REPRESENTATIVE
  7876 | ADAMS  | CLERK     | ADMINISTRATIVE ASSISTANT
  7900 | JAMES  | CLERK     | ADMINISTRATIVE ASSISTANT
  7902 | FORD   | ANALYST   | R & D SPECIALIST
  7934 | MILLER | CLERK     | ADMINISTRATIVE ASSISTANT
(14 rows)
```

The following set of `ALTER TABLE` statements drops the columns that were added in the first example.

```
ALTER TABLE edb.emp DROP COLUMN gender;
ALTER TABLE edb.emp DROP COLUMN
gradelevel;
ALTER TABLE edb.emp DROP COLUMN
title;
```

10.8.1 DDL change replication process

The DDL statement is executed in a controlled manner such that the target table is exclusively locked (by the default setting of configuration option `ddlChangeTableLock`) during the course of the operation. This is done to avoid loss of any transactions while the replication triggers and shadow table are modified by the DDL change replication process. Only one target table is locked at a time while DDL change replication takes place on that table, its triggers, and shadow table.

If there's a backlog of pending transactions, we recommend that you perform an explicit synchronization replication before performing DDL change replication. This approach avoids prolonging the DDL change replication process.

Perform DDL change replication when the OLTP rate is very low (near zero).

> **Note**
>
> You can turn off exclusive acquisition of each target table during the DDL change replication process by setting `ddlChangeTableLock` to `false`. However, make this change only when there are no write transactions taking place against the target table. Otherwise, transactions might not be recorded by the replication system. See DDL change replication table locking for more information on the `ddlChangeTableLock` configuration option.

The following is the series of steps that occur during the DDL change replication process.

- The publication server performs a health check across all databases in the replication system to ensure you can access them. If any database is not available, the DDL change replication process stops and notifies you.
- If the publication server configuration option `ddlChangeTableLock` is set to its default value of `true`, an exclusive table lock is requested on the table to which the DDL change is to be applied. If another application already has a lock on the table, there's a wait time of two minutes. After that the DDL change replication process stops if the lock isn't released before then. If `ddlChangeTableLock` is set to `false`, an exclusive table lock isn't requested.
- The DDL statement is executed against the target table. The replication triggers and shadow table are modified accordingly. If an error occurs, you're notified and the operation stops. If `ddlChangeTableLock` is set to `true`, the exclusive lock is released.
- These two actions are repeated on the target table for each database in the replication system.
- The in-memory table metadata definition is refreshed to reflect the DDL change. You're notified when the operation completes successfully.
- If an error occurs during these steps, any changes up to that point are rolled back so that the publication table, replication triggers, and shadow table revert to their original state prior to the start of this operation. If one or more databases goes down before the operation finishes, the publication is marked as dirty to avoid further replication events.

> **Note**
>
> When you execute the alterDDL command, non-MDN nodes shouldn't have any CDC changes during that window for MMR setup. If there are any CDC changes, it may result in data loss and a break in replication.

> **Note**
>
> The replicateDDL command performs an implicit synchronization operation that replicates any backlog changes before applying the DDL changes. When continuous data becomes available for replication, this operation may take a long time to complete.

10.8.2 DDL change replication using the Replication Server console

You can apply DDL change replication using the Replication Server console as follows:

1. If you plan to use a file to supply the `ALTER TABLE` statements to a publication table, prepare the text file. Make sure this text file is accessible by the operating system account with which you open the Replication Server console.

   Alternatively, you can copy and paste or directly type the `ALTER TABLE` statements into the Alter Publication Table dialog box without having to save the statements in a file.

2. Make sure the publication server whose node is the parent of the publication containing the table you want to change is running and registered in the Replication Server console you're using. See Registering a publication server to learn how to start and register a publication server.

3. Under the publication database of a single-master replication system, or under the primary definition node of a multi-master replication system, open the Alter Publication Table dialog box. From the context menu of the Table node of the table you want to modify, select **Alter Table**.

4. In the Alter Publication Table dialog box, if you saved the `ALTER TABLE` statements in a text file, make sure the DDL Script File option is selected, browse for this file, and select **OK**.

   Alternatively, if you are directly entering the `ALTER TABLE` statements, select the **DDL Script** option instead of the **DDL Script File** option. Directly type or copy and paste the `ALTER TABLE` statements from your source into the text box and then selecting **OK**.

5. When DDL Replicated Successfully appears, the DDL change was successful across all databases. Select **OK**.

If DDL replication wasn't successful, you must investigate and resolve the problem on a case-by-case basis. Factors to look for include the following:

- Were the modifications in the `ALTER TABLE` statements successfully applied to the target table in each database of the replication system?
- For the trigger-based method, were the replication triggers on the target table modified to account for the `ALTER TABLE` statements?
- For the trigger-based method, was the shadow table `RRST_schema_table` located in the `_edb_replicator_pub` schema in each database of the replication system modified to account for the `ALTER TABLE` statements?

If it is apparent that the replication system isn't in a consistent state regarding the table definitions, seeValidating a publication for guidance on how to deal with these issues.

10.9 Loading tables from an external data source (offline snapshot)

You might want to initially load your target tables (subscription tables of a single-master replication system, or non-MDN nodes of a multi-master replication system) using a method other than the snapshot replication functionality of Replication Server. This method is referred to as using an offline snapshot.

For example, you might at first load the tables by running the Migration Toolkit from the command line or by using a backup from an external data source. When you load the target tables using an offline snapshot, you must take into account special preparations for the following deviations from the default target table creation and loading process:

- In the typical default scenario, Replication Server creates the target table definitions when you define the subscription in a single-master replication system or add another primary node in a multi-master replication system. When using an offline snapshot, creating the target table definitions is your responsibility. You must therefore prevent Replication Server from creating the target table definitions.
- In the typical default scenario, Replication Server performs synchronization replication using batches of SQL statements. If any statement in a batch results in an error, all statements in the batch are rolled back. When using an offline snapshot, the external data source used to load the target tables might already have transactions applied to it that are also recorded in the shadow tables of the source tables. In this case, you must perform the first synchronization replication in non-batch mode. This condition can result in a statement failure in certain cases.

> **Note**
>
> You can use Migration Toolkit in parallel with Replication Server only to perform offline snapshots. We don't recommend using Migration Toolkit in parallel with Replication Server for online snapshots.

Non-batch mode synchronization

Synchronization replications are done in batches of updates, each batch committed in a separate transaction. If any single update in a batch fails, all the updates in the batch are rolled back.

This process has the following implications.

Prior to and during the time when the offline snapshot is in progress, there might be updates to the source tables, which are recorded in the source tables' shadow tables. After the offline snapshot completes, there might be more updates to the source tables that are also recorded in the shadow tables.

Since Replication Server does not know about the external data source used to load the target tables, Replication Server doesn't know whether any of the updates made to the source tables during or after the offline snapshot were already included in the data used to load the target tables.

As a result, the shadow tables might include a mixture of duplicate updates that were already applied to the target tables as well as new updates that weren't applied to the target tables.

If you then perform synchronization replication, the publication server attempts to apply all updates recorded in the shadow tables in batches.

If one of the updates was inserting a new row and this new row is already in the target table loaded from the offline snapshot, a duplicate key error results when the publication server attempts to apply the batch containing the `INSERT` statement for this row. The duplicate key error forces the rollback of the entire batch. This causes the exclusion of updates in the batch that might not yet have been carried over to the target tables. The source tables and target tables are now inconsistent, since there were updates to the source tables that weren't applied to the target tables.

> **Note**
>
> The effects of applying `UPDATE` and `DELETE` statements in the batch to a target table that was already changed by these updates doesn't cause the same problem as repeated application of `INSERT` statements. The `UPDATE` statement changes the row to the same values a second time. When a `DELETE` statement doesn't affects any rows, this isn't considered an error by the database server. No rollback of the batch occurs.

The solution to the potential rollback of a batch is to apply the shadow table updates in non-batch mode. That is, commit each SQL statement individually. In that way, if inserting a row fails due to a duplicate key error, that statement alone is rolled back. The error doesn't affect the other shadow table updates that must be applied since all updates are enclosed in their own, individual transactions.

The `batchInitialSync` configuration option controls whether the first synchronization replication occurs in batch or non-batch mode. Suppose you're using an offline snapshot in an active replication system where updates are occurring to the source tables. Transactions are thus accumulating in the shadow tables for the trigger-based method. In this case, we recommend setting `batchInitialSync` to `false` to perform the first synchronization replication in non-batch mode.

> **Note**
>
> You can't use an offline snapshot to add a subscription or a primary node to an active replication system that uses the log-based method. For the log-based method, you can use offline snapshots only to first configure the system. You can't use them to update the system with additional nodes after the publication database or primary node is actively receiving transactions.

Suppose you're using offline snapshots to first create the entire replication system that has yet to be activated. The content of the offline snapshots are all assumed to be consistent for the source and target tables. You can then leave `batchInitialSync` with its default setting of `true`. It is assumed that the first synchronization replication will not apply any duplicate updates.

Offline snapshot configuration options

The following are the configuration options that you need to modify when using an offline snapshot.

> **Note**
>
> These options apply only to the publication server.

`offlineSnapshot`

You must set the `offlineSnapshot` to `true` before creating the subscription for a single-master replication system or before adding the primary node for a multi-master replication system.

`offlineSnapshot={true | false}`

The default value is `false`.

When set to `true`, the `offlineSnapshot` option prevents the usual creation of the subscription schema and table definitions when the subscription is defined in a single-master replication system. It is assumed that you're creating the subscription table definitions and loading them from an external source other than the publication.

When adding the primary node in a multi-master replication system, leave the **Replicate Publication Schema** and **Perform Initial Snapshot** boxes cleared (see Creating more primary nodes).

When `offlineSnapshot` is set to `true`, this configuration has the direct effect in the control schema by setting column `has_initial_snapshot` to a value of `0`. This setting indicates an offline snapshot is used for the target subscription or primary node represented by the row. Column `has_initial_snapshot` is set in table _edb_replicator_pub.xdb_publication_subscriptions for a single-master replication system and in table _edb_replicator_pub.xdb_mmr_pub_group for a multi-master replication system.

After the first replication completes to the target subscription or primary node, `has_initial_snapshot` is changed to `Y` by Replication Server.

The setting of `has_initial_snapshot` influences the behavior of the `batchInitialSync` option.

`batchInitialSync`

The `batchInitialSync` option controls whether the first synchronization after loading the target tables from an offline snapshot is done in batch mode (the default) or non-batch mode.

Set the `batchInitialSync` option to `false` to perform synchronization replication in non-batch mode.

You must set the `offlineSnapshot` configuration option to `true` prior to creating the subscription or adding another primary node. A non-batch mode synchronization occurs only if `batchInitialSync` is `false` and the `has_initial_snapshot` column in the control schema is set to a value of `0` as described for the `offlineSnapshot` option.

`batchInitialSync={true | false}`

The default value is `true`.

Single-master replication offline snapshot

You can use an offline snapshot to first load the subscription tables of a single-master replication system. For a publication that's intended to have multiple subscriptions, you can create some of the subscriptions using the default Replication Server snapshot replication process as described in Performing snapshot replication. You can create other subscriptions from an offline snapshot.

To create a subscription from an offline snapshot:

1. Register the publication server, add the publication database definition, and create the publication as described in Creating a publication.

2. Register the subscription server and add the subscription database definition as described in Registering a subscription server and Adding a subscription database.

> **Note**
>
> You must perform steps 3 and 4 before creating the subscription. You can repeat steps 5 through 9 each time you want to create another subscription from an offline snapshot.

3. Modify the publication server configuration file if these options aren't already set as described by the following:

   - Change the `offlineSnapshot` option to `true`. When you restart the publication server or reload the publication server's configuration via reloadconf, `offlineSnapshot` set to `true` has two effects. One is that creating a subscription doesn't create the schema and subscription table definitions in the subscription database as is done with the default setting. The other is that creating a subscription sets a column in the control schema indicating an offline snapshot is used to load this subscription.
   - Set the `batchInitialSync` option to the appropriate setting for your situation as discussed at the end of Non-batch mode synchronization.

4. If you modified the publication server configuration file in Step 3, reload configuration of the publication server. See "Reloading the Publication or Subscription Server Configuration File (reloadconf)" for directions on reloading the publication server's configuration.

5. In the subscription database, create the schema and the subscription table definitions, and load the subscription tables from your offline data source. The subscription database user name used in Adding a subscription database must have full privileges over the database objects created in this step. Also review the beginning of Adding a subscription database regarding the rules as to how Replication Server creates the subscription definitions from the publication for each database type. You must follow these same conventions when you create the target definitions manually.

6. Add the subscription as described in Adding a subscription.

7. Perform an on-demand synchronization replication. See Performing synchronization replication to learn how to perform an on-demand synchronization replication.

8. If you aren't planning to load any other subscriptions using an offline snapshot at this time, change the `offlineSnapshot` option back to `false` and the `batchInitialSync` option to `true` in the publication server configuration file.

9. If you modified the publication server configuration file in step 8, reload configuration of the publication server.

Multi-master replication offline snapshot

You can use an offline snapshot to first load the primary nodes of a multi-master replication system. You can load some of the primary nodes using the Replication Server snapshot replication functionality when defining the primary node as described in Creating more primary nodes or by using an on-demand snapshot as described in Performing snapshot replication. You can load other primary nodes from an offline snapshot.

> **Note**
>
> Offline snapshots aren't supported for a multi-master replication system that's actively in use. Any changes on an active primary node are lost during the offline snapshot process of dumping or restoring the data of another node.

To create a primary node from an offline snapshot:

1. Register the publication server, add the primary definition node, and create the publication as described in Creating a publication.

   > **Note**
   >
   > You must perform the steps 3 and 4 before adding a primary node to be loaded by an offline snapshot. You can repeat Steps 5 through 10 each time you want to create another primary node from an offline snapshot.

2. Be sure there's no schedule defined on the replication system. If there is, remove the schedule until you complete this process. See Removing a schedule for details.

3. Modify the publication server configuration file so the options are set as follows:

    - Set the `offlineSnapshot` option to `true`. When you restart the publication server or reload the publication server's configuration via reloadconf, this setting has the effect that adding a primary node sets a column in the control schema indicating an offline snapshot is used to load this primary node.
    - Set the `batchInitialSync` option to the appropriate setting for your situation as discussed at the end of Non-batch mode synchronization.

4. If you modified the publication server configuration file in step 3, reload configuration of the publication server. See "Reloading the Publication or Subscription Server Configuration File (reloadconf)" for directions to reload the publication server's configuration.

5. In the database to use as the new primary node, create the schema and the table definitions, and load the tables from your offline data source.

6. Add the primary node as described in Creating more primary nodes with the options **Replicate Publication Schema** and **Perform Initial Snapshot** cleared.

7. Perform an initial on-demand synchronization. See Performing synchronization replication to learn how to perform an on demand-synchronization.

8. If you aren't planning to load any other primary nodes using an offline snapshot at this time, change the `offlineSnapshot` option back to `false` and the `batchInitialSync` option to `true` in the publication server configuration file.

9. If you modified the publication server configuration file in step 8, reload configuration of the publication server.

10. Add the schedule again if you removed it. See Creating a schedule to learn how to create a schedule.

10.10 Replicating Postgres partitioned tables

Both PostgreSQL and EDB Postgres Advanced Server support partitioned tables, which you can replicate with Replication Server in either a single-master or multi-master replication system.

The following are the partitioning techniques:

- EDB Postgres Advanced Server partitioning compatible with Oracle databases
- Postgres declarative partitioning (applies to both PostgreSQL and EDB Postgres Advanced Server version 10 and later)
- Postgres table inheritance (applies to both PostgreSQL and EDB Postgres Advanced Server)

If you're using EDB Postgres Advanced Server, you can create partitioned tables using the `CREATE TABLE` statement with partitioning syntax compatible with Oracle databases. For information on partitioning compatible with Oracle databases, see Table partitioning.

If you're using version 10 or later of PostgreSQL or EDB Postgres Advanced Server, you can use declarative partitioning to create partitioned tables. The `CREATE TABLE` syntax for creating a declarative partitioned table is similar to the partitioning compatible with Oracle databases. However, you must create the individual partitions of the declarative partitioned table separately with their own `CREATE TABLE` statements.

For information on declarative partitioning and table inheritance, see the PostgreSQL core documentation.

Regardless of the partitioning method, the resulting partitioned table is made up of a parent table with a set of child tables.

You can accomplish replication of these Postgres partitioned tables in a single-master or multi-master replication system in the same manner.

Note the following general restrictions when the publication contains a partitioned table:

- You can't use SQL Server as a subscription database.
- When using table inheritance, the subscription databases must be Postgres. They can't be Oracle or SQL Server.

All three partitioning techniques are shown on the emp table. The partitioned table is then used in a publication of a multi-master replication system in the following sections:

- For creating a publication in Postgres 9.x, see Creating a Postgres 9.x partitioned table publication.
- For creating a publication in Postgres 10 or later, see Creating a Postgres version 10 or later partitioned table publication.

The following creates the partitioned table in EDB Postgres Advanced Server using partitioning compatible with Oracle databases:

```
CREATE TABLE emp
(
    empno           NUMERIC(4) PRIMARY KEY,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
NUMERIC(4),
    hiredate
DATE,
    sal
NUMERIC(7,2),
    comm            NUMERIC(7,2),
    deptno
NUMERIC(2)
)
PARTITION BY LIST(deptno)
(
    PARTITION dept_10 VALUES
(10),
    PARTITION dept_20 VALUES
(20),
    PARTITION dept_30 VALUES
(30)
);
--  Load the 'emp'
table
--
INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'17-DEC-
80',800,NULL,20);
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-
81',1600,300,30);
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'22-FEB-
81',1250,500,30);
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'02-APR-
81',2975,NULL,20);
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'28-SEP-
81',1250,1400,30);
INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,'01-MAY-
81',2850,NULL,30);
INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,'09-JUN-
81',2450,NULL,10);
INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,'19-APR-
87',3000,NULL,20);
INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,'17-NOV-
81',5000,NULL,10);
INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,'08-SEP-
81',1500,0,30);
INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,'23-MAY-
87',1100,NULL,20);
INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,'03-DEC-
81',950,NULL,30);
INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,'03-DEC-
81',3000,NULL,20);
INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,'23-JAN-
82',1300,NULL,10);
```

The following creates the partitioned table in PostgreSQL or EDB Postgres Advanced Server 10 or later using declarative partitioning:

Note

When creating a declarative partitioned table to replicate using Replication Server, you must include the `PRIMARY KEY` in the `CREATE TABLE` statements of the individual partitions, not in the `CREATE TABLE` statement of the parent table being partitioned.

```
CREATE TABLE emp
(
    empno           NUMERIC(4),
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
NUMERIC(4),
    hiredate
DATE,
    sal
NUMERIC(7,2),
    comm            NUMERIC(7,2),
    deptno
NUMERIC(2)
)
PARTITION BY LIST(deptno);
--
--  Create the
partitions
--
--  The partitions must contain the PRIMARY KEY
constraint
--
CREATE TABLE emp_dept_10 PARTITION OF emp (empno PRIMARY
KEY)
    FOR VALUES IN
(10);
CREATE TABLE emp_dept_20 PARTITION OF emp (empno PRIMARY
KEY)
    FOR VALUES IN
(20);
CREATE TABLE emp_dept_30 PARTITION OF emp (empno PRIMARY
KEY)
    FOR VALUES IN
(30);
--
--  Load the 'emp'
table
--
INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'17-DEC-
80',800,NULL,20);
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-
81',1600,300,30);
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'22-FEB-
81',1250,500,30);
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'02-APR-
81',2975,NULL,20);
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'28-SEP-
81',1250,1400,30);
INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,'01-MAY-
81',2850,NULL,30);
INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,'09-JUN-
81',2450,NULL,10);
INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,'19-APR-
87',3000,NULL,20);
INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,'17-NOV-
81',5000,NULL,10);
INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,'08-SEP-
81',1500,0,30);
INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,'23-MAY-
87',1100,NULL,20);
```

```
INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,'03-DEC-
81',950,NULL,30);
INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,'03-DEC-
81',3000,NULL,20);
INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,'23-JAN-
82',1300,NULL,10);
```

The following creates the partitioned table in PostgreSQL or EDB Postgres Advanced Server using table inheritance:

```
--
--  Create the parent
table
--
CREATE TABLE emp
(
    empno           NUMERIC(4) NOT NULL CONSTRAINT emp_pk PRIMARY
KEY,
    ename           VARCHAR(10),
    job
VARCHAR(9),
    mgr
NUMERIC(4),
    hiredate
DATE,
    sal
NUMERIC(7,2),
    comm            NUMERIC(7,2),
    deptno
NUMERIC(2)
);
--
--  Create the child
tables
--
CREATE TABLE emp_dept_10
(
    CHECK (deptno = 10)
) INHERITS
(emp);
CREATE TABLE emp_dept_20
(
    CHECK (deptno = 20)
) INHERITS
(emp);
CREATE TABLE emp_dept_30
(
    CHECK (deptno = 30)
) INHERITS
(emp);

ALTER TABLE emp_dept_10 ADD CONSTRAINT emp_dept_10_pk PRIMARY KEY
(empno);
ALTER TABLE emp_dept_20 ADD CONSTRAINT emp_dept_20_pk PRIMARY KEY
(empno);
ALTER TABLE emp_dept_30 ADD CONSTRAINT emp_dept_30_pk PRIMARY KEY
(empno);
--
--  Create the trigger function to insert into the proper child by
deptno
--
CREATE OR REPLACE FUNCTION emp_insert_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.deptno = 10 THEN
        INSERT INTO emp_dept_10 VALUES
(NEW.*);
    ELSIF NEW.deptno = 20 THEN
```

```
        INSERT INTO emp_dept_20 VALUES
(NEW.*);
    ELSIF NEW.deptno = 30 THEN
        INSERT INTO emp_dept_30 VALUES
(NEW.*);
    ELSE
        RAISE EXCEPTION 'Department # out of
range.';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE
plpgsql;
--
--  Create the
trigger
--
CREATE TRIGGER insert_emp_trigger
    BEFORE INSERT ON
emp
    FOR EACH ROW EXECUTE PROCEDURE
emp_insert_trigger();
--
--  Load the 'emp'
table
--
INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'17-DEC-
80',800,NULL,20);
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'20-FEB-
81',1600,300,30);
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'22-FEB-
81',1250,500,30);
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'02-APR-
81',2975,NULL,20);
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'28-SEP-
81',1250,1400,30);
INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,'01-MAY-
81',2850,NULL,30);
INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,'09-JUN-
81',2450,NULL,10);
INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,'19-APR-
87',3000,NULL,20);
INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,'17-NOV-
81',5000,NULL,10);
INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,'08-SEP-
81',1500,0,30);
INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,'23-MAY-
87',1100,NULL,20);
INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,'03-DEC-
81',950,NULL,30);
INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,'03-DEC-
81',3000,NULL,20);
INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,'23-JAN-
82',1300,NULL,10);
```

The following shows the types of SQL queries that you can make on the parent and child tables to show which tables contain the rows.

Querying the parent table, emp, with the asterisk appended to the table name in the `SELECT statement`, shows the rows in the parent and child tables. This is the default behavior if the asterisk is omitted.

```
edb=# SELECT * FROM
emp*;
```

```
 empno | ename  |    job    | mgr  |      hiredate      |   sal   |  comm   | deptno
-------+--------+-----------+------+--------------------+---------+---------+--------
```

```
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00 | 2450.00 |         |     10
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00 | 5000.00 |         |     10
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00 | 1300.00 |         |     10
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00 |  800.00 |         |     20
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00 | 2975.00 |         |     20
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00 | 3000.00 |         |     20
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00 | 1100.00 |         |     20
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00 | 3000.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00 | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00 | 1250.00 |  500.00 |     30
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00 | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00 | 2850.00 |         |     30
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00 | 1500.00 |    0.00 |     30
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00 |  950.00 |         |     30
(14 rows)
```

The following queries show how the rows are physically divided among the child tables. The use of the `ONLY` keyword results in rows only in the specified table of the `SELECT` statement and not from any of its children.

```
edb=# SELECT * FROM ONLY emp;
```

```
 empno | ename | job | mgr | hiredate | sal | comm | deptno
-------+-------+-----+-----+----------+-----+------+--------
(0 rows)
```

```
edb=# SELECT * FROM ONLY emp_dept_10;
```

```
 empno | ename  |    job    | mgr  |      hiredate       |   sal   | comm | deptno
-------+--------+-----------+------+---------------------+---------+------+--------
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00 | 2450.00 |      |     10
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00 | 5000.00 |      |     10
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00 | 1300.00 |      |     10
(3 rows)
```

```
edb=# SELECT * FROM ONLY emp_dept_20;
```

```
 empno | ename |   job   | mgr  |      hiredate       |   sal   | comm | deptno
-------+-------+---------+------+---------------------+---------+------+--------
  7369 | SMITH | CLERK   | 7902 | 17-DEC-80 00:00:00 |  800.00 |      |     20
  7566 | JONES | MANAGER | 7839 | 02-APR-81 00:00:00 | 2975.00 |      |     20
  7788 | SCOTT | ANALYST | 7566 | 19-APR-87 00:00:00 | 3000.00 |      |     20
  7876 | ADAMS | CLERK   | 7788 | 23-MAY-87 00:00:00 | 1100.00 |      |     20
  7902 | FORD  | ANALYST | 7566 | 03-DEC-81 00:00:00 | 3000.00 |      |     20
(5 rows)
```

```
edb=# SELECT * FROM ONLY emp_dept_30;
```

```
 empno | ename  |   job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+----------+------+---------------------+---------+---------+--------
  7499 | ALLEN  | SALESMAN | 7698 | 20-FEB-81 00:00:00 | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN | 7698 | 22-FEB-81 00:00:00 | 1250.00 |  500.00 |     30
  7654 | MARTIN | SALESMAN | 7698 | 28-SEP-81 00:00:00 | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER  | 7839 | 01-MAY-81 00:00:00 | 2850.00 |         |     30
  7844 | TURNER | SALESMAN | 7698 | 08-SEP-81 00:00:00 | 1500.00 |    0.00 |     30
  7900 | JAMES  | CLERK    | 7698 | 03-DEC-81 00:00:00 |  950.00 |         |     30
(6 rows)
```

Creating a Postgres version 10 or later partitioned table publication shows creating the publication when using partitioning compatible with Oracle

databases or declarative partitioning on a Postgres 10 or later database server.

Creating a Postgres version 10 or later partitioned table publication

Create the publication using either partitioning compatible with Oracle databases or Postgres declarative partitioning on a Postgres 10 or later database server.

> **Note**
>
> If you're using table inheritance, you must use the process described in Creating a Postgres 9.x partitioned table publication even when creating the publication on a Postgres 10 or later database server.

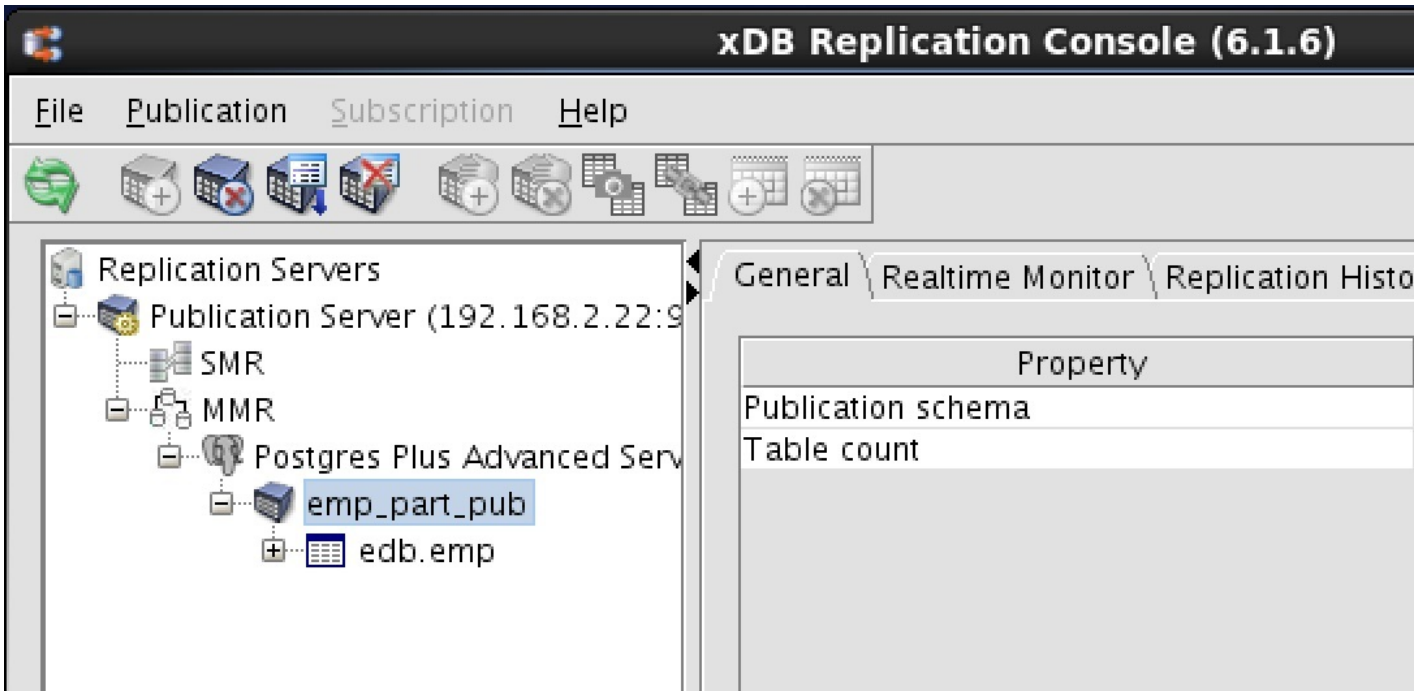
The following restrictions apply when the publication contains a table with partitioning compatible with Oracle databases or declarative partitioning:

- You must select the log-based method of synchronization replication for the publication database. You can't use the trigger-based method.
- In a single-master replication system, the subscription databases must be Postgres version 10 or later. You can't use Oracle and SQL Server as a subscription database.
- In a multi-master replication system, all primary nodes must be Postgres version 10 or later with the same compatibility mode as the primary definition node (that is, either compatible with native PostgreSQL or compatible with Oracle databases). For more information on the multi-master replication system compatibility modes, see Permitted MMR database server configurations.

Follow the steps in Creating a publication to create a primary definition node along with a publication containing the partitioned table. (For a single-master replication system, create the publication database along with the publication following the steps in Creating a publication.)

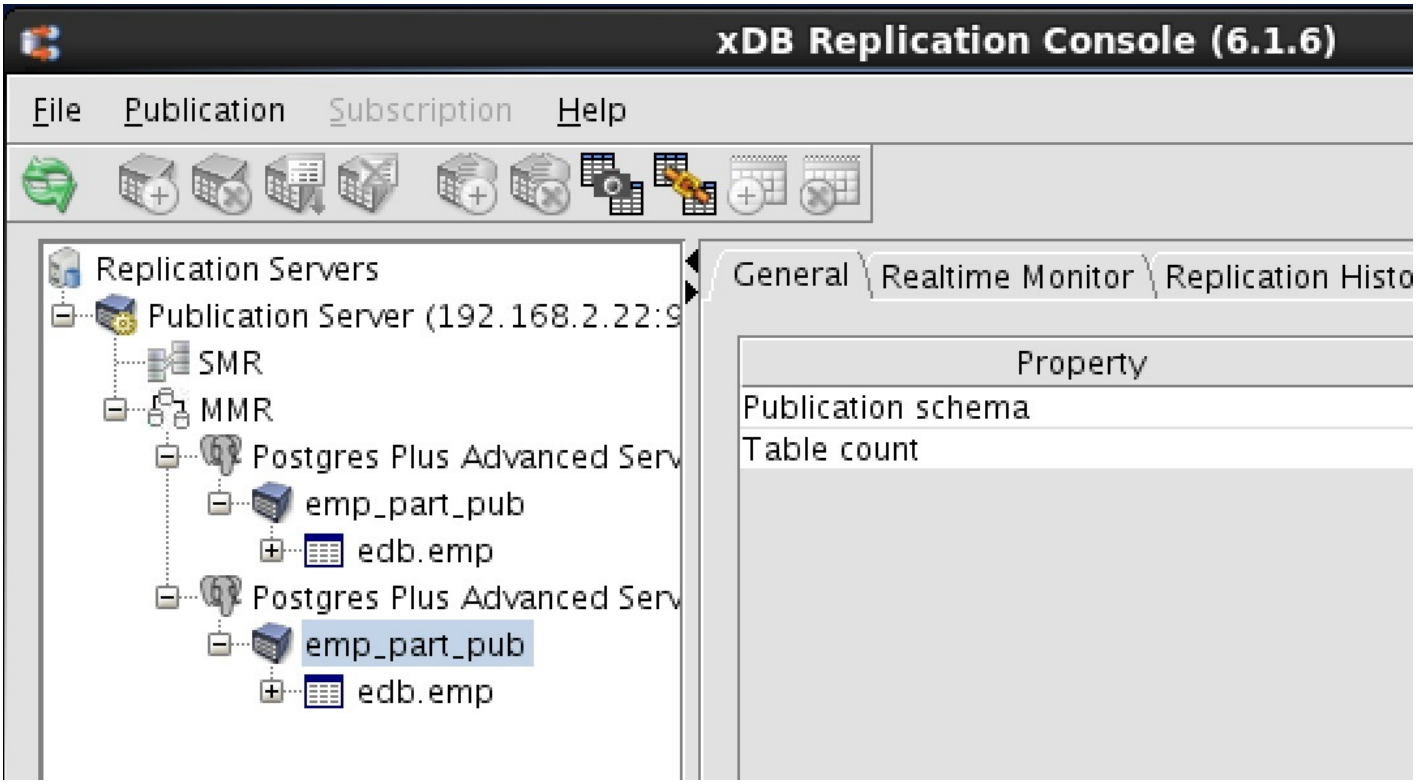
When creating the publication, only the parent table appears and is selected.

The following shows the resulting replication tree for the partitioned table in the primary definition node:



Create more primary nodes as described in Creating more primary nodes. (For a single-master replication system, create the subscription database and subscription following the steps in Creating a subscription.)

The following shows the resulting multi-master replication system after you add another primary node.



You can now keep the partitioned table synchronized on the primary nodes of the multi-master replication system.

10.11 Using secure sockets layer (SSL) connections

You can make publication server and subscription server connections to Postgres publication databases, Postgres subscription databases, and Postgres primary nodes using secure sockets layer (SSL) connectivity.

Replication Server doesn't support SSL connections to Oracle and SQL Server databases used in any Replication Server replication system.

For a single-master replication system, you can make the following connections to Postgres databases enabled with SSL:

- Publication server connection to the publication database and to the subscription databases
- Subscription server connection to the subscription databases
- Migration Toolkit connection to the publication and subscription databases

For a multi-master replication system, you can make the following connections to Postgres databases enabled with SSL:

- Publication server connection to the primary definition node and the non-MDN nodes
- Migration Toolkit connection to the primary definition node and the non-MDN nodes

> **Note**
>
> SSL connections aren't used from the Replication Server console or CLI. The Replication Server user interfaces communicate with the publication server and subscription server, which in turn connect to the publication/subscription databases or primary nodes.

> **Note**
>
> The Migration Toolkit connection using SSL occurs in the context of the publication server and subscription server SSL connections. Therefore, you don't need to perform separate steps for the Migration Toolkit SSL connection.

Using SSL requires various prerequisite configuration steps. Perform these steps on the database servers involved with the SSL connections and on the

Java truststore and keystore on the hosts running the publication server and subscription server.

The Java truststore is the file containing the Certificate Authority (CA) certificates. The Java client (the publication server and subscription server) use this certificate to verify the authenticity of the server to which it is starting an SSL connection.

The Java keystore is the file containing private and public keys and their corresponding certificates. The keystore is required for client authentication to the server, which is used for Replication Server SSL connections.

The following is material you can refer to for guidance in setting up the SSL connections:

- See the section on secure TCP connections with SSL in the PostgreSQL Core Documentation for information on setting up SSL connectivity to Postgres database servers.
- For information on JDBC client connectivity using SSL, see the section on configuring the client in thePostgreSQL JDBC Interface documentation.

The following sections provide more information for the configuration steps of using SSL with the Replication Server.

- Configuring SSL on a Postgres database server
- Configuring SSL on a JDBC client for the publication and subscription servers (Configuring SSL for the publication server and subscription server)
- Requesting SSL connection to the Replication Server databases

Configuring SSL on a Postgres database server

This example shows configuring SSL on a Postgres database server to show the use of SSL with Replication Server. A self-signed certificate is used for this purpose.

1. Create the certificate signing request (CSR).

   In the following example, the generated certificate signing request file is `server.csr`. The private key is generated as file server.key.

   ```
   $ openssl req -new -text -nodes -subj
   '/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=enterpr
   edb' -keyout server.key -out server.csr
   Generating a 1024 bit RSA private key
   ..........................................................++++++
   .++++++
   writing new private key to 'server.key'
   -----
   ```

   > **Note**
   >
   > When creating the certificate, the value specified for the common name field (designated as `CN=enterprisedb` in this example) must be the database user name that is specified in the **User** field of the Add Database or Update Database dialog box used when defining the publication database, subscription database, or primary nodes. See Adding a publication database), Adding a subscription database), Adding the primary definition node), and Creating more primary nodes).

   Alternatively, you can use user name maps as defined in the `pg_ident.conf` file to permit more flexibility for the common name and database user name. Steps 8 and 9 describe the use of user name maps.

   There are two options when entering the value for the common name field: the database user name and host name/ip-adress. If entering `CN=hostname/ip-address`, use the following command.

   - Create root Certificate Authority (CA) private key (`root.key`) and root CA certificate (`root.crt`) files with these commands:

```
 $ openssl genrsa -des3 -out root.key 4096
Generating RSA private key, 4096 bit long modulus
........++
........++
e is 65537 (0x10001)
Enter pass phrase for root.key:
Verifying - Enter pass phrase for root.key:

$ openssl req -x509 -new -nodes -key root.key -subj
'/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=192
168.22.174' -sha256 -days 1024 -out root.crt
Enter pass phrase for root.key:
$ openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.....................................................................................+++
e is 65537 (0x10001)

Note: Please note in below command database server ip-address which is given as CN(Common Name)
value for server.csr(server certificate signing request) file generation could not be identical
as you have used above while generating root.crt file. It must be different.

$ openssl req -new -sha256 -key server.key -subj
'/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=192
168.18.2' -out server.csr
```

2. Generate the self-signed certificate.

   The following generates a self-signed certificate to file `server.crt` using the certificate signing request file, `server.csr`, and the private key, `server.key`, as input.

```
 $ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=
terprisedb
Getting Private key
```

   The following command generates a server certificate with `CN=hostname/ip-address` to file `server.crt` using the certificate signing request file, `server.csr`, and the private root key, `root.key`, as input.

```
 $ openssl x509 -req -in server.csr -CA root.crt -CAkey root.key -CAcreateserial -out server.crt -days
1024 -sha256
Signature ok
subject=/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=
calhost.localdomain
Getting CA Private Key
Enter pass phrase for root.key:
```

3. Make a copy of the server certificate (`server.crt`) to use as the root Certificate Authority (CA) file (`root.crt`).

```
 $ cp server.crt root.crt
```

   If following the instructions specific to `CN=hostname/ip-address`, make a copy of the `root.crt` file from Step 1.

4. Delete the now redundant certificate signing request (`server.csr`).

```
$ rm server.csr
```

5. Move or copy the certificate and private key files to the Postgres database server data directory, `POSTGRES_INSTALL_HOME/data`.

```
$ mv root.crt /var/lib/edb/as14/data
$ mv server.crt /var/lib/edb/as14/data
$ mv server.key /var/lib/edb/as14/data
```

6. Set the file ownership and permissions on the certificate files and private key file.

   Set the ownership to the operating system account that owns the data subdirectory of the Postgres database server, which is either `enterprisedb` or `postgres` depending upon the installation mode you chose when you installed your Postgres database server (Oracle compatible or PostgreSQL compatible).

```
$ chown enterprisedb root.crt server.crt server.key
$ chgrp enterprisedb root.crt server.crt server.key
$ chmod 600 root.crt server.crt server.key
$ ls -l
total 140
        .
        .
        .
-rw------- 1 enterprisedb enterprisedb  1346 Mar 15 09:31 root.crt
-rw------- 1 enterprisedb enterprisedb  1346 Mar 15 09:30 server.crt
-rw------- 1 enterprisedb enterprisedb  1704 Mar 15 09:28 server.key
```

7. In the `postgresql.conf` file, make the following changes.

```
ssl = on                                # (change requires
restart)
ssl_cert_file = 'server.crt'           # (change requires
restart)
ssl_key_file = 'server.key'            # (change requires
restart)
ssl_ca_file = 'root.crt'               # (change requires
restart)
```

8. Modify the `pg_hba.conf` file to enable SSL usage on the desired publication, subscription, or primary node databases.

   In the `pg_hba.conf` file, the `hostssl` type indicates the entry is used to validate SSL connection attempts from the client (the publication server and the subscription server).

   The authentication method is set to cert with the option `clientcert=1` to require an SSL certificate from the client against which authentication is performed using the common name of the certificate (`enterprisedb` in this example).

   The `map=sslusers` option specifies to use a mapping named `sslusers` defined in the `pg_ident.conf` file for authentication. This mapping allows a connection to the database if the common name from the certificate and the database user name attempting the connection match the `SYSTEM-USERNAME/PG-USERNAME` pair listed in the `pg_ident.conf` file.

   The following is an example of the settings in the `pg_hba.conf` file if the publication and subscription databases (edb and subnode) must use SSL connections.

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     md5

# IPv4 local connections for for Postgres v13:
```

```
hostssl   edb,subnode   all  192.168.2.0/24   cert clientcert=verify-ca map=sslusers

# IPv4 local connections for for Postgres v14:
hostssl   edb,subnode   all  192.168.2.0/24   cert clientcert=verify-full map=sslusers
```

9. The following shows the user name maps in the `pg_ident.conf` file related to the `pg_hba.conf` file by the `map=sslusers` option. These user name maps permit you to specify database user names pubuser, subuser, MMRuser, or enterprisedb in the **User** field of the Add Database or Update Database dialog box when adding the publication, subscription, or primary node databases in the EPRS Replication Console.

   In other words, these are the permitted set of database user names that can be used by the publication server and subscription server to connect to the publication, subscription, or primary node databases.

```
 # MAPNAME       SYSTEM-USERNAME         PG-USERNAME
sslusers        enterprisedb            pubuser
sslusers        enterprisedb            subuser
sslusers        enterprisedb            MMRuser
sslusers        enterprisedb            enterprisedb
```

10. Restart the Postgres database server after you make the changes to the Postgres configuration files.

Configuring SSL for the publication server and subscription server

After you configure SSL on the Postgres database server, the following steps provide an example of generating a certificate and keystore file for the publication server and subscription server (the JDBC clients).

Before you begin, configure the client for SSL with trigger mode.

- If you are using PostgreSQL, on the SSL-enabled Postgres database server:

  Make the following client/cert files available on the publication/subscription server using an SSL connection:

  - `postgresql.crt`
  - `postgresql.pk8`
  - `root.crt`

  In our example, we use the copy of this self-signed certificate and key generated for the database server on the client side.

  The default location of these files is `{user.home}/.postgresql(e.g/var/lib/edb/.postgresql/)`. The file location can be overridden using SSL connection parameters or Postgres SSL environmental variables, see Setting non-default paths using environment variables for more information.

  > **Note**
  >
  > If you have installed Replication Server using the RPM install package from the EDB repository, the value of the {user.home} property is `/var/lib/edb/` and you need to create the `.postgresql` directory in this path to have a valid default client SSL directory path (`/var/lib/edb/.postgresql/`).

  Copy and rename the files:

```
$ cd /var/lib/pgsql/.postgresql/

$ cp /var/lib/pgsql/14/data/server.crt postgresql.crt

$ cp /var/lib/pgsql/14/data/root.crt .
```

```
$ cp /var/lib/pgsql/14/data/server.key postgresql.key

$ openssl pkcs8 -topk8 -inform PEM -in postgresql.key -outform DER -out postgresql.pk8 -v1 PBE-MD5-DES
-nocrypt
```

> **Note**
>
> This completes the SSL configuration for the PostgreSQL publication server and subscription server.

- If you are using EDB Postgres Advanced Server, on the SSL-enabled Postgres database server:

  Make the following client/cert files available on the publication/subscription server using an SSL connection:

  - `xdb.keystore`
  - `xdb_pkcs.p12`

- If you generated the server certificate with `CN=hostname/ip-address`, you cannot use `server.crt/key` as client server/key. Instead, generate the client private key and ceriticate where CN value is specified as db user name. To do so, enter the following commands in the same location as above:

  - `postgresql.key`
  - `postgresql.crt`
  - `postgresql.pk8`
  - `root.crt` (copied from database server data directory)

```
 $ openssl genrsa -out postgresql.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................................................++
+
.....................................................................................................
....+++
e is 65537 (0x10001)
$ openssl req -new -sha256 -key postgresql.key -subj
"/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=enterpr
edb" -out postgresql.csr

Create the client certificate using above postgresql.csr certificate signing request and root.crt and
root.key files will be given as input(root.crt and root.key files will be used we created in
Configuring SSL on a Postgres database server section).
$ openssl x509 -req -in postgresql.csr -CA root.crt -CAkey root.key -CAcreateserial -out
postgresql.crt -days 1024 -sha256
Signature ok
subject=/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=support@enterprisedb.com/CN=
terprisedb
Getting CA Private Key
Enter pass phrase for root.key:
$ openssl pkcs8 -topk8 -inform PEM -in postgresql.key -outform DER -out postgresql.pk8 -v1 PBE-MD5-DES
-nocrypt

$ cp /var/lib/edb/as14/data/root.crt
```

1. Using files `server.crt` and server.key located under the Postgres database server data subdirectory, create copies of these files and move them to the host where the publication server and subscription server are running.

```
 cp server.crt xdb.crt
cp server.key xdb.key
```

For this example, assume file `xdb.crt` is a copy of `server.crt` and `xdb.key` is a copy of `server.key`.

If you generated the server certificate with `CN=hostname/ip-address`, create xdb.keystore/xdb_pkcs.p12 with the `postgresql.crt` and `postgresql.ke` y client files as created with `CN=username` using the following commands :

```
$ cp postgresql.crt xdb.crt
$ cp postgresql.key xdb.key
```

2. Create a copy of `xdb.crt`.

```
$ cp xdb.crt xdb_root.crt
$ ls -l
total 12
-rw-r--r-- 1 user user 1346 Mar 15 09:58 xdb.crt
-rw-r--r-- 1 user user 1704 Mar 15 09:58 xdb.key
-rw-r--r-- 1 user user 1346 Mar 15 10:00 xdb_root.crt
```

If you generated the server certificate with `CN=hostname/ip-address`, use the `root.crt file` from the database data directory in the following command to make `xdb_root.crt` file:

```
$ cp root.crt xdb_root.crt
```

3. Create a distinguished encoding rules (DER) format of file `xdb_root.crt`. The generated DER format of this file is `xdb_root.crt.der`. The DER format of the file is required for the keytool program in the next step.

```
$ openssl x509 -in xdb_root.crt -out xdb_root.crt.der -outform der
$ ls -l
otal 16
-rw-r--r-- 1 user user 1346 Mar 15 09:58 xdb.crt
-rw-r--r-- 1 user user 1704 Mar 15 09:58 xdb.key
-rw-r--r-- 1 user user 1346 Mar 15 10:00 xdb_root.crt
-rw-rw-r-- 1 user user  954 Mar 15 10:05 xdb_root.crt.der
```

4. Use the keytool program to create a keystore file (`xdb.keystore`) using `xdb_root.crt.der` as the input. This process adds the certificate of the Postgres database server to the keystore file.

   You can find the keytool program under the `bin` subdirectory of the Java Runtime Environment installation.

   You are prompted for a new password. Save this password for the next step.

```
$ /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.el7_9.x86_64/jre/bin/keytool -keystore xdb.keystore
-alias xdbstore -import -file xdb_root.crt.der
Enter keystore password:
Re-enter new password:
Owner: CN=enterprisedb, EMAILADDRESS=support@enterprisedb.com, OU=XDB, O=EnterpriseDB, L=Bedford,
ST=Massachusetts, C=US
Issuer: CN=enterprisedb, EMAILADDRESS=support@enterprisedb.com, OU=XDB, O=EnterpriseDB, L=Bedford,
ST=Massachusetts, C=US
Serial number: d7e9966b48e91523
Valid from: Tue Mar 15 08:30:37 GMT-05:00 2016 until: Wed Mar 15 08:30:37 GMT-05:00 2017
Certificate fingerprints:
   MD5:  5D:32:AB:47:A2:44:48:84:0B:CA:EC:9E:C9:28:CE:64
   SHA1: 31:14:C4:0A:E6:93:AA:2C:3E:4B:09:77:AB:94:DB:71:CB:58:99:D9
   SHA256:
2B:EA:59:35:E6:5B:07:07:30:96:D4:80:B0:E1:13:5B:5E:45:97:2E:D0:5C:4F:D8:2F:A6:23:DA:F8:30:D6:17
   Signature algorithm name: SHA1withRSA
```

```
   Version: 1
Trust this certificate? [no]:  yes
Certificate was added to keystore
$ ls -l
total 20
-rw-r--r-- 1 user user 1346 Mar 15 09:58 xdb.crt
-rw-r--r-- 1 user user 1704 Mar 15 09:58 xdb.key
-rw-rw-r-- 1 user user 1019 Mar 15 10:18 xdb.keystore
-rw-r--r-- 1 user user 1346 Mar 15 10:00 xdb_root.crt
-rw-rw-r-- 1 user user  954 Mar 15 10:05 xdb_root.crt.der
```

5. Generate the encrypted form of the new password specified in the preceding step.

   You must specify the encrypted password with the `sslTrustStorePassword` configuration option of the publication server configuration file for publication server SSL connections and the subscription server configuration file for subscription server SSL connections. (See Publication and subscription server configuration options for information on the publication server and subscription server configuration files.)

   Encrypt the password using the Replication Server CLI `encrypt` command. The following example shows this process encrypting the password contained in file `infile`.

```
 $ export PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.el7_9.x86_64/jre/bin:$PATH
$ cd /usr/edb/xdb/bin
$ java -jar edb-repcli.jar -encrypt -input ~/infile -output ~/pwdfile
$ cat ~/pwdfile
LGn6+AagiXqumxVHlOKk3w==
```

6. Create a `PKCS #12` format of the keystore file (`xdb_pkcs.p12`) using files `xdb.crt` and `xdb.key` as input.

   You're prompted for a new password. Save this password for the next step.

```
 $ openssl pkcs12 -export -in xdb.crt -inkey xdb.key -out xdb_pkcs.p12
Enter Export Password:
Verifying - Enter Export Password:
$ ls -l
total 24
-rw-r--r-- 1 user user 1346 Mar 15 09:58 xdb.crt
-rw-r--r-- 1 user user 1704 Mar 15 09:58 xdb.key
-rw-rw-r-- 1 user user 1019 Mar 15 10:18 xdb.keystore
-rw-rw-r-- 1 user user 2557 Mar 15 10:34 xdb_pkcs.p12
-rw-r--r-- 1 user user 1346 Mar 15 10:00 xdb_root.crt
-rw-rw-r-- 1 user user  954 Mar 15 10:05 xdb_root.crt.der
```

7. Generate the encrypted form of the new password specified in the preceding step.

   Specify the encrypted password with the `sslKeyStorePassword` configuration option of the publication server configuration file for publication server SSL connections and the subscription server configuration file for subscription server SSL connections.

   Encrypt the password using the Replication Server CLI `encrypt` command.

```
 $ export PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.322.b06-1.el7_9.x86_64/jre/bin:$PATH
$ cd /usr/edb/xdb/bin
$ java -jar edb-repcli.jar -encrypt -input ~/infile -output ~/pwdfile
$ cat ~/pwdfile
LGn6+AagiXqumxVHlOKk3w==
```

8. Copy files `xdb.keystore` and `xdb_pkcs.p12` to a directory where they will be accessed by the publication server and subscription server.

9. In the publication server and subscription server configuration files, set the location of file `xdb.keystore` with the `sslTrustStore` option and the location of file `xdb_pkcs.p12` with the `sslKeyStore` option.

   The following shows the SSL configuration options set for the files generated in this example.

   ```
   sslTrustStore=/tmp/sslclient/xdb.keystore
   sslTrustStorePassword=LGn6+AagiXqumxVHlOKk3w==
   sslKeyStore=/tmp/sslclient/xdb_pkcs.p12
   sslKeyStorePassword= ygJ9AxoJEX854elcVIJPTw==
   ```

   The encrypted `sslTrustStorePassword` is obtained from Step 5 after being specified for the keytool program in Step 4.

   The encrypted `sslKeyStorePassword` is obtained from Step 7 after being specified for the `openssl pkcs12` program in Step 6.

   Summary of SSL configuration options contains a summary of the publication server and subscription server configuration options for SSL connections.

10. Restart the publication and subscription servers.

Configuring publication/subscription server in case of WAL stream changeset logging

In the case of WAL stream changeset logging, while adding a publication or a subscription database that accepts only SSL connection, Replication Server validates if the database server is configured for logical replication using `libpq` connection.

> **Note**
>
> Ownership depends on the Replication Server service account user. If you have installed Replication Server using the native packages from the EDB repository, the default account user is enterprisedb so ownership needs be given to the enterprisedb user.
>
> ```
> chown enterprisedb postgresql.key
> ```

For the SSL connection, libpq must have the certificates and key as given in the following table along with the client certs and key you set up for trigger mode. The default directory is `${user.home}/.postgresql`.

> **Note**
>
> If you are using EDB Postgres Advanced Server, you need to add and configure the following files, in addition to adding and configuring the `xdb.keystore` and `xdb_pkcs` files, which you added and configured in an earlier step.
>
> If you are using PostgreSQL, you need to add and configure the following files, in addition to adding and configuring `postgresql.pk8`, which you added and configured in an earlier step.

File name	Contents	Description
`~/.postgresql/postgresql.crt`	Client certificate	Requested by the server.
`~/.postgresql/postgresql.key`	Client private key	Proves that the client certificate is sent by the owner. However, doesn't indicate that the certificate owner is trustworthy.
`~/.postgresql/root.crt`	Trusted certificate authorities (CA)	Checks that the server certificate is signed by a trusted certificate authority.

Make sure that the name of the certificates and key is the same as given in the table.

Execute the following commands to change the permission of the certificates in `${user.home}/.postgresql`.

```
chmod 0644 root.crt postgresql.crt
chmod 0600 postgresql.key
chown postgres postgresql.key
```

To set up different source and target database types (for example, source database `=POSTGRESQL` and target database `=enterprisedb`):

1. Generate the certificate for `POSTGRESQL` database and follow the table for placing certificate files in the default directory.

2. Copy these certificates in the EDB Postgres Advanced Server data directory.

   ```
   cd /var/lib/edb/as14/data
   [root@localhost data]# cp /var/lib/pgsql/14/data/root.crt .
   [root@localhost data]# cp /var/lib/pgsql/14/data/server.crt .
   [root@localhost data]# cp /var/lib/pgsql/14/data/server.key .
   ```

3. Use the following commands to change the permissions of the certificates in the EDB Postgres Advanced Server data directory.

   ```
   [root@localhost data]# sudo chown enterprisedb root.crt server.crt server.key
   [root@localhost data]# sudo chgrp enterprisedb root.crt server.crt server.key
   [root@localhost data]# sudo chmod 600 root.crt server.crt server.key
   ```

Using different databases for the source and target

Follow these steps if you are using different databases for the source and target; for example, if you are using PostgreSQL for your source database and EDB Postgres Advanced Server for your target database.

> **Note**
>
> The commands in this section assume `CN=db user name`.

1. Generate the certificate for the PostgreSQL database. See Configuring SSL on a Postgres database server.

2. Configure SSL for Replication Server. See the steps for PostgreSQL in Configuring SSL for the publication server and subscription server.

3. Create the same user in EDB Postgres Advanced Server which is same as the CN value used to generate the certificate for the PostgreSQL database. For example if `CN=postgres` if specified as shown in following command then the `postgres` role should be created in EDB Postgres Advanced Server.

   ```
   openssl req -new -text -nodes -subj
   '/C=US/ST=Massachusetts/L=Bedford/O=EnterpriseDB/OU=XDB/emailAddress=muhammad.imtiaaz@enterprisedb.com/C
   postgres' -keyout server.key -out server.csr
   ```

   Create the user:

   ```
   CREATE ROLE postgres LOGIN SUPERUSER PASSWORD
   'edb';
   ```

   If you specified `map=sslusers` for PostgreSQL and EDB Postgres Advanced Server in `pg_hba.conf`, add the following to `pg_ident.conf` using the same user name for both PostgreSQL and EDB Postgres Advanced Server:

```
cat /var/lib/pgsql/14/data/pg_ident.conf
# ---------------------------------

# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
sslusers        postgres                postgres
```

4. Copy the certificates from the PostgreSQL data directory to the EDB Postgres Advanced Server data directory:

```
cd /var/lib/edb/as14/data
[root@localhost data]# cp /var/lib/pgsql/14/data/root.crt .
[root@localhost data]# cp /var/lib/pgsql/14/data/server.crt .
[root@localhost data]# cp /var/lib/pgsql/14/data/server.key .
```

5. Restart the EDB Postgres Advanced Server service.

```
systemctl restart edb-as-14.service
```

6. Change the permissions of the certificates in the EDB Postgres Advanced Server data directory.

```
[root@localhost data]# sudo chown enterprisedb root.crt server.crt server.key
[root@localhost data]# sudo chgrp enterprisedb root.crt server.crt server.key
[root@localhost data]# sudo chmod 600 root.crt server.crt server.key
```

7. Make the required configuration changes for EDB Postgres Advanced Server see Configuring SSL on a Postgres database server and restart the service:

```
systemctl restart edb-as-14.service
```

Requesting SSL connection to the Replication Server databases

Once SSL connectivity is configured, you must supply a URL option when configuring a single-master or multi-master replication system for those databases to which you intend to use an SSL connection.

The SSL URL option informs Java to use SSL when the publication server or subscription server attempts to connect to a Replication Server database (publication, subscription, or primary node database) on which the SSL URL option is set to `true`.

The configuration steps where these options are specified are as follows:

- For using SSL connections in a single-master replication system, specify the URL options as shown in Adding a publication database for the publication database and in Adding a subscription database for the subscription databases.
- For using SSL connections in a multi-master replication system, you must specify the URL options as shown in Adding the primary definition node for the primary definition node and in Creating more primary nodes for the non-MDN nodes.

Earlier we created self-signed certificates for the database server by specifying the value of the CN field as the database user name (for example, postgres or enterprisedb, and so on). In this case, we use the “verify-ca” value for sslmode parameter to indicate the server certificate is validated against the CA. We do this because the hostname given in the command Add Database or Update Database could not be verified against CN value present certificate, which is the database user name.

For publication, subscription, and primary node databases, in the URL Options field of the Add Database or Update Database dialog box, enter the following:

```
ssl=true&sslmode=verify-ca
```

> **Note**
>
> `sslmode=verify-full` can be used if server certificates are generated by using hostname/IP. When a server certificate is created with the hostname/IP as the CN value there is no need to specify `sslmode=verify-ca` and replication server by default will take it `sslmode=verify-full`.

You can specify the `ssl=true&sslmode=verify-ca` URL option on the Add Database dialog box.

> **Note**
>
> If you no longer want to use an SSL connection to a Replication Server database, you must delete the `ssl=true` text from the **URL Options** field of the Add Database or Update Database dialog box. Changing `true` to `false` doesn't disable the SSL option.

Setting non-default paths using environment variables

You can override the default paths of certificates and keys by setting the non-default paths in a terminal using the environment variables PGSSLKEY, PGSSLCERT, and PGSSLROOTCERT. You then need to export the paths in a terminal before running any Replication Server CLI command or launching the EPRS Replication Console. For example:

```
$ export PGSSLKEY=/home/postgresql.pk8
$ export PGSSLCERT=/home/postgresql.crt
$ export PGSSLROOTCERT=/home/root.crt
```

After setting and exporting the environment variables, from the same terminal, you may either run the Replication Server CLI command or launch the EPRS Replication Console.

Setting non-default paths using SSL connection parameters

Non-default paths of certificates and keys can be overridden using SSL connection parameters sslrootcert, sslcert, and sslkey. You need to specify these parameter values in urlOptions:

```
-urloptions "ssl=true&sslmode=verify-
ca&sslcert=/home/postgresql.crt&sslkey=/home/postgresql.pk8&sslrootcert=/home/root.crt"
```

You set urlOptions using either the:

- Replication Server CLI using the addpubdb/addsubdb command
- EPRS Replication Console while adding the publication and subscription database

Summary of SSL configuration options

The following is a summary of the publication server and subscription server configuration options that apply to SSL connections.

`sslTrustStoreType`

The `sslTrustStoreType` option specifies the truststore format. Set this option to the Java truststore format of the client.

`sslTrustStoreType=truststore_format`

The default value for `truststore_format` is `jks` for the JKS truststore file format.

`sslTrustStore`

The Replication Server uses the default Java truststore for SSL connectivity.

The typical default location of the truststore is in directory `JAVA_HOME/jre/lib/security` or `JAVA_HOME/lib/security` in a file named `cacerts`. (`JAVA_HOME` is the Java installation directory.)

Specify the full directory path to the truststore file with this option. `sslTrustStore=truststore_file`

`sslTrustStorePassword`

Encrypt the password for the Java system truststore using the Replication Server CLI `encrypt` command (see Encrypting passwords), and specify the encrypted password with the `sslTrustStorePassword` option.

`sslTrustStorePassword=encrypted_password`

`sslKeyStoreType`

The `sslKeyStoreType` option specifies the keystore format. Set this option to the Java keystore format of the client.

`sslKeyStoreType=keystore_format`

The default value for `keystore_format` is `pkcs12` for the PKCS #12 keystore file format.

`sslKeyStore`

Specify the full directory path to the keystore file with this option.

`sslKeyStore=keystore_file`

`sslKeyStorePassword`

Encrypt the password for the Java system keystore using the Replication Server CLI `encrypt` command (see Encrypting passwords) and specify the encrypted password with the `sslKeyStorePassword` option.

`sslKeyStorePassword=encrypted_password`

11 Replication Server command line interface

The Replication Server command line interface (CLI) is a command-line-driven alternative to the Replication Server console.

The steps for creating a replication system using the Replication Server CLI are the same as those required when using the Replication Server console. The logical components of the replication system must be created in the same order, with the same sets of attributes as when creating the replication system with the Replication Server console.

Before building a replication system using the Replication Server CLI, be sure you understand the concepts and steps presented in theoverview and Single-master replication operation (for single-master replication) or Multi-Master Replication Operation (for multi-master replication). You can use both the Replication Server console and the Replication Server CLI to build and manage a replication system.

In Replication Server CLI commands, the syntax and examples given for each Replication Server CLI command run individually. Where applicable, the

discussion of a command contains a reference back to its Replication Server console counterpart, where you can find a detailed description of the affected component and its attributes.

11.1 Prerequisite steps before using the Replication Server CLI

Perform these installation and setup steps before using the Replication Server CLI.

The Replication Server CLI is included if you choose the Replication Server console component when installing Replication Server. The Replication Server CLI is a Java application found in the directory `XDB_HOME/bin`.

1. Follow the installation steps in Installation and uninstallation to install Replication Server.

2. Follow the prerequisite steps in Prerequisite steps for single-master replication systems or Prerequisite steps for multi-master replication systems.

3. Set the Java Runtime Environment:

   The Java Runtime Environment (JRE) must be present on the host from which you intend to run the Replication Server CLI. The Java runtime `bin` directory must be included in the path of the operating system user name that will run Replication Server CLI.

   The Replication Server startup configuration file, `xdbReplicationServer-xx.config`, contains the path of the JRE runtime program that was detected during the installation of Replication Server. The following is an example of the Replication Server startup configuration file. (See Installation details for the location of this file.)

   ```
   #!/bin/sh

   JAVA_EXECUTABLE_PATH="/usr/bin/java"
   JAVA_MINIMUM_VERSION=1.7
   JAVA_BITNESS_REQUIRED=64
   JAVA_HEAP_SIZE="-Xms2048m -
   Xmx4096m"
   PUBPORT=9051
   SUBPORT=9052
   ```

   For example, using the JRE path shown in the configuration file, enter the following on the command line or add it to your profile:

   ```
   export PATH=/usr/bin:$PATH
   ```

   On Windows systems, open the Properties dialog box of My Computer. Select **Advanced System Settings**, and then select **Environment Variables**. Edit the `Path` system environment variable to include the Java Runtime Environment `bin` directory. Alternatively, you can set the path for just the current session when you open the Command Prompt as in the following example:

   ```
   SET Path=C:\Program Files\Java\jre1.8.0_45\bin;%Path%
   ```

11.2 General usage of the Replication Server CLI

Running Replication Server CLI

You can run the Replication Server CLI from any host on which you can run the Replication Server console. Run the Replication Server CLI by executing the Java runtime program and specifying the following arguments to the Java program:

- The path to the Replication Server CLI jar file `edb-repcli.jar`
- A Replication Server CLI command
- One or more publication names or subscription names if the command acts on a publication or subscription
- Parameters and their values that apply to the command

The Java jar file `edb-repcli.jar` is located in directory `XDB_HOME/bin`.

Each Replication Server CLI command has the following general syntax:

```
-command [ { pubname | subname } ...]
        ``[ -parameter [ value ] ...] ...
```

`command` is the name of a Replication Server CLI command. The command name must be prefixed by a hyphen character (-). If the command acts on a publication, specify the name of the publication, represented by `pubname`. If the command acts on a subscription, spcify the subscription name, represented by `subname`. Some commands might allow you to specify more than one publication name or more than one subscription name.

One or more parameters might follow. Each parameter name must have a hyphen prefix. You might need to specify one or more values for a parameter.

If a command takes more than one parameter, the order in which you specify the parameters makes no difference. Each parameter must be followed only by the values that pertain to it.

Command names and parameter names are all case sensitive and must be given as shown in Getting help.

The general, complete execution syntax that you enter at the command line prompt has the following format:

```
java -jar XDB_HOME/bin/edb-repcli.jar
  -command [ { pubname | subname } ...]
           [ -parameter [ value ] ...] ...
```

Enter the syntax as one logical line on the command line. It is broken up into multiple lines in the syntax diagram for the purpose of clarity.

> **Note**
>
> You can continue a command onto the next physical line if you enter the operating system's continuation character (for example, the backslash character (`\`) in Linux or the caret character (`^`) in Windows) before pressing **Enter**.

Getting help

If you execute the Replication Server CLI with the `help` command, Replication Server CLI lists a syntax summary of all commands. See Getting help.

Supplying the publication or subscription server login information

Using the parameter `repsvrfile` is the Replication Server CLI equivalent for the process of registering the publication server or the subscription server in the Replication Server console.

Registering a publication server discusses how the first step in building a replication system is to register the publication server. In the Replication Server console, the registered publication server appears as a node in the replication tree. The Publication Server node provides a context to which you can add other logical components of the replication system.

When using the Replication Server CLI, there's no replication tree image available with which to relate the other logical components of the replication system. Instead, whenever you execute a Replication Server CLI command that requires the context of a publication server or subscription server, you must

specify the publication server's login information or the subscription server's login information by means of the `repsvrfile` parameter.

The `repsvrfile` parameter takes as its value the path to a text file that contains the login information of either the publication server instance or the subscription server instance that you want to use. The general Replication Server CLI command syntax that includes the `repsvrfile` parameter is shown in the following diagram:

```
-command [ { pubname | subname } ...]
         [ -parameter [ value ] ...] ...
         [ -repsvrfile repsvrfile ]
         [ -parameter [ value ] ...] ...
```

The Replication Server CLI command to execute is represented by `command`. If required, publication names represented by `pubname` or subscription names represented by `subname` are specified next. The path to the text file containing either the publication server or subscription server login information is represented by `repsvrfile`. The parameters and their values that are used with the command are denoted by `parameter` and `value`.

The order on the command line in which you give `-repsvrfile repsvrfile` and `-parameter` and its values doesn't matter. For example, you can give `-repsvrfile repsvrfile` as the first parameter on the command line, the last parameter on the command line, or somewhere in between other parameters.

The following is an example of `repsvrfile` for a publication server:

```
host=localhost
port=9051
user=admin
# Password is in encrypted
form.
password=ygJ9AxoJEX854elcVIJPTw==
```

The following is an example of `repsvrfile` for a subscription server:

```
host=localhost
port=9052
user=admin
# Password is in encrypted
form.
password=ygJ9AxoJEX854elcVIJPTw==
```

You can locate these files in any directory as long as you can read them.

In your file, be sure to replace the values of the following fields with the values for your publication server or subscription server:

- `Host`
- `Port`
- `User`
- `Password`

This is the same information you need to register the publication server or subscription server using the Replication Server console. See Registering a publication server and Registering a subscription server.

The following example shows how to use the `repsvrfile` parameter with the `printpublist` command.

```
$ java -jar edb-repcli.jar -printpublist -repsvrfile ~/pubsvrfile.prop
Printing publications ...
analysts_managers
dept_emp
emp_pub
```

Using encrypted passwords in text files

When you use the Replication Server CLI, text files store certain information, which can include user names and passwords. An example is the files containing publication server and subscription server login information used with the `repsvrfile` parameter.

In the file specified with parameter `repsvrfile`, the password field must be set to a password in encrypted form. Using an encrypted password prevents unauthorized personnel from accessing the publication server or subscription server using the values of `user` and `password` if the file is compromised. (You can't use the encrypted password to access the publication server or subscription server from the dialog box in the Replication Server console.)

See Encrypting passwords to learn how to generate an encrypted password using the `encrypt` command.

Running Replication Server CLI using a parameter file

The `paramfile` command allows you to run a Replication Server CLI command and its parameters that were coded into a text file. This technique is useful if you want to save the command and its parameters for repeated use.

The syntax for executing `paramfile` is shown by the following:

```
java -jar XDB_HOME/bin/edb-repcli.jar
  -paramfile cmdparamfile
```

The syntax of the Replication Server CLI command and its parameters coded into text file `cmdparamfile` is the same as if given at the command line prompt, as shown by the following:

```
-command [ { pubname | subname } ...]
         [ -parameter [ value ] ...] ...
         [ -repsvrfile repsvrfile ]
         [ -parameter [ value ] ...] ...
```

Using the `paramfile` command has the following restrictions:

- You can code only one Replication Server CLI command into the parameter file `cmdparamfile`.
- The parameters to use with the Replication Server CLI command must all be included in `cmdparamfile`. You can't code some of the parameters into `cmdparamfile` and give other parameters on the command line.

The following example creates an EDB Postgres Advanced Server publication database definition using a parameter file named `addpubdb_advsvr`.

The following is the content of parameter file `addpubdb_advsvr`:

```
-addpubdb
  -repsvrfile /home/user/pubsvrfile.prop
  -dbtype enterprisedb
  -dbhost 192.168.2.4
  -dbport 5444
  -dbuser pubuser
  -dbpassword ygJ9AxoJEX854elcVIJPTw==
  -database edb
  -repgrouptype s
```

For Windows only: You can specify the `-repsvrfile` directory path with either the forward slash or backslash character. Enclose the entire directory path in double quotation marks if a directory name contains space characters:

```
-addpubdb
  -repsvrfile "C:\Users\User Name\repcli\pubsvrfile.prop"
  -dbtype enterprisedb
  -dbhost 192.168.2.23
  -dbport 5444
  -dbuser pubuser
  -dbpassword ygJ9AxoJEX854elcVIJPTw==
  -database edb
  -repgrouptype s
```

> **Note**
>
> Unlike entering the Replication Server CLI command and its parameters directly at the command line prompt, when coded into a text file, no continuation characters are needed to continue onto the following lines.

The following shows executing the `paramfile` command:

```
 $ java -jar edb-repcli.jar -paramfile ~/addpubdb_advsvr
Adding Publication Database...
Publication database added successfully. Publication Database id:1
```

Testing the command exit status

After executing a Replication Server CLI command, you can test the exit status to determine if the command was successful.

An exit status of 0 indicates success. A non-zero exit status indicates a failure has occurred.

For Linux only: The environment variable $? contains the exit status. The following example shows the 0 exit status upon the successful execution of the `addpubdb` command contained in the `addpubdb_advsvr` parameter file. See Running Replication Server CLI using a parameter file.

```
 $ java -jar edb-repcli.jar -paramfile ~/addpubdb_advsvr
Adding publication database...
Publication database added successfully. Publication database id:1
$ echo $?
0
```

The following example shows a non-zero exit status when the command failed with an error.

Exit status	Description
0	Success
201	Invalid command
202	I/O error
203	Decryption failed
204	Authentication failed
205	Publication service failure
206	Remote exception
207	Subscription service failure

```
$ java -jar edb-repcli.jar -paramfile ~/addpubdb_advsvr
```

```
Adding publication database...
Error:The connection attempt failed.
$ echo $?
200
```

For Windows only: The environment variable `%ERRORLEVEL%` contains the exit status.

The following shows the exit status upon successful command execution on a Windows system.

```
 C:\Users>java -jar C:\\"Program Files"\edb\EnterpriseDB-xDBReplicationServer\bin\edb-repcli.jar -paramfile
addpubdb_advsvr
Adding publication database...
Publication database added successfully. Publication database id:1

C:\Users>ECHO %ERRORLEVEL%
0
```

The following shows the exit status upon unsuccessful command execution on a Windows system.

```
 C:\Users>java -jar C:\\"Program Files"\edb\EnterpriseDB-xDBReplicationServer\bin\edb-
repcli.jar -paramfile addpubdb_advsvr
Adding publication database...
Error:FATAL: password authentication failed for user "myuser"

C:\Users>ECHO %ERRORLEVEL%
200
```

11.3 Replication Server CLI commands

The CLI commands are listed here in the order in which they are typically used, following the order of Replication Server console operations.

The examples assume that the Replication Server CLI commands are executed after you make `XDB_HOME/bin` your current working directory. This configuration means you don't have to specify the full path of `XDB_HOME/bin` for each execution of the `edb-repcli.jar` file. For example, assuming Replication Server is installed in the default installation directory, you issued the following command in Linux:

```
cd /usr/edb/xdb/bin
```

In Windows, the equivalent is the following:

```
cd C:\Program Files\edb\EnterpriseDB-xDBReplicationServer\bin
```

Whenever the `repsvrfile` parameter appears in the examples, the file `~/pubsvrfile` contains the publication server login information and is located in the user's home directory while `~/subsvrfile` contains the subscription server login information. For Windows, the equivalent usage is `%HOMEPATH%\pubsvrfile and %HOMEPATH%\subsvrfile.`

The examples are shown on a Linux system. They use the Linux continuation character, which is a backslash (`\`) to show how a Replication Server CLI command can continue onto the next line if you don't want to wrap the text in your terminal window. For Windows, use the Windows continuation character, which is a caret (`^`).

11.3.1 Getting help (help)

The `help` command provides a syntax summary of all Replication Server CLI commands.

Synopsis

`-help`

Examples

```
 $ java -jar edb-repcli.jar -help
Usage: java -jar edb-repcli.jar [OPTIONS]

Where OPTIONS include:
-help   Prints out Replication CLI command-line usage
-version        Prints out Replication CLI version
-encrypt -input <file> -output <file>   Encrypts input file to output file
-repversion -repsvrfile <file>  Prints Replication Server version
                      .
```

11.3.2 Printing the version number (version)

The `version` command displays the Replication Server CLI version number.

Synopsis:

`-version`

Examples

```
 $ java -jar edb-repcli.jar -version
Version: 7.0.0                      .
```

11.3.3 Printing the Replication Server version number (repversion)

The `repversion` command displays the Replication Server version number.

Synopsis

```
-repversion -repsvrfile pubsvrfile
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

Examples

```
$ java -jar edb-repcli.jar -repversion -repsvrfile ~/pubsvrfile.prop
7.0.0

                      .
```

11.3.4 Encrypting passwords (encrypt)

The `encrypt` command encrypts the text supplied in an input file and writes the encrypted result to a specified output file. Use the `encrypt` command to generate an encrypted password to copy into a text file. This text file is referenced by a Replication Server CLI command that requires a user name and the user's password.

Synopsis

```
-encrypt -input infile -output pwdfile
```

The text in `infile` is processed using the MD5 encryption algorithm, and the encrypted text is written to file `pwdfile`. Make sure that `infile` contains only the text that you want to encrypt and that there are no extra characters or empty lines before or after the text that you want to encrypt.

Parameters

`infile`

The file containing the text to be encrypted.

`pwdfile`

The file containing the encrypted form of the text from `infile`.

Examples

The file `infile` contains the word `password`.

```
password
```

Executing the `encrypt` command produces a file named `pwdfile`.

```
$ java -jar edb-repcli.jar -encrypt -input ~/infile -output ~/pwdfile
```

The content of file pwdfile contains the encrypted form of `password`.

```
ygJ9AxoJEX854elcVIJPTw==
```

11.3.5 Printing the length of time the server has been running (uptime)

The `uptime` command displays the time interval since the publication server was up and running.

Synopsis

```
-uptime -repsvrfile pubsvrfile
```

Parameters

```
pubsvrfile
```

The file containing the publication server login information.

Examples

```
$ java -jar edb-repcli.jar -uptime -repsvrfile ~/pubsvrfile.prop
0 days 0 hours 4 minutes
```

11.3.6 Adding a publication database (addpubdb)

The `addpubdb` command adds a publication database definition.

Synopsis

```
-addpubdb
```

```
  -repsvrfile pubsvrfile
  -dbtype { oracle | enterprisedb | postgresql | sqlserver }
  -dbhost host
  -dbport port
  -dbuser user
{ -dbpassword encrypted_pwd | -dbpassfile pwdfile }
[ -oraconnectiontype { sid | servicename } ]
  -database dbname
[ -urloptions jdbc_url_parameters ]
[ -filterrule filterid_1[,filterid_2 ] ...]
[ -repgrouptype { m | s } ]
[ -replicatepubschema { true | false } ]
[ -initialsnapshot
  [ -verboseSnapshotOutput { true | false } ] ]
[ -nodepriority priority_level ]
[ -changesetlogmode { T | W } ]
```

The `addpubdb` command creates a new publication database definition. The `addpubdb` command displays a unique publication database ID that is assigned to the newly created publication database definition. The publication database ID is used to identify the publication database definition on which to operate when running other Replication Server CLI commands.

See Adding a publication database for details on the database connection information you must supply when adding a publication database definition for a single-master replication system. See Adding the primary definition node and Creating more primary nodes for a multi-master replication system.

Parameters

`pubsvrfile`

The file containing the publication server login information.

`-dbtype`

Specify the values as shown in the table.

Value	Database
`oracle`	Oracle
`enterprisedb`	EDB Postgres Advanced Server database in Oracle-compatible configuration mode
`postgresql`	PostgreSQL database or an EDB Postgres Advanced Server database in PostgreSQL-compatible configuration mode
`sqlserver`	Microsoft SQL Server

`host`

The IP address of the host where the publication database server is running.

`port`

The port number on which the database server is listening for connections.

`user`

The publication database user name.

`encrypted_pwd`

The encrypted password of the publication database user. See Encrypting passwords to learn how to use the `encrypt` command to generate an encrypted password.

`pwdfile`

The file containing the encrypted password of the publication database user.

`-oraconnectiontype`

Specify `sid` to use the Oracle system ID (SID) to identify the publication database in the database parameter. Specify `servicename` to use the Oracle service name to identify the publication database in the database parameter.

> **Note**
>
> For Oracle 12c, use the service name.

`dbname`

The Postgres or SQL Server database name, the Oracle SID, or the Oracle service name of the publication database.

`jdbc_url_parameters`

Extended usage of JDBC URL parameters such as for support of SSL connectivity. See Preparing using secure sockets layer (SSL) connections for information on SSL connectivity to the publication database.

`filterid_n`

For MMR only: Applies to non-MDN nodes. Comma-separated list of filter IDs identifying the filter rules from the set of available table filters to enable on the corresponding tables in the new primary node. Use the `printpubfilterslist` command to get the filter IDs for the available filter rules in the publication (see Printing a list of filters in a publication). Do not use any white space between the comma and filter IDs.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system. The default is `s`.

`-replicatepubschema`

For MMR only: Applies to non-MDN nodes. Set this option to `true` if you want the publication table definitions replicated from the primary definition node when creating a new primary node. Set this option to `false` if you already created the table definitions in the new primary node. The default is `true`. Don't specify this parameter when creating the primary definition node.

Unless you intend to use the offline snapshot technique (see Loading tables from an external data source (offline snapshot), we suggest that you specify this option. You must perform an initial snapshot replication from the primary definition node to every other primary node before performing synchronization replications on demand (see Performing a synchronization) or by a schedule (see Configuring a multi-master schedule). If a newly added primary node didn't undergo an initial snapshot, any later synchronization replication might not apply the transactions to that primary node. You can also take the initial snapshot by performing an on-demand snapshot (see Take a multi-master snapshot).

`-initialsnapshot`

For MMR only: Applies to non-MDN nodes. Specify this option if you want an initial snapshot replication performed when creating the primary node. Omit this option if you don't want an initial snapshot replication performed when creating the primary node.

> **Note**

> Unless you intend to use the offline snapshot technique (see Loading tables from an external data source (offline snapshot)), we suggest that you specify this option. You must perform an initial snapshot replication from the primary definition node to every other primary node before performing synchronization replications on demand (see Performing a synchronization) or by a schedule (see Configuring a multi-master schedule). If a newly added primary node didn't undergo an initial snapshot, any later synchronization replication might not apply the transactions to that primary node. You can also take the initial snapshot by performing an on-demand snapshot (see Take a multi-master snapshot) or by a schedule (see Configuring a multi-master schedule).

`-verboseSnapshotOutput`

Set this option to `true` if you want to display the output from the snapshot. Set this option to `false` if you don't want to display the snapshot output. The default is `true`.

Use this option after the `-initialsnapshot` option.

`priority_level`

For MMR only: Integer value from 1 through 10 assigning the priority level to a primary node. 1 has the highest priority and 10 has the lowest priority.

`-changesetlogmode`

Specify `T` to use the trigger-based method of synchronization replication for this publication database. Specify `W` to use the log-based (WAL) method of synchronization replication for this publication database. The default is `T`.

Examples

This example adds a publication database definition for an Oracle database. The encrypted password is given on the command line with the `dbpassword` parameter. A publication database ID of 1 is assigned to the database by the publication service.

```
$ java -jar edb-repcli.jar -addpubdb -repsvrfile ~/pubsvrfile.prop \
>   -dbtype oracle -dbhost 192.168.2.6 -dbport 1521 \
>   -dbuser pubuser -dbpassword ygJ9AxoJEX854elcVIJPTw== \
>   -oraconnectiontype sid \
>   -database xe \
>   -repgrouptype s
Adding publication database...
Publication database added successfully. Publication database id:1
```

This example adds a publication database definition for an EDB Postgres Advanced Server database. The encrypted password is read from a file named `pwdfile` with the `dbpassfile` parameter. A publication database ID of 2 is assigned to the database by the publication service.

```
$ java -jar edb-repcli.jar -addpubdb -repsvrfile ~/pubsvrfile.prop \
>   -dbtype enterprisedb -dbhost 192.168.2.7 -dbport 5444 \
>   -dbuser pubuser -dbpassfile ~/pwdfile \
>   -database edb \
>   -repgrouptype s
Adding publication database...
Publication database added successfully. Publication database id:2
```

This example adds a publication database definition for a primary definition node in a multi-master replication system.

```
$ java -jar edb-repcli.jar -addpubdb -repsvrfile ~/pubsvrfile.prop \
>   -dbtype enterprisedb -dbhost 192.168.2.6 -dbport 5444 \
>   -dbuser pubuser -dbpassword ygJ9AxoJEX854elcVIJPTw== \
>   -database edb \
```

```
>   -repgrouptype m \
>   -nodepriority 1
Adding publication database...
Publication database added successfully. Publication database id:3
```

This example adds a publication database definition for a primary node (other than the primary definition node) in a multi-master replication system. An initial snapshot isn't invoked (the `initialsnapshot` parameter is omitted). Filter rules with filter IDs 8 and 16 are applied to this primary node. A node priority level of 3 is assigned to the primary node.

> **Note**
>
> A publication must be created in the primary definition node before creating additional primary nodes. SeeCreating a Publication for the command to create a publication.

```
$ java -jar edb-repcli.jar -addpubdb -repsvrfile ~/pubsvrfile.prop \
>   -dbtype enterprisedb -dbhost 192.168.2.7 -dbport 5444 \
>   -dbuser MMRuser -dbpassword ygJ9AxoJEX854elcVIJPTw== \
>   -database MMRnode \
>   -filterrule 8,16 \
>   -repgrouptype m \
>   -nodepriority 3
Adding publication database...
Replicating publication schema...
Publication database added successfully. Publication database id:24
```

11.3.7 Printing publication database IDs (printpubdbids)

The `printpubdbids` command displays the publication database IDs of the publication database definitions.

Synopsis

```
-printpubdbids -repsvrfile pubsvrfile
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

Examples

This example lists the publication database IDs of the publication database definitions.

```
$ java -jar edb-repcli.jar -printpubdbids -repsvrfile ~/pubsvrfile.prop
Printing publication database ids...
2
```

```
1
24
3
```

11.3.8 Printing publication database details (printpubdbidsdetails)

The `printpubdbidsdetails` command displays the connection information for each publication database definition.

Synopsis

`-printpubdbidsdetails -repsvrfile pubsvrfile`

The output has the following components:

`dbid:host:port:dbname:user`

> **Note**
>
> The database user's password isn't displayed.

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID assigned to the publication database definition.

`host`

The IP address of the host where the publication database server is running.

`port`

The port number on which the database server is listening for connections.

`dbname`

The Postgres or SQL Server database name, the Oracle SID, or the Oracle service name of the publication database.

`user`

The publication database user name.

Examples

Four publication database definitions are subordinate to the publication server identified by the content of file `pubsvrfile.prop`.

```
 $ java -jar edb-repcli.jar -printpubdbidsdetails \
>   -repsvrfile ~/pubsvrfile.prop
Printing publication database ids with details...
id:host:port:database|sid:user
2:192.168.2.7:5444:edb:pubuser
1:192.168.2.6:1521:xe:pubuser
24:192.168.2.7:5444:MMRnode:MMRuser
3:192.168.2.6:5444:edb:pubuser
```

11.3.9 Printing the controller database ID (printcontrollerdbid)

The `printcontrollerdbid` command displays the publication database ID of the controller database.

Synopsis

```
-printcontrollerdbid -repsvrfile pubsvrfile
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

Examples

This example displays the publication database ID of the controller database.

```
 $ java -jar edb-repcli.jar -printcontrollerdbid -repsvrfile ~/pubsvrfile.prop
Printing Controller database id...
1
```

11.3.10 Printing the primary definition node database ID (printpdndbid)

For MMR only: The `printmdndbid` command prints the publication database ID of the primary definition node.

Synopsis

```
-printmdnbid -repsvrfile pubsvrfile
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

Examples

This example displays the publication database ID of the primary definition node.

```
$ java -jar edb-repcli.jar -printmdndbid -repsvrfile ~/pubsvrfile.prop
Printing MDN Publication database id...
3
```

11.3.11 Updating a publication database (updatepubdb)

The `updatepubdb` command lets you change the connection information for an existing publication database definition identified by its publication database ID.

Synopsis

```
-updatepubdb
  -repsvrfile pubsvrfile
  -pubdbid dbid
  -dbhost host
  -dbport port
  -dbuser user
{ -dbpassword encrypted_pwd | -dbpassfile pwdfile }
[ -oraconnectiontype { sid | servicename } ]
  -database dbname
[ -urloptions jdbc_url_parameters ]
[ -nodepriority priority_level ]
```

Identify the publication database definition to update with the `pubdbid` parameter.

See Adding a publication database for details on the database connection information to supply for a publication database definition for a single-master replication system. See Adding the primary definition node and Creating more primary nodes for a multi-master replication system.

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the publication database definition to update.

`host`

The IP address of the host where the publication database server is running.

`port`

The port number on which the database server is listening for connections.

`user`

The publication database user name.

`encrypted_pwd`

The encrypted password of the database user. See Encrypting passwords to learn how to use the `encrypt` command to generate an encrypted password.

`pwdfile`

The file containing the password of the database user in encrypted form.

`-oraconnectiontype`

Specify `sid` to use the Oracle system ID (SID) to identify the publication database in the database parameter. Specify `servicename` to use the Oracle service name to identify the publication database in the database parameter.

> **Note**
>
> For Oracle 12c, use the service name.

`dbname`

The Postgres or SQL Server database name, the Oracle SID, or the Oracle service name of the publication database.

`jdbc_url_parameters`

Extended usage of JDBC URL parameters such as for support of SSL connectivity. See Preparing using secure sockets layer (SSL) connections for information on SSL connectivity to the publication database. Specifying the `urloptions` parameter completely replaces any existing JDBC URL parameters that were previously specified with this database. Omitting the `urloptions` parameter deletes any existing JDBC URL parameters that were previously specified with this database.

`priority_level`

For MMR only: Integer value from 1 through 10 assigning the priority level to a primary node. 1 has the highest priority, and 10 has the lowest priority.

Examples

This example updates an existing publication database definition with publication database ID 1.

```
$ java -jar edb-repcli.jar -updatepubdb -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 1 \
>   -dbhost 192.168.2.6 -dbport 1521 \
>   -dbuser pubuser -dbpassfile ~/pwdfile \
>   -oraconnectiontype sid \
>   -database xe
Updating publication database ...
Publication database with ID 1 is updated successfully.
```

11.3.12 Removing a publication database (removepubdb)

The `removepubdb` command removes a publication database definition.

Synopsis

```
-removepubdb
  -repsvrfile pubsvrfile
  -pubdbid dbid
```

The `pubdbid` parameter identifies the publication database definition to remove.

See Removing a publication database
for more information.

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the publication database definition to remove.

Examples

This example removes the publication database definition identified by publication database ID 1.

```
$ java -jar edb-repcli.jar -removepubdb -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 1
Removing Publication Database...
Publication Database removed.
```

11.3.13 Get tables for a new publication (gettablesfornewpub)

The `gettablesfornewpub` command lists the tables and views available for including in a new publication from a given publication database definition.

Synopsis

```
-gettablesfornewpub -repsvrfile repsvrfile -pubdbid dbid
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the publication database definition whose available tables and views to list.

Examples

For the publication database definition identified by publication database ID 1, the tables available for inclusion in a publication are `EDB.DEPT`, `EDB.EMP`, and `EDB.JOBHIST`. The view available for inclusion in a publication is `EDB.SALESEMP`.

```
 $ java -jar edb-repcli.jar -gettablesfornewpub \
>   -repsvrfile ~/pubsvrfile.prop -pubdbid 1
Fetching tables/views list ...
[[EDB.DEPT, TABLE], [EDB.EMP, TABLE], [EDB.JOBHIST, TABLE], [EDB.SALESEMP, VIEW]]
```

11.3.14 Creating a publication (createpub)

The `createpub` command creates a publication.

Synopsis

```
 -createpub pubname
-repsvrfile pubsvrfile
-pubdbid dbid
-reptype { s | t }
[ { -tables schema_t1.table_1 [schema_t2.table_2] ... | -alltables
    [ schema1 [ schema2 ] ... ] } ]
```

```
[ { -views schema_v1.view_1 [ schema_v2.view_2 ] ... | -allviews
    [ schema_v1 [ schema_v2 ] ... ] } ]
[ -tablesfilterclause
    "schema_t1.table_1:filtername_t1:filterclause_t1"
  [ "schema_t2.table_2:filtername_t2:filterclause_t2" ] ...]
[ -viewsfilterclause
    "schema_v1.view_1:filtername_v1:filterclause_v1"
  [ "schema_v2.view_2:filtername_v2:filterclause_v2" ] ...]
[-defaultconflictresolution {E|L|N|M}]
[-defaultstandbyconflictresolution {E|L|N|M}]
[ -conflictresolution
    schema_t1.table_1:{ E | L | N | M | C:customhandler_t1 }
  [ schema_t2.table_2:{ E | L } N | M | C:customhandler_t2 } ] ...]
[ -standbyconflictresolution
    schema_t1.table_1:{ E | L | N | M | C:customhandler_t1 }
  [ schema_t2.table_2:{ E | L } N | M | C:customhandler_t2 } ] ...]
```

The `createpub` command adds a publication subordinate to the publication database definition with the publication database ID given by parameter `pubdbid`. If the publication is designated as snapshot-only by setting parameter `reptype` to `s`, then any views listed after the views parameter are ignored.

See Adding a publication for more information on creating a publication for a single-master replication system. See Adding a publication for a multi-master replication system.

> **Note**
>
> The schema names, table names, and view names that you supply as values for the tables and views parameters are case sensitive. Unless quoted identifiers were used to build the database objects, you must enter Oracle names using uppercase letters (for example, `EDB.DEPT`) and EDB Postgres Advanced Server names in lowercase letters (for example `edb.dept`). See Quoted identifiers and default case translation for more information.

Parameters

`pubname`

The publication name to give to the new publication.

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the publication database definition the new publication is added as a subordinate. This `createpub` command is applied to a single-master replication (SMR) or multi-master replication (MMR) system according to the Replication Group Type of this database.

`-reptype`

Specify `s` for a snapshot-only publication. Specify `t` if the publication is transactional (to allow synchronization replications).

`-tables` or `-alltables`

Use these options to include tables into publication. If `reptype` is `t` (transactional), then you must include `-tables` or `-alltables`. Otherwise, these options are optional.

The `-tables` option includes tables from the tables list to the publication. The `-alltables` option allows you to include all tables of the database or only tables from the listed schemas (optional) to the publication.

`-views or -allviews`

For SMR and -reptype s (snapshot) only: Use these options to include views into publication.

The `-views` option includes views from the views list to the publication. The `-allviews` option allows you to include all views of the database or only views from the listed schemas (optional) to the publication.

`schema_tn`

The name of the schema containing the nth table of the tables parameter list or the name of the nth schema of the `alltables` parameter list . This value is case sensitive.

`table_n`

The table name of the nth table in the tables parameter list. This value is case sensitive.

`schema_vn`

For SMR only: The name of the schema containing the nth view of the views parameter list or the name of the nth schema of the `allviews` parameter list . This value is case sensitive.

`view_n`

For SMR only: View name of the nth view in the views parameter list. This value is case sensitive.

`filtername_tn`

The filter name to assign to the filter rule on the table.

`filterclause_tn`

The filter clause to apply to the table in the tables parameter list for the table indicated by `schema_t1.table_tn` .

`filtername_vn`

The filter name to assign to the filter rule on the view.

`filterclause_vn`

For SMR only: The filter clause to apply to the view in the views parameter list for the table indicated by `schema_v1.table_vn` .

`-defaultconflictresolution`

For MMR only: Default conflict resolution (`E` , `L` , `N` , or `M`), if the `-conflictresolution` option is omitted. The default is `E` .

`-defaultstandbyconflictresolution`

For MMR only: Default standby conflict resolution (`E` , `L` , `N` , or `M`), if the `-standbyconflictresolution` option is omitted. The default is `M` .

`-conflictresolution`

For MMR only: For the `conflictresolution` option, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

The specified conflict resolution applies to the table `schema_tn.table_n` from the tables parameter list. The default is `E`.

`-standbyconflictresolution`

For MMR only: For the `standbyconflictresolution option`, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

The specified conflict resolution applies to the table `schema_tn.table_n` from the tables parameter list. The default is `M`.

`customhandler_tn`

For MMR only: For the `conflictresolution` option or the `standbyconflictresolution` option, specify `customhandler_tn` as the function name. Include an optional schema prefix (that is, formatted as `schema.function_name`) as given in the `CREATE FUNCTION` command for the custom conflict handling function created for the table `schema_tn.table_n` from the tables parameter list. You must add the custom conflict handling function to the primary definition node. See Adding a custom conflict handling function for an example of adding the custom conflict handling function using PSQL. You must specify the custom handler name option only if you set the `conflictresolution` or `standbyconflictresolution` option for custom conflict handling by using the `C` value.

Examples

This example creates a publication named `dept_emp` that contains the EDB.DEPT and EDB.EMP tables of an Oracle database. The replication method is synchronization.

```
$ java -jar edb-repcli.jar -createpub dept_emp \
>   -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 1 \
>   -reptype t \
>   -tables EDB.DEPT EDB.EMP
Creating publication...
Tables:[[EDB.DEPT, TABLE], [EDB.EMP, TABLE]]
Filter clause:[]
Publication created.
```

This example creates a publication named `salesemp` that contains the `EDB.SALESEMP` view of an Oracle database. The replication method is snapshot-only.

```
$ java -jar edb-repcli.jar -createpub salesemp \
>   -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 1 \
>   -reptype s \
>   -views EDB.SALESEMP
```

```
Creating publication...
Tables:[[EDB.SALESEMP, VIEW]]
Filter clause:[]
Publication created.
```

This example creates a publication named `analysts_managers` that contains the edb.dept table and employees from the edb.emp table who are analysts or managers. The tables are in an EDB Postgres Advanced Server database. The replication method is snapshot-only.

```
$ java -jar edb-repcli.jar -createpub analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 2 \
>   -reptype s \
>   -tables edb.dept edb.emp \
>   -tablesfilterclause "2:jobgrade_11:job IN ('ANALYST', 'MANAGER')"
Creating publication...
Tables:[[edb.dept, TABLE], [edb.emp, TABLE]]
Filter clause:[FilterName:jobgrade_11   FilterClause:job IN ('ANALYST', 'MANAGER')      ]
Publication created.
```

This example creates a publication for a multi-master replication system. One table filter is defined on table edb.dept, and three table filters are defined on the table edb.emp. Table edb.dept is assigned node priority conflict resolution and latest timestamp as the standby conflict resolution strategy. Table edb.emp is assigned earliest timestamp conflict resolution and manual resolution (the default) as its standby strategy.

```
$ java -jar edb-repcli.jar -createpub emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 3 \
>   -reptype t \
>   -tables edb.dept edb.emp \
>   -tablesfilterclause "1:dept_10_20_30:deptno in (10, 20, 30)" \
>     "2:dept_10:deptno = 10" \
>     "2:dept_20:deptno = 20" \
>     "2:dept_30:deptno = 30" \
>   -conflictresolution 1:N 2:E \
>   -standbyconflictresolution 1:L 2:M \

Creating publication...
Tables:[[edb.dept, TABLE], [edb.emp, TABLE]]
Filter clause:[FilterName:dept_10_20_30 FilterClause:deptno in (10, 20, 30)     , FilterName:dept_10
FilterClause:deptno = 10 , FilterName:dept_20    FilterClause:deptno = 20        , FilterName:dept_30
FilterClause:deptno = 30 ]
Conflict Resolution Option:[ Node Priority, Earliest Timestamp ]
Standby Conflict Resolution Option:[ Latest Timestamp, Manual ]
Publication created.
```

11.3.15 Printing a list of publications (printpublist)

The `printpublist` command displays a list of publication names.

Synopsis

```
-printpublist -repsvrfile pubsvrfile
 [ -pubdbid dbid ]
```

```
 [ -printpubid ]
```

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

If you specify the `pubdbid` parameter, only the publication names subordinate to the publication database definition specified by `dbid` are displayed. If you omit the `pubdbid` parameter, all publication names subordinate to the publication server are displayed.

`-printpubid`

Specify this parameter to print the publication IDs and the publication names.

Examples

```
 $ java -jar edb-repcli.jar -printpublist -repsvrfile ~/pubsvrfile.prop
Printing publications ...
analysts_managers
dept_emp
emp_pub
salesemp
```

11.3.16 Printing a list of tables in a publication (printpublishedtables)

The `printpublishedtables` command displays a list of tables and views that belong to the given publication.

Synopsis

```
-printpublishedtables pubname -repsvrfile pubsvrfile
```

Parameters

`pubname`

The name of the publication whose tables and views to display.

`pubsvrfile`

The file containing the publication server login information.

Examples

```
 The tables belonging to publication dept_emp are printed.
$ java -jar edb-repcli.jar -printpublishedtables dept_emp \
>   -repsvrfile ~/pubsvrfile.prop
Printing tables under publication: dept_emp

EDB.DEPT
EDB.EMP
```

11.3.17 Printing a list of filters in a publication (printpubfilterslist)

The `printpubfilterslist` command displays a list of table filters defined in the given publication.

Synopsis

```
-printpubfilterslist pubname -repsvrfile pubsvrfile
```

Parameters

`pubname`

The name of the publication whose table filters to display.

`pubsvrfile`

The file containing the publication server login information.

Examples

This example displays the table filters in publication `analysts_managers`.

```
 $ java -jar edb-repcli.jar -printpubfilterslist analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop
Printing publications ...
FilterID:47      FilterName:jobgrade_11  FilterClause:job IN ('ANALYST', 'MANAGER')
```

This example displays the table filters defined in publication `emp_pub`.

```
$ java -jar edb-repcli.jar -printpubfilterslist emp_pub \
>   -repsvrfile ~/pubsvrfile.prop
Printing publications ...
FilterID:8       FilterName:dept_10_20_30        FilterClause:deptno in (10, 20, 30)
FilterID:9       FilterName:dept_10      FilterClause:deptno = 10
FilterID:10      FilterName:dept_20      FilterClause:deptno = 20
FilterID:16      FilterName:dept_30      FilterClause:deptno = 30
```

11.3.18 Adding tables to a publication (addtablesintopub)

The `addtablesintopub` command adds tables or views into an existing publication.

Synopsis

```
-addtablesintopub pubname
-repsvrfile pubsvrfile
[ { -tables schema_t1.table_1 [schema_t2.table_2] ... | -alltables
    [ schema1 [ schema2 ] ... ] } ]
[ { -views schema_v1.view_1 [ schema_v2.view_2 ] ... | -allviews
    [ schema_v1 [ schema_v2 ] ... ] } ]
[ -tablesfilterclause
    "schema_t1.table_1:filtername_t1:filterclause_t1"
  [ "schema_t2.table_2:filtername_t2:filterclause_t2" ] ...]
[ -viewsfilterclause
    "schema_v1.view_1:filtername_v1:filterclause_v1"
  [ "schema_v2.view_2:filtername_v2:filterclause_v2" ] ...]
[ -enablefilters
  {filtername_1:{subscription_1,subscription_2,... |*}
  [ filtername_2:{subscription_3,subscription_4,... |*} ]... |
  filtername_1:{dbid_1,dbid_2,... |*}
  [ filtername_2:{dbid_3,dbid_4,... |*} ]...} ]
[-defaultconflictresolution {E|L|N|M}]
[-defaultstandbyconflictresolution {E|L|N|M}]
[ -conflictresolution
    schema_t1.table_1:{ E | L | N | M | C:customhandler_t1 }
  [ schema_t2.table_2:{ E | L } N | M | C:customhandler_t2 } ] ...]
[ -standbyconflictresolution
    schema_t1.table_1:{ E | L | N | M | C:customhandler_t1 }
  [ schema_t2.table_2:{ E | L } N | M | C:customhandler_t2 } ] ...]
[ -replicatetableschema {true|false} ]
[ -initialsnapshot [ {true|false} ] ]
```

The `addtablesintopub` command updates an existing publication identified by `pubname`.

The `views` parameter applies only for a snapshot-only publication. It's ignored if the publication isn't defined as snapshot-only. SeeAdding tables to a publication for more information.

> **Note**
>
> The schema names, table names, and view names that you supply as values for the tables and views parameters are case sensitive. Unless quoted identifiers were used to build the database objects, you must enter Oracle names using uppercase letters (for example, `EDB.DEPT`) and EDB Postgres Advanced Server names in lowercase letters (for example `edb.dept`). See Quoted identifiers and default case translation for more information.

> **Note**
>
> Column mappings are applied when you create subscriptions. If you add a table which qualifies for column mapping to the related publication, then Replication Server applies the column mapping for the newly added table. For more information on column mappings, see Creating a subscription.

Parameters

`pubname`

The name of the publication to which to add tables or views.

`pubsvrfile`

The file containing the publication server login information.

`-tables` or `-alltables`

Use these options to add tables into the publication. If the replication type of the publication is `t` (transactional), then `-tables` or `-alltables` is required. Otherwise it's optional.

The `-tables` option adds tables from the tables list to the publication. The `-alltables` option allows you to add all tables of the database or only tables from the listed schemas (optional) to the publication.

`-views or -allviews`

For SMR only: If the replication type of the publication is `s` (snapshot), then you can use `-views` or `-allviews` to include views into the publication.

The `-views` option adds views from the views list to the publication. The `-allviews` option allows you to add all views of the database or only views from the listed schemas (optional) to the publication.

`schema_tn`

The name of the schema containing the nth table of the tables parameter list or the name of the nth schema of the alltables parameter list. This value is case sensitive.

`table_n`

The name of the nth table in the tables parameter list. This value is case sensitive.

`schema_vn`

For SMR only: The name of the schema containing the nth view of the `views` parameter list or the name of the nth schema of the `allviews` parameter list. This value is case sensitive.

`view_n`

For SMR only: The name of the schema containing the nth view of the `views` parameter list. This value is case sensitive.

`filtername_tn`

The filter name to assign to the filter rule on the table.

`filterclause_tn`

The filter clause to apply to the table in the tables parameter list at the position indicated by `schema_t1.table_tn`.

`filtername_vn`

The filter name to assign to the filter rule on the view.

`filterclause_vn`

For SMR only: The filter clause to apply to the view in the views parameter list at the position indicated by `schema_v1.table_vn`.

`-enablefilters`

Enables filters for the list of subscriptions (SMR only) or non-MDN database IDs (MMR only). Enabled filters are applied during the snapshot operation of the tables added to the publication.

`filtername_n`

The filter name to enable for the list of subscriptions/db IDs. Use a name from the filter names specified with options `-tablesfilterclause` and `-viewsfilterclause`.

`subscription_m,subscription_n,...`

For SMR only: Comma-separated (no spaces allowed) list of the names of subscriptions for which to enable the filter `filtername_n`.

`dbid_m,dbid_n,...`

For MMR only: Comma-separated (no spaces allowed) list of the database IDs for which to enable the filter `filtername_n`.

`-defaultconflictresolution`

For MMR only: Default conflict resolution (`E`, `L`, `N`, or `M`), if the `-conflictresolution` option is omitted. The default is `E`.

`-defaultstandbyconflictresolution`

For MMR only: Default standby conflict resolution (`E`, `L`, `N`, or `M`), if the `-standbyconflictresolution` option is omitted. The default is `M`.

`-conflictresolution`

For MMR only: For the `conflictresolution` option, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

The specified conflict resolution applies to the table schema_tn.table_n from the tables parameter list. The default is `E`.

`-standbyconflictresolution`

For MMR only: For the `standbyconflictresolution` option, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

The specified conflict resolution applies to the table schema_tn.table_n from the tables parameter list. The default is `M`.

`customhandler_tn`

For MMR only: For the `conflictresolution` or `standbyconflictresolution` option, specify `customhandler_tn` as the function name. Use an optional schema prefix (formatted as `schema.function_name`) as given in the `CREATE FUNCTION command` for the custom conflict handling function created for the table schema_tn.table_n from the tables parameter list. You must add the custom conflict handling function to the primary definition node. See Adding a custom conflict handling function for an example of adding the custom conflict handling function using PSQL. You must specify the `customhandler` name option only if the conflict resolution option or the standby conflict resolution option is set for custom conflict handling with the `C` value.

`replicatetableschema`

For SMR only: Applies to target subscriptions nodes. Set this option to `true` if you want the publication table definitions replicated from the publication database to all target subscription nodes when adding new tables to the publication. Set this option to `false` if you already created the table definitions in the target subscription nodes. The default is `true`.

For MMR only: Applies to non-MDN nodes. Set this option to `true` if you want the publication table definitions replicated from the master definition node (MDN) when adding tables to the publication. Set this option to `false` if you already created the table definitions in the non-MDN master node. The default is `true`.

`-initialsnapshot`

If the value is `true`, a snapshot replication of the new tables to target nodes is performed. The default value is `false`.

For MMR only: Applies to non-MDN nodes. Specify this option if you want an initial snapshot replication of the new tables performed from the master definition node to every other master node when adding tables to the publication.

For SMR only: Applies to all target subscription nodes. Specify this option if you want an initial snapshot replication of the new tables performed from the publication node to all target subscription nodes when adding tables to the publication.

> **Note**
>
> Unless you intend to use the offline snapshot technique (see Loading tables from an external data source (offline snapshot)), we recommend that you specify this option. You must perform an initial snapshot replication of the new tables before performing synchronization replications on demand (see Performing a synchronization (dosynchronize)) or by a schedule (see Configuring a multi-master schedule (confschedulemmr)). If the newly added tables didn't undergo an initial snapshot to all target nodes, any later synchronization replication might not apply the transactions to that master node.

Examples

This example adds table edb.jobhist and view `edb.salesemp` to an existing publication named `analysts_managers`.

```
$ java -jar edb-repcli.jar -addtablesintopub analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -tables edb.jobhist \
>   -views edb.salesemp
Adding tables to publication analysts_managers ...
```

```
t:[[edb.jobhpst, TABLE], [edb.salesemp, VIEW]]
Filter clause:[null, null]
Publication updated successfully
```

11.3.19 Removing tables from a publication (removetablesfrompub)

The `removetablesfrompub` command removes tables from a publication.

Synopsis

```
-removetablesfrompub pubname
  -repsvrfile pubsvrfile
[ -tables schema_t1.table_1 [ schema_t2.table_2 ] ...]
[ -views schema_v1.view_1 [ schema_v2.view_2 ] ...]
```

See Removing tables from a publication for more information.

> **Note**
>
> The schema names, table names, and view names that you supply as values for the tables and views parameters are case sensitive. Unless quoted identifiers were used to build the database objects, you must use uppercase letters for Oracle names (for example, `EDB.DEPT`) and lowercase letters for EDB Postgres Advanced Server names(for example `edb.dept`). See Quoted identifiers and default case translation for more information.

Parameters

`pubname`

The name of the publication from which to remove tables or views.

`pubsvrfile`

The file containing the publication server login information.

`schema_tn`

The name of the schema containing the nth table of the tables parameter list. This value is case sensitive.

`table_n`

The name of the nth table in the tables parameter list. This value is case sensitive.

`schema_vn`

The name of the schema containing the nth view of the views parameter list. This value is case sensitive.

`view_n`

The name of the nth view in the views parameter list. This value is case sensitive.

Examples

This example removes the table edb.jobhist and view `edb.salesemp` from the `analysts_managers` publication.

```
$ java -jar edb-repcli.jar -removetablesfrompub analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -tables edb.jobhist \
>   -views edb.salesemp

Removing tables and views from publication `analysts_managers` ...

Tables and views removed successfully
```

11.3.20 Adding table filters to a publication (addfilter)

The `addfilter` command adds the definition of table filter rules to the specified publication. This addition makes the filter rules available to enable later on target subscriptions or non-MDN nodes.

Enabling a filter rule on a specified, target subscription or non-MDN node results in the filtering of data from the source table to the target table during replication.

If the filter rule isn't enabled on a target subscription or non-MDN node, then it has no impact on that subscription or non-MDN node during replication. See Enabling filters on a subscription or non-MDN node for information on enabling table filter rules.

Synopsis

```
-addfilter pubname
  -repsvrfile pubsvrfile
[ -tables schema_t1.table_1 [ schema_t2.table_2 ] ...]
[ -views schema_v1.view_1 [ schema_v2.view_2 ] ...]
[ -tablesfilterclause
    "ordinal_t1:filtername_t1:filterclause_t1"
  [ "ordinal_t2:filtername_t2:filterclause_t2" ] ...]
[ -viewsfilterclause
    "ordinal_v1:filtername_v1:filterclause_v1"
  [ "ordinal_v2:filtername_v2:filterclause_v2" ] ...]
```

See Table filters for more information.

> **Note**
>
> The schema names and table or view names that you supply as values for the tables or views parameters are case sensitive. Unless quoted identifiers were used to build the database objects, you must enter Oracle names using uppercase letters (for example, `EDB.DEPT`) and EDB Postgres Advanced Server names in lowercase letters (for example `edb.dept`). See Quoted identifiers and default case translation for more information.

Parameters

`pubname`

The name of the publication in which to add table filters.

`pubsvrfile`

The file containing the publication server login information.

`schema_tn`

The name of the schema containing the nth table of the tables parameter list. This value is case sensitive.

`table_n`

The name of the nth table in the tables parameter list. This value is case sensitive.

`schema_vn`

For SMR only: The name of the schema containing the nth view of the views parameter list. This value is case sensitive.

`view_n`

For SMR only: The name of the nth view in the views parameter list. This value is case sensitive.

`ordinal_tn`

The ordinal number (that is, the position in the list counting from left to right starting with 1) of a table in the tables parameter list to which to apply an attribute.

`filtername_tn`

The filter name to assign to the filter rule on the table.

`filterclause_tn`

The filter clause to applied to the table in the tables parameter list at the position indicated by `ordinal_tn`.

`ordinal_vn`

For SMR only: The ordinal number (that is, the position in the list counting from left to right starting with 1) of a view in the views parameter list to which to apply an attribute.

`filtername_vn`

The filter name to assign to the filter rule on the view.

`filterclause_vn`

For SMR only: The filter clause to apply to the view in the views parameter list at the position indicated by `ordinal_vn`.

Examples

This example adds a table filter to the table edb.emp in the publication `analysts_managers`.

```
$ java -jar edb-repcli.jar -addfilter analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -tables edb.emp \
>   -tablesfilterclause "1:jobgrade_9:job = 'SALESMAN'"
Creating Filter(s)
Tables:[[edb.emp, TABLE]]
Filter clause:[FilterName:jobgrade_9    FilterClause:job = 'SALESMAN'   ]
Filter(s) created successfully.
```

11.3.21 Updating table filters in a publication (updatefilter)

The `updatefilter` command changes the filter clauses of the specified tables or views.

Synopsis

```
-updatefilter pubname
  -repsvrfile pubsvrfile
  -tablesfilterclause
    "filterid_1:filterclause_1"
  [ "filterid_2:filterclause_2" ] ...
```

The next replication to any target subscriptions or non-MDN nodes on which these filter rules were enabled reflects the changes to the filter clauses.

See Table filters for more information.

Parameters

`pubname`

The name of the publication in which to update the filter clauses.

`pubsvrfile`

The file containing the publication server login information.

`filterid_n`

Filter ID identifying the filter rule for which to change the filter clause. Use the `printpubfilterslist` command to get the filter IDs for the available filter rules in the publication (see Printing a list of filters in a publication).

`filterclause_n`

The new filter clause to use.

Examples

This example modifies filter clause with filter `ID 26` in publication `analysts_managers`.

```
$ java -jar edb-repcli.jar -updatefilter analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -tablesfilterclause "26:job = 'CLERK'"
Updating Filter(s)
Filter clause:[26:job = 'CLERK']
Filter(s) updated successfully.
```

11.3.22 Removing a table filter from a publication (removefilter)

The `removefilter` command removes the table filter from the specified publication.

Synopsis

```
-removefilter pubname
  -repsvrfile pubsvrfile
  -filterid filterid
```

The removed filter rule no longer applies to any target subscriptions or non-MDN nodes on which the filter rule was enabled.

See Table filters for more information.

Parameters

`pubname`

The name of the publication containing the filter rule to remove.

`pubsvrfile`

The file containing the publication server login information.

`filterid`

Filter ID identifying the filter rule to remove. Use the `printpubfilterslist` command to get the filter IDs for the filter rules in the publication (see Printing a list of filters in a publication).

Examples

This example removes the filter rule with the filter `ID 26` from the publication `analysts_managers`.

```
$ java -jar edb-repcli.jar -removefilter analysts_managers \
>   -repsvrfile ~/pubsvrfile.prop \
>   -filterid 26
Removing Filter
Filter removed successfully
```

11.3.23 Printing the conflict resolution strategy (printconfresolutionstrategy)

For MMR only: The `printconfresolutionstrategy` command displays the conflict resolution strategy and the standby conflict resolution strategy of the specified table.

Synopsis

```
-printconfresolutionstrategy pubname
  -repsvrfile pubsvrfile
  -table schema_t.table_name
```

See Conflict resolution for more information.

> **Note**
>
> The schema name and table or view name that you supply as values for the table parameter are case sensitive. Unless quoted identifiers were used to build the database objects, you must enter Oracle names using uppercase letters (for example, `EDB.DEPT`) and EDB Postgres Advanced Server names in lowercase letters (for example `edb.dept`). See Quoted identifiers and default case translation for more information.

Parameters

`pubname`

The name of the publication containing the table whose conflict resolution strategy you want to display.

`pubsvrfile`

The file containing the publication server login information.

`schema_t`

The name of the schema containing `table_name`. This value is case sensitive.

`table_name`

The name of the table whose conflict resolution strategy you want to display. This value is case sensitive.

Examples

This example displays the conflict resolution strategy on the EDB Postgres Advanced Server table edb.emp in the publication `emp_pub`.

```
$ java -jar edb-repcli.jar -printconfresolutionstrategy emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -table edb.emp
Primary/Standby Conflict Resolution Strategy...
Conflict Resolution Option:[ Earliest Timestamp ]
Standby Conflict Resolution Option:[ Manual ]
```

11.3.24 Updating the conflict resolution strategy (updateconfresolutionstrategy)

For MMR only: The `updateconfresolutionstrategy` command changes the conflict resolution strategy or standby conflict resolution strategy of the specified table.

Synopsis

```
-updateconfresolutionstrategy pubname
  -repsvrfile pubsvrfile
  -table schema_t.table_name
  -conflictresolution { E | L | N | M | C }
  -standbyconflictresolution { E | L | N | M | C }
[ -customhandlername customhandler ]
```

See Updating the conflict resolution options for more information.

> **Note**
>
> The schema name and table or view name that you supply as values for the table parameter are case sensitive. Unless quoted identifiers were used to build the database objects, you must enter Oracle names using uppercase letters (for example, `EDB.DEPT`) and EDB Postgres Advanced Server names in lowercase letters (for example `edb.dept`). See Quoted identifiers and default case translation for more information.

Parameters

`pubname`

The name of the publication containing the table whose conflict resolution strategy to update.

`pubsvrfile`

The file containing the publication server login information.

`schema_t`

The name of the schema containing `table_name`. This value is case sensitive.

`table_name`

The name of the table whose conflict resolution strategy to update. This value is case sensitive.

`-conflictresolution`

For the `conflictresolution` option, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

`-standbyconflictresolution`

For the `standbyconflictresolution` option, specify one of these values:

- `E` for earliest timestamp conflict resolution
- `L` for latest timestamp conflict resolution
- `N` for node priority conflict resolution
- `M` for manual conflict resolution
- `C` for custom conflict handling

`customhandler`

For the `customhandler` name option, specify `customhandler` as the function name with an optional schema prefix (that is, formatted as schema.function_name) as given in the `CREATE FUNCTION` command for the custom conflict handling function. You must add the custom conflict handling function to the primary definition node. See Adding a custom conflict handling function for an example of adding the custom conflict handling function using PSQL. You must specify the custom handler name option only if you set the conflict resolution option or the standby conflict resolution option for custom conflict handling with the `C` value.

Examples

This example modifies the conflict resolution strategy on the EDB Postgres Advanced Server table edb.emp in the publication `emp_pub` to use latest timestamp conflict resolution with a standby strategy of node priority conflict resolution.

```
$ java -jar edb-repcli.jar -updateconfresolutionstrategy emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -table edb.emp \
>   -conflictresolution L \
>   -standbyconflictresolution N
Updating Primary/Standby Conflict Resolution Strategy...
The Primary/Standby conflict resolution strategies were updated successfully.
```

This example sets custom conflict handling for the `edb.dept` table along with the custom conflict handling function `edb.custom_conflict_dept`.

```
$ java -jar edb-repcli.jar -updateconfresolutionstrategy emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -table edb.dept \
>   -conflictresolution C \
>   -standbyconflictresolution N \
>   -customhandlername edb.custom_conflict_dept
Updating Primary/Standby Conflict Resolution Strategy...
The Primary/Standby conflict resolution strategies were updated successfully.
```

11.3.25 Setting the master definition node (setasmdn)

For MMR only: The `setasmdn` command sets a primary node to the role of master definition node.

Synopsis

```
-setasmdn pubdbid
  -repsvrfile pubsvrfile
```

See Switching the primary definition node for more information on setting the primary definition node.

Parameters

`pubdbid`

The publication database ID of the primary node to be given the role of primary definition node.

`pubsvrfile`

The file containing the publication server login information.

Examples

This example sets the primary node with publication database `ID 9` as the primary definition node.

```
 $ java -jar edb-repcli.jar -setasmdn 9 -repsvrfile ~/pubsvrfile.prop
Updating the database node to be promoted as the new MDN node.
The database has been successfully promoted as the new MDN node.
```

11.3.26 Setting the controller (setascontroller)

The `setascontroller` command designates a publication database as the controller database. The publication database can be the primary database of a single-master replication system or a primary node of a multi-master replication system.

Synopsis

```
-setascontroller pubdbid
  -repsvrfile pubsvrfile
```

See Switching the controller database for information on setting the controller database.

Parameters

`pubdbid`

The publication database ID of the publication database to designate as the controller database.

`pubsvrfile`

The file containing the publication server login information.

Examples

This example sets the publication database with publication database `ID 4` as the controller database.

```
 $ java -jar edb-repcli.jar -setascontroller 4 -repsvrfile ~/pubsvrfile.prop
Updating the Publication database to be promoted as the new Controller database...
The Publication database has been successfully promoted as the new Controller database.
```

11.3.27 Validating a publication (validatepub)

The `validatepub` command checks whether any of the definitions of the tables in the given publication changed since the publication was created.

Synopsis

```
-validatepub pubname
  -repsvrfile pubsvrfile
  -repgrouptype { m | s }
```

> **Note**
>
> The `-repgrouptype` option is not needed with Replication Server 7.1.0 and later.

See Validating a publication for more information.

Parameters

`pubname`

The name of the publication whose tables to validate.

`pubsvrfile`

The file containing the publication server login information.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system.

> **Note**
>
> The `-repgrouptype` option only applies to Replication Server 7.0.0 and earlier.

Examples

This example validates the publication `dept_emp`.

```
$ java -jar edb-repcli.jar -validatepub dept_emp \
>   -repsvrfile ~/pubsvrfile.prop
Validating publication dept_emp ...
All schema of published tables in Publication {0} are up-to-date
```

11.3.28 Validating all publications (validatepubs)

The `validatepubs` command checks whether any of the definitions of the tables subordinate to the given publication database definition changed since the publication was created.

Synopsis

```
-validatepubs
  -repsvrfile pubsvrfile
  -pubdbid dbid
  -repgrouptype { m | s }
```

> **Note**
>
> The `-repgrouptype` option is not needed with Replication Server 7.1.0 and later.

See Validating a publication for more information.

Parameters

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the publication database definition under which to validate all publication tables.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system.

> **Note**
>
> The `-repgrouptype` option only applies to Replication Server 7.0.0 and earlier.

Examples

This example validates the Oracle publication database definition identified by publication database ID 1.

```
$ java -jar edb-repcli.jar -validatepubs \
>   -repsvrfile ~/pubsvrfile.prop \
>   -pubdbid 1
Validating all available publications ...
The schema definitions for all the non snapshot-only publications tables are in sync
with the source.
The "validatepubs" feature is not available for the following snapshot-only publications:
-  salesemp
```

11.3.29 Removing a publication (removepub)

The `removepub` command removes one or more publications.

Synopsis

```
-removepub pubname_1 [ pubname_2 ] ...
  -repsvrfile pubsvrfile
  -repgrouptype { m | s }
  [ -force ]
```

See Removing a publication for more information.

Parameters

`pubname_n`

The name of a publication to remove.

`pubsvrfile`

The file containing the publication server login information.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system.

`-force`

Removes the publication even when one or more subscriptions are defined against the publication.

Examples

This example removes a publication named `dept_emp` from a single-master replication system.

```
$ java -jar edb-repcli.jar -removepub dept_emp \
>   -repsvrfile ~/pubsvrfile.prop -repgrouptype s

Removing publication...
Publication dept_emp unpublished successfully.
```

This example removes a publication named `pub2` by using the `-force` option.

```
$ java -jar edb-repcli.jar -removepub pub2 -repsvrfile pubsvr.prop -repgrouptype s -force

Removing publication via force option (force option will remove the publication even when one or more
subscriptions are defined against the publication)...
Publication pub2 unpublished successfully.
```

11.3.30 Replicating DDL changes (replicateddl)

The `replicateddl` command applies an `ALTER TABLE` statement to a publication table in all databases of a replication system. It also updates the Replication Server insert/update/delete triggers and shadow table associated with that publication table.

Synopsis

```
-replicateddl pubname
  -repsvrfile pubsvrfile
  -table schema_t.table_name
  -ddlscriptfile script_file
```

See Replicating DDL changes for more information.

Parameters

`pubname`

The name of the publication containing the table to which to apply the `ALTER TABLE` statement.

`pubsvrfile`

The file containing the publication server login information.

`schema_t`

The name of the schema containing `table_name`. This value is case sensitive.

`table_name`

The name of the table in the `ALTER TABLE` statement whose definition to modify. This value is case sensitive.

`script_file`

Path to the file containing the `ALTER TABLE` statements.

Examples

This example shows the addition of a column named `title` to table edb.emp. The `ALTER TABLE` statement is in the text file `addcolumn.sql`.

```
 ALTER TABLE edb.emp ADD COLUMN title VARCHAR(20);
The replicateddl command is executed using the addcolumn.sql file to update the triggers and shadow tables
on the primary nodes:
$ java -jar edb-repcli.jar -replicateddl emp_pub
\
>   -repsvrfile ~/pubsvrfile.prop
\
>   -table edb.emp
\
>   -ddlscriptfile
~/addcolumn.sql
DDL changes successfully replicated to all primary
nodes.
```

11.3.31 Adding a subscription database (addsubdb)

For SMR only: The `addsubdb` command adds a subscription database definition.

Synopsis

```
-addsubdb
  -repsvrfile subsvrfile
  -dbtype { oracle | enterprisedb | postgresql | sqlserver }
  -dbhost host
  -dbport port
  -dbuser user
{ -dbpassword encrypted_pwd | -dbpassfile pwdfile }
```

```
[ -oraconnectiontype { sid | servicename } ]
  -database dbname
[ -urloptions jdbc_url_parameters ]
```

The `addsubdb` command creates a new subscription database definition. The `addsubdb` command displays a unique subscription database ID that is assigned to the newly created subscription database definition. The subscription database ID is used to identify the subscription database definition on which to operate when running other Replication Server CLI commands.

See Adding a subscription database for details on the database connection information to supply when adding a subscription database definition.

Parameters

`subsvrfile`

The file containing the subscription server login information.

`-dbtype`

Specify the values as shown in the table.

Value	Database
`oracle`	Oracle
`enterprisedb`	EDB Postgres Advanced Server database in Oracle-compatible configuration mode
`postgresql`	PostgreSQL database or an EDB Postgres Advanced Server database in PostgreSQL-compatible configuration mode
`sqlserver`	Microsoft SQL Server

`host`

The IP address of the host where the subscription database server is running.

`port`

The port number on which the database server is listening for connections.

`user`

The subscription database user name.

`encrypted_pwd`

The encrypted password of the subscription database user. See Encrypting passwords to learn how to use the `encrypt` command to generate an encrypted password.

`pwdfile`

The file containing the encrypted password of the subscription database user.

`-oraconnectiontype`

Specify `sid` to use the `Oracle system ID (SID)` to identify the subscription database in the database parameter. Specify `servicename` to use the Oracle service name to identify the subscription database in the database parameter.

> **Note**
>
> For Oracle 12c, use the service name.

`dbname`

The Postgres or SQL Server database name, the `Oracle SID`, or the Oracle service name of the subscription database.

`jdbc_url_parameters`

Extended usage of JDBC URL parameters such as for support of SSL connectivity. (See Using secure sockets layer (SSL) connections for information on SSL connectivity to the subscription database.)

Examples

This example adds a subscription database definition for an Oracle database. The encrypted password is given on the command line with the `dbpassword` parameter. A subscription database ID of 1 is assigned to the database by the subscription server.

```
$ java -jar edb-repcli.jar -addsubdb -repsvrfile ~/subsvrfile.prop \
>   -dbtype oracle -dbhost 192.168.2.6 -dbport 1521 \
>   -dbuser subuser -dbpassword ygJ9AxoJEX854elcVIJPTw== \
>   -oraconnectiontype sid \
>   -database xe
Adding Subscription Database...
Subscription database added successfully. Subscription Database id:1
```

This example adds a subscription database definition for an EDB Postgres Advanced Server database. The encrypted password is read from a file named `pwdfile` with the `dbpassfile` parameter. A subscription database ID of 2 is assigned to the database by the subscription server.

```
$ java -jar edb-repcli.jar -addsubdb -repsvrfile ~/subsvrfile.prop \
>   -dbtype enterprisedb -dbhost 192.168.2.7 -dbport 5444 \
>   -dbuser subuser -dbpassfile ~/pwdfile \
>   -database subdb
Adding Subscription Database...
Subscription database added successfully. Subscription Database id:2
```

11.3.32 Printing subscription database IDs (printsubdbids)

For SMR only: The `printsubdbids` command displays the subscription database IDs of the subscription database definitions.

Synopsis

`-printsubdbids -repsvrfile subsvrfile`

Parameters

`subsvrfile`

The file containing the subscription server login information.

Examples

This example lists the subscription database IDs of the subscription database definitions.

```
 $ java -jar edb-repcli.jar -printsubdbids -repsvrfile ~/subsvrfile.prop
Printing subscription database ids...
1
2
```

11.3.33 Printing Subscription Database Details (printsubdbidsdetails)

For SMR only: The `printsubdbidsdetails` command prints the connection information for each subscription database definition.

Synopsis

`-printsubdbidsdetails -repsvrfile subsvrfile`

The output has the following components:

`dbid:host:port:dbname:user`

> **Note**
>
> The database user's password is not displayed.

Parameters

`subsvrfile`

The file containing the subscription server login information.

`dbid`

The subscription database ID assigned to the subscription database definition.

`host`

The IP address of the host where the subscription database server is running.

`port`

The port number on which the database server is listening for connections.

`dbname`

The Postgres or SQL Server database name, the Oracle SID, or the Oracle service name of the subscription database.

`user`

The subscription database user name.

Examples

This example displays the subscription database definitions subordinate to the subscription server identified by the content of file `subsvrfile.prop`.

```
$ java -jar edb-repcli.jar -printsubdbidsdetails \
>   -repsvrfile ~/subsvrfile.prop
Printing subscription database ids with details...
id:host:port:database|sid:user
1:192.168.2.6:1521:xe:subuser
2:192.168.2.7:5444:subdb:subuser
```

11.3.34 Updating a subscription database (updatesubdb)

For SMR only: The `updatesubdb` command lets you change the connection information for an existing subscription database definition identified by its subscription database ID.

Synopsis

```
-updatesubdb
  -repsvrfile subsvrfile
  -subdbid dbid
  -dbhost host
  -dbport port
  -dbuser user
{ -dbpassword encrypted_pwd | -dbpassfile pwdfile }
[ -oraconnectiontype { sid | servicename } ]
  -database dbname
[ -urloptions jdbc_url_parameters ]
```

The subscription database definition to update is identified by the `subdbid` parameter.

See Adding a subscription database for details on the database connection information to supply for a subscription database definition.

Parameters

`subsvrfile`

The file containing the subscription server login information.

`dbid`

The subscription database ID of the subscription database definition to update.

`host`

The IP address of the host where the subscription database server is running.

`port`

The port number on which the database server is listening for connections.

`user`

The subscription database user name.

`encrypted_pwd`

The password of the database user in encrypted form. See Encrypting passwords to learn how to use the `encrypt` command to generate an encrypted password.

`pwdfile`

The file containing the password of the database user in encrypted form.

`-oraconnectiontype`

Specify `sid` to use the Oracle system ID (SID) to identify the subscription database in the database parameter. Specify `servicename` to use the Oracle service name to identify the subscription database in the database parameter.

> **Note**
>
> For Oracle 12c, use the service name.

`dbname`

The Postgres or SQL Server database name, the Oracle SID, or the Oracle service name of the subscription database.

`jdbc_url_parameters`

Extended usage of JDBC URL parameters such as for support of SSL connectivity. (See Using secure sockets layer (SSL) connections for information on SSL connectivity to the subscription database.) Specifying the `urloptions` parameter completely replaces any existing JDBC URL parameters that were previously specified with this database. Omitting the `urloptions` parameter deletes any existing JDBC URL parameters that were previously specified with this database.

Examples

This example updates an existing subscription database definition with subscription database ID 2.

```
$ java -jar edb-repcli.jar -updatesubdb -repsvrfile ~/subsvrfile.prop \
>   -subdbid 2 \
>   -dbhost 192.168.2.7 -dbport 5444 \
>   -dbuser subuser -dbpassfile ~/pwdfile \
>   -database subdb
Updating subscription database ...
Subscription database with ID 2 is updated successfully.
```

11.3.35 Removing a subscription database (removesubdb)

For SMR only: The `removesubdb` command removes a subscription database definition.

Synopsis

```
-removesubdb -repsvrfile subsvrfile -subdbid dbid
```

The subscription database definition to remove is identified by the `subdbid` parameter.

See Removing a subscription database for more information.

Parameters

`subsvrfile`

The file containing the subscription server login information.

`dbid`

The subscription database ID of the subscription database definition to remove.

Examples

This example removes the subscription database definition identified by subscription database ID 2.

```
$ java -jar edb-repcli.jar -removesubdb -repsvrfile ~/subsvrfile.prop \
>   -subdbid 2
Removing Subscription Database...
Subscription Database removed.
```

11.3.36 Creating a subscription (createsub)

For SMR only: The `createsub` command creates a new subscription.

Synopsis

```
-createsub subname
  -subsvrfile subsvrfile
  -subdbid dbid
  -pubsvrfile pubsvrfile
  -pubname pubname
[ -filterrule filterid_1[,filterid_2 ] ...]
[-customcolmap <user provided column mappings separated by a semicolon> | -customcolmapfile <path of file
containing user provided column mappings>]
```

The `createsub` command adds a subscription subordinate to the subscription database definition with the subscription database ID given by parameter `subdbid.`

See Adding a subscription for more information.

Parameters

`subname`

The subscription name for the new subscription.

`subsvrfile`

The file containing the subscription server login information of the subscription server under which the new subscription is subordinate.

`dbid`

The subscription database ID of the subscription database definition to which to add the subscription subordinate to.

`pubsvrfile`

The file containing the publication server login information of the publication server under which the publication is subordinate. This is the publication the new subscription is associated with.

`pubname` The publication to which to associate the new subscription.

`filterid_n`

Comma-separated list of filter IDs identifying the filter rules from the set of available table filters to enable on the corresponding tables in the new subscription. Use the `printpubfilterslist` command to obtain the filter IDs for the available filter rules in the publication (see Printing a list of filters in a Publication). Don't use white space between the comma and filter IDs.

`customcolmap`

The column mappings between different but compatible columns in the source and target database. Table-qualified column names can be used in mappings, or a regular expression can be used to implement a mapping rule on multiple columns.

> **Note**
>
> Column mapping can be added only with the Replication Server CLI; this feature isn't supported by Replication Console. Column mappings are supported only for single-master replication clusters.

> Column data type mapping supports snapshots and synchronization between two databases having different but compatible column data types. For example, the data types of a column may be different but data can be populated without any loss. Ensure that there is no possibility of data loss while providing the column data type mapping. This is very important when adding column mapping for columns that act as primary or unique keys as data or precision loss may create inconsistent data across the source and target database.
>
> Column mapping is currently not supported for large object type columns.

To enter multiple mappings, use a semicolon to separate the column mappings.

Enter mappings in the following format, where `dataType` represents the data type of the column in the target database.

```
tableName\.columName=dataType
```

> **Note**
>
> If the table-qualified `columnName` on which the mapping is applied is also present in another schema, then the provided column mapping is applied on both the table-qualified columns.

The following is an example of custom column mapping:

```
emp\.emp_id=integer
```

The `\` characters act as an escape string, while `.` is a reserved character in regular expressions. Use `\.` to represent the `.` character.

The following is an example of regular expression for a column mapping entery:

```
.*id=integer
```

Where you map any column whose name ends in `id` to type `integer`.

`customcolmapfile`

The file containing the column mappings. Multiple custom type mappings can also be provided in the file. You can use either the fully qualify column name or regression expression. Enter each mapping on a new line.

For example, create the file `/usr/edb/xdb/replicator/bin/columnMapping.txt` with the following contents:

```
uuidasprimarykey\\.id=char
mapjson1\\..*=clob
.*no=numeric
```

> **Note**
>
> Often, a single `\` acts as an escape character for the `.` operator. However, this file requires a `\\` for the `.` operator in order to read and write its contents. The `\.` acts as a separator and the single `.` operator acts as regular expression in this case.

And enter the complete path for the file in the `customcolmapfile` option:

```
-customcolmapfile /usr/edb/xdb/replicator/bin/columnMapping.txt
```

Examples

This example creates a subscription named `dept_emp_sub` in the EDB Postgres Advanced Server subscription database identified by subscription database ID 2. The subscription is associated with a publication named `dept_emp`.

```
$ java -jar edb-repcli.jar -createsub dept_emp_sub \
>   -subsvrfile ~/subsvrfile.prop \
>   -subdbid 2 \
>   -pubsvrfile ~/pubsvrfile.prop \
>   -pubname dept_emp
Creating subscription...
Subscription created successfully
```

11.3.37 Printing a subscription list (printsublist)

For SMR only: The `printsublist` command displays a list of subscription names.

Synopsis

```
-printsublist -repsvrfile subsvrfile -subdbid dbid
```

Parameters

`subsvrfile`

The file containing the subscription server login information.

`dbid`

The subscription names subordinate to the subscription database definition specified by `dbid` are displayed.

Examples

```
$ java -jar edb-repcli.jar -printsublist -repsvrfile ~/subsvrfile.prop \
>   -subdbid 2
Printing subscriptions ...
dept_emp_sub
```

11.3.38 Printing the column mappings added to a publication (printusercolmapping)

For SMR only: The `printusercolmapping` command prints column data type mapping between the publication and subscription database.

Synopsis

```
-printusercolmapping <pubname> -repsvrfile <pubsvrfile> [-subname <subname>] [-details]
```

Parameters

`pubname`

The name of the publication whose column mappings are displaying.

`pubsvrfile`

The file containing the publication server login information.

`subname`

The name of the subscription whose column mappings are displaying. Providing the name allows for the column mappings to be filtered according to the subscription name.

`details`

Provides additional details about implicit column mappings and internal mapping representation of the column mapping. Implicit column mappings are'nt provided by the user. They are added by default depending on source and target database and their versions. For example, JSON is'nt supported on Oracle database versions 21 and earlier. Memory mapping is referred to as internal mapping in Replication Server.

Examples

The following is an example where `-details` isn't included:

```
$ java -jar edb-repcli.jar -printusercolmapping emp_pub -repsvrfile ~/pubsvrfile.prop

Printing user column mapping details...

Publication name:Subscription name:User expressions
emp_pub:emp_sub:uuidasprimarykey.id=char;public.mapjson.details=varchar2(4000);.*no=numeric;mapjson1..*=clo
b
```

The following is an example where `-details` is included:

```
java -jar edb-repcli.jar -printusercolmapping emp_pub -repsvrfile ~/pubsvrfile.prop  -details

Printing user column mapping details...

Publication name:Subscription name:User expressions
emp_pub:emp_sub:uuidasprimarykey.id=char;public.mapjson.details=varchar2(4000);.*no=numeric;mapjson1..*=clo
b

Printing implicit column mapping details (Not provided by user)...

Publication name:Subscription name:Database incompatible datatype-->Database compatible datatype
```

```
emp_pub:emp_sub1:json-->varchar
Note: Please provide custom column mapping with -customcolmap argument to override implicit mapping.

Printing internal column mapping representation...

Publication name:Subscription name:Table name:Column name:Source datatype-->Target datatype
emp_pub:emp_sub:public.test123:deptbuildno:integer-->number
emp_pub:emp_sub:public.test123:deptno:numeric-->number
emp_pub:emp_sub1:public.mapjson1:details:json-->varchar
emp_pub:emp_sub1:public.mapjson:details:json-->varchar
```

> **Note**
>
> User provided column data type mappings override implicit column data type mappings when they are present in the same column.

11.3.39 Enabling filters on a subscription or non-MDN node (enablefilter)

The `enablefilter` command enables one or more filter rules on a single-master replication system subscription or on a multi-master replication system primary node other than the primary definition node.

Use the `enablefilter` command when you want to apply a filter rule to a subscription or a non-MDN node, but the filter rule didn't yet exist or it wasn't included with the subscription or the non-MDN node when these components were created.

Synopsis

```
-enablefilter
  -repsvrfile pubsvrfile
{ -subname subname | -dbid dbid }
  -filterids filterid_1 [ filterid_2 ] ...
```

Enabling a filter rule applies it to the specified, target subscription or non-MDN node and thus filters the data during replication from the source table to the target table.

See Table filters for more information.

Before enabling a filter rule, define it in the source publication in one of several possible ways:

For SMR:

- The table filter was defined in the publication of the primary database when it was first created either by the `createpub` command (see Creating a publication) or by the Replication Server console (see Adding a publication).
- The table filter was added to an existing publication using the `addfilter` command (see Adding table filters to a publication) or by the Replication Server console (see Updating the set of available table filters in a publication).

For MMR:

- The table filter was defined in the publication of the primary definition node when it was first created either by the `createpub` command (see Creating a publication) or by the Replication Server console (see Adding a publication).
- The table filter was added to an existing publication using the `addfilter` command (see Adding table filters to a publication) or by the Replication Server console (see Updating the set of available table filters in a publication).

Enable the filter rule as follows:

For SMR: Use the `enablefilter` command or the Replication Server console (see Enabling and disabling table filters on a subscription).

For MMR: Use the `enablefilter` command or the Replication Server console (see Enabling and disabling table filters on a primary node.

After you enable a filter rule, it filters the data during replication from the source table to the target table. You can disable a filter rule so that it no longer filters the data during replication to the target table (see Disabling filters on a subscription or non-MDN node).

Parameters

`pubsvrfile`

The file containing the publication server login information.

`subname`

For SMR only: The name of the subscription containing the tables on which to enable the filter rules.

`dbid`

For MMR only: The publication database ID of the non-MDN node containing the tables on which to enable the filter rules.

`filterid_n`

One or more filter IDs separated by space characters. These IDs identify the filter rules from the set of available table filters. You can enable these filters on the corresponding tables in the SMR subscription specified by `subname` or in the MMR non-MDN node specified by `dbid`. Use the `printpubfilterslist` command to obtain the filter IDs for the available filter rules in the publication (see Printing a list of filters in a publication).

Examples

This example enables a filter rule on a subscription of a single-master replication system.

```
$ java -jar edb-repcli.jar -enablefilter -repsvrfile ~/pubsvrfile.prop \
>   -subname analysts_managers_sub \
>   -filterids 47
Enabling filters...
Filter rule(s) updated successfully.
```

This example enables multiple filter rules on a primary node that isn't the primary definition node of a multi-master replication system.

```
$ java -jar edb-repcli.jar -enablefilter -repsvrfile ~/pubsvrfile.prop \
>   -dbid 139 \
>   -filterids 8 16
Enabling filters...
Filter rule(s) updated successfully.
```

11.3.40 Disabling filters on a subscription or non-MDN node (disablefilter)

The `disablefilter` command disables one or more filter rules on a single-master replication system subscription or on a multi-master replication system primary node other than the primary definition node.

Synopsis

```
-disablefilter
  -repsvrfile pubsvrfile
{ -subname subname | -dbid dbid }
  -filterids filterid_1 [ filterid_2 ] ...
```

Disabling a filter rule prevents it from being applied to the specified target subscription or non-MDN node and thus doesn't filter the data during replication from the source table to the target table.

See Table filters for more information.

Disable the filter rule as follows:

For SMR: Use the `disablefilter` command or the Replication Server console (see Enabling and disabling table filters on a subscription).

For MMR: Use the `disablefilter` command or the Replication Server console (see Enabling and disabling table filters on a primary node).

Disabling a filter rule doesn't remove its definition from the publication. The filter rule still exists and you can still enable it on target subscriptions or non-MDN nodes.

To remove a filter rule so that it no longer exists, use the `removefilter` command (see Removing a table filter from a publication) or the Replication Server console (see Updating the set of available table filters in a publication).

Parameters

`pubsvrfile`

The file containing the publication server login information.

`subname`

For SMR only: The name of the subscription containing the tables on which to disable the filter rules.

`dbid`

For MMR only: The publication database ID of the non-MDN node containing the tables on which to disable the filter rules.

`filterid_n`

One or more filter IDs separated by space characters. These IDs identify the currently enabled table filters to disable in the SMR subscription specified by `subname` or in the MMR non-MDN node specified by `dbid`.

Examples

This example disables a filter rule on a subscription of a single-master replication system.

```
 $ java -jar edb-repcli.jar -disablefilter -repsvrfile ~/pubsvrfile.prop \
>   -subname analysts_managers_sub \
>   -filterids 47
Disabling filters...
Filter rule(s) updated successfully.
```

This example disables multiple filter rules on a primary node that isn't the primary definition node of a multi-master replication system.

```
 $ java -jar edb-repcli.jar -disablefilter -repsvrfile ~/pubsvrfile.prop \
>   -dbid 139 \
>   -filterids 8 16
Disabling filters...
Filter rule(s) updated successfully.
```

11.3.41 Taking a single-master snapshot (dosnapshot)

For SMR only: The `dosnapshot` command performs snapshot synchronization on the specified subscription in a single-master replication system.

Synopsis

```
 -dosnapshot subname -repsvrfile subsvrfile
[ -verboseSnapshotOutput { true | false } ]
```

See Performing snapshot replication for more information.

Parameters

`subname`

The name of the subscription for which to take the snapshot.

`subsvrfile`

The file containing the subscription server login information.

`-verboseSnapshotOutput`

Set this option to `true` if you want to display the output from the snapshot. Set this option to `false` if you don't want to display the snapshot output. The default is `true`.

Examples

This example performs snapshot replication on subscription `dept_emp_sub`.

```
$ java -jar edb-repcli.jar -dosnapshot dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop
Performing snapshot...
Source database connectivity info...
conn =jdbc:oracle:thin:@192.168.2.6:1521:xe
user =pubuser
password=******
Target database connectivity info...
conn =jdbc:edb://192.168.2.7:5444/subdb
user =subuser
password=******
Connecting with source Oracle database server...
Connecting with target EnterpriseDB database server...
Importing redwood schema EDB...
Table List: 'DEPT','EMP'
Loading Table Data in 8 MB batches...
Disabling FK constraints & triggers on edb.dept before truncate...
Truncating table DEPT before data load...
Disabling indexes on edb.dept before data load...
Loading Table: DEPT ...
[DEPT] Migrated 4 rows.
[DEPT] Table Data Load Summary: Total Time(s): 0.182 Total Rows: 4
Disabling FK constraints & triggers on edb.emp before truncate...
Truncating table EMP before data load...
Disabling indexes on edb.emp before data load...
Loading Table: EMP ...
[EMP] Migrated 14 rows.
[EMP] Table Data Load Summary: Total Time(s): 0.178 Total Rows: 14
Enabling FK constraints & triggers on edb.dept...
Enabling indexes on edb.dept after data load...
Enabling FK constraints & triggers on edb.emp...
Enabling indexes on edb.emp after data load...
Performing ANALYZE on EnterpriseDB database...
Data Load Summary: Total Time (sec): 1.866 Total Rows: 18 Total Size(MB): 0.0

Schema EDB imported successfully.


Migration process completed successfully.

Migration logs have been saved to /var/log/xdb-rep/build57l

******************** Migration Summary ********************
Tables: 2 out of 2
Constraints: 4 out of 4

Total objects: 6
Successful count: 6
Failure count: 0

************************************************************
Snapshot taken successfully.
```

11.3.42 Take a multi-master snapshot (dommrsnapshot)

For MMR only: The `dommrsnapshot` command performs snapshot synchronization on the specified primary node in a multi-master replication system.

Synopsis

```
-dommrsnapshot pubname
  -repsvrfile pubsvrfile
  -pubhostdbid dbid
[ -verboseSnapshotOutput { true | false } ]
```

Parameters

`pubname`

The name of the publication for which to take the snapshot.

`pubsvrfile`

The file containing the publication server login information.

`dbid`

The publication database ID of the target primary node for the snapshot replication.

`-verboseSnapshotOutput`

Set this option to `true` if you want to display the output from the snapshot. Set this option to `false` if you don't want to display the snapshot output. The default is `true`.

Examples

This example performs snapshot replication on publication `emp_pub` to the target primary node identified by publication database ID 9.

```
$ java -jar edb-repcli.jar -dommrsnapshot emp_pub \
>   -pubhostdbid 9 \
>   -repsvrfile ~/pubsvrfile.prop
Performing snapshot...
Source database connectivity info...
conn =jdbc:edb://192.168.2.6:5444/edb
user =pubuser
password=******
Target database connectivity info...
conn =jdbc:edb://192.168.2.7:5444/MMRnode
user =MMRuser
password=******
Connecting with source EnterpriseDB database server...
Connecting with target EnterpriseDB database server...
Importing enterprisedb schema edb...
Table List: 'dept','emp'
```

```
Loading Table Data in 8 MB batches...
Disabling FK constraints & triggers on edb.dept before truncate...
Truncating table dept before data load...
Disabling indexes on edb.dept before data load...
Loading Table: dept ...
[dept] Migrated 5 rows.
[dept] Table Data Load Summary: Total Time(s): 0.247 Total Rows: 5
Disabling FK constraints & triggers on edb.emp before truncate...
Truncating table emp before data load...
Disabling indexes on edb.emp before data load...
Loading Table: emp ...
[emp] Migrated 14 rows.
[emp] Table Data Load Summary: Total Time(s): 0.163 Total Rows: 14
Enabling FK constraints & triggers on edb.dept...
Enabling indexes on edb.dept after data load...
Enabling FK constraints & triggers on edb.emp...
Enabling indexes on edb.emp after data load...
Performing ANALYZE on EnterpriseDB database...
Data Load Summary: Total Time (sec): 0.8 Total Rows: 19 Total Size(MB): 0.0

Schema edb imported successfully.


Migration process completed successfully.

Migration logs have been saved to /var/log/xdb-rep/build57l

******************** Migration Summary ********************
Tables: 2 out of 2
Constraints: 4 out of 4

Total objects: 6
Successful count: 6
Failure count: 0

************************************************************
Snapshot taken successfully.
```

11.3.43 Performing a synchronization (dosynchronize)

The `dosynchronize` command performs synchronization replication on the specified subscription for a single-master replication system or for an entire multi-master replication system.

Synopsis

```
-dosynchronize { subname | pubname }
  -repsvrfile { subsvrfile | pubsvrfile }
[ -repgrouptype { s | m } ]
For a single-master replication system use:
-dosynchronize subname -repsvrfile subsvrfile
```

For a multi-master replication system use:

```
-dosynchronize pubname -repsvrfile pubsvrfile -repgrouptype m
```

> **Note**
>
> **For SMR only**: You can use the `dosynchronize` command on a subscription without first having to perform a snapshot using the `dosnapshot` command. The `dosynchronize` command automatically performs the first required snapshot.

> **Note**
>
> **For MMR only**: Be sure an initial snapshot replication was performed from the primary definition node to every other primary node in the multi-master replication system. If a newly added primary node didn't undergo an initial snapshot, any later synchronization replication might not apply the transactions to that primary node. You can take the initial snapshot when you add the primary node (see Creating more primary nodes or Adding a publication database). Alternatively, you can perform an on-demand snapshot (see Performing snapshot replication or Take a multi-master snapshot).

See Performing synchronization replication for more information on performing synchronization replication for a single-master replication system. See Performing synchronization replication for a multi-master replication system.

Parameters

`subname`

For SMR only: The name of the subscription for which to perform synchronization replication.

`pubname`

For MMR only: The name of the publication for which to perform synchronization replication.

`subsvrfile`

For SMR only: The file containing the subscription server login information.

`pubsvrfile`

For MMR only: The file containing the publication server login information.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system. The default is `s`.

Examples

This example performs synchronization replication on subscription `dept_emp_sub` of a single-master replication system.

```
 $ java -jar edb-repcli.jar -dosynchronize dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop
Performing synchronize...
Synchronize done successfully.
```

This example performs synchronization replication on publication `emp_pub` of a multi-master replication system. The `-repgrouptype m` parameter is required in this case.

```
$ java -jar edb-repcli.jar -dosynchronize emp_pub \
>   -repsvrfile ~/pubsvrfile.prop -repgrouptype m
Performing synchronize...
Publication synchronized successfully.
```

11.3.44 Configuring a single-master schedule (confschedule)

For SMR only: The `confschedule` command creates a schedule for when to start recurring replications for a single-master replication system.

Synopsis

```
-confschedule subname -repsvrfile subsvrfile
{ -remove | -jobtype { s | t }
  { -realtime no_of_sec |
    -daily hour minute |
    -weekly day_of_week hour minute |
    -monthly month day_of_month hour minute |
    -cronexpr "cron_expression"
  }
}
```

Specifying the `remove` parameter deletes the schedule from the subscription. In this case, you can specify only the `subname` and `repsvrfile` parameters.

If you omit the `remove` parameter, then you must specify the `jobtype` parameter and one of the parameters `realtime`, `daily`, `weekly`, `monthly`, or `cronexpr`. You must also specify the `subname` and `repsvrfile` parameters. The new schedule replaces any existing schedule for subscription `subname`. See Performing synchronization replication for more information on creating a schedule.

Parameters

`subname`

The name of the subscription whose replication schedule you want to create.

`subsvrfile`

The file containing the subscription server login information.

`-remove`

Specifying the `remove` parameter removes any existing schedule from the subscription. If you don't specify the `remove` parameter, then a schedule is created for the subscription.

`-jobtype`

Specify `s` to perform the scheduled replication by snapshot. Specify `t` to perform the scheduled replication by synchronization. If the associated publication is a snapshot-only publication, then you must specify `-jobtype s`.

`no_of_sec`

The number of seconds between scheduled replications. Specify an integer greater than 0.

`hour`

The hour of the day based on a 24-hour clock. Specify an integer from 0 to 23.

`minute`

The minute of the hour. Specify an integer from 0 to 59.

`day_of_week`

The day of the week. Specify any of the following values: `SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, or `SAT`. This value is not case sensitive.

`month`

The month of the year. Specify any of the following values: `JAN`, `FEB`, `MAR`, `APR`, `MAY`, `JUN`, `JUL`, `AUG`, `SEP`, `OCT`, `NOV`, or `DEC`. This value is not case sensitive.

`day_of_month`

The day of the month. Specify an integer 1 or greater up to and including the number of days in month.

`cron_expression`

A cron expression. See Writing a cron expression for details.

Examples

This example creates a schedule to perform synchronization replication on subscription `dept_emp_sub` once every five minutes.

```
$ java -jar edb-repcli.jar -confschedule dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop \
>   -jobtype t \
>   -realtime 300
Configuring scheduler ...
Job is successfully scheduled.
```

This example removes the schedule from subscription `dept_emp_sub.`

```
$ java -jar edb-repcli.jar -confschedule dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop \
>   -remove
Configuring scheduler ...
Scheduled job is removed.
```

11.3.45 Configuring a multi-master schedule (confschedulemmr)

For MMR only: The `confschedulemmr` command creates a schedule for when to start recurring replications for a multi-master replication system.

> **Note**
>
> Be sure an initial snapshot replication was performed from the primary definition node to every other primary node in the multi-master replication system. If a newly added primary node didn't undergo an initial snapshot, any subsequent synchronization replication started by a schedule might not apply the transactions to that primary node. You can take the initial snapshot when you first add the primary node (see Creating more primary nodes or Adding a publication database) or by performing an on-demand snapshot (see Performing snapshot replication or Take a multi-master snapshot).

Synopsis

```
-confschedulemmr pubdbid -pubname pubname
  -repsvrfile pubsvrfile
{ -remove |
  { -realtime no_of_sec |
    -daily hour minute |
    -weekly day_of_week hour minute |
    -monthly month day_of_month hour minute |
    -cronexpr "cron_expression"
  }
}
```

Specifying the `remove` paramter deletes the schedule from the publication. In this case, you can specify only the `pubdbid, pubname`, and `repsvrfile` parameters.

If you omit the `remove` parameter, then you must specify one of the parameters `realtime`, `daily`, `weekly`, `monthly`, or `cronexpr`. You must also specify the `pubdbid`, `pubname`, and `repsvrfile` parameters. Any existing schedule for publication `pubname` is replaced by the new schedule.

See Creating a schedule for more information.

Parameters

`pubdbid`

The publication database ID of the publication database definition representing the primary definition node on which to configure the schedule.

`pubname`

The name of the publication for which you want to create a replication schedule.

`pubsvrfile`

The file containing the publication server login information.

`-remove`

Specifying the `remove` parameter removes any existing schedule from the publication. If you don't include the `remove` parameter, then a schedule is created for the publication.

`no_of_sec`

The number of seconds between scheduled replications. Specify an integer greater than 0.

`hour`

The hour of the day based on a 24-hour clock. Specify an integer from 0 to 23.

`minute`

The minute of the hour. Specify an integer from 0 to 59.

`day_of_week`

The day of the week. Specify one of the following values: `SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, or `SAT`. This value is not case sensitive.

`month`

The month of the year. Specify one of the following values: `JAN`, `FEB`, `MAR`, `APR`, `MAY`, `JUN`, `JUL`, `AUG`, `SEP`, `OCT`, `NOV`, or `DEC`. This value is not case sensitive.

`day_of_month`

The day of the month. Specify an integer 1 or greater up to and including the number of days in month.

`cron_expression`

A cron expression. See Writing a cron expression for details.

Examples

This example creates a schedule to perform synchronization replication on publication `emp_pub` subordinate to the primary definition node whose publication database ID is 6. Replication occurs daily at 8:00 a.m.

```
$ java -jar edb-repcli.jar -confschedulemmr 6 -pubname emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -daily 8 00
Configuring scheduler ...
Job is successfully scheduled.
```

This example removes the schedule from publication `emp_pub`.

```
$ java -jar edb-repcli.jar -confschedulemmr 6 -pubname emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -remove
Configuring scheduler ...
Scheduled job is removed.
```

11.3.46 Print schedule (printschedule)

The `printschedule` command displays a recurring replication schedule.

Synopsis

```
-printschedule { subname | pubname }
  -repsvrfile { subsvrfile | pubsvrfile }
[ -repgrouptype { s | m } ]
For a single-master replication system use:
-printschedule subname -repsvrfile subsvrfile
```

For a multi-master replication system:

```
-printschedule pubname -repsvrfile pubsvrfile -repgrouptype m
```

Parameters

`subname`

For SMR only: The name of the subscription whose schedule to display.

`pubname`

For MMR only: The name of the publication whose schedule to display.

`subsvrfile`

For SMR only: The file containing the subscription server login information.

`pubsvrfile`

For MMR only: The file containing the publication server login information.

`-repgrouptype`

Specify `s` if this command applies to a single-master replication system. Specify `m` if this command applies to a multi-master replication system. The default is `s`.

Examples

This example displays the schedule for a subscription in a single-master replication system.

```
$ java -jar edb-repcli.jar -printschedule dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop
```

```
Printing subscription schedule ...

Job type                Synchronize

Scheduled time          2012-06-19 13:27:20

Previous fire time      2012-06-19 13:27:20

Next fire time          2012-06-19 13:32:20
```

This example displays the schedule for a publication in a multi-master replication system. The `-repgrouptype m` parameter is required in this case.

```
 $ java -jar edb-repcli.jar -printschedule emp_pub \
>   -repsvrfile ~/pubsvrfile.prop \
>   -repgrouptype m
Printing subscription schedule ...

Job type                Synchronize

Scheduled time          2012-06-19 13:27:55

Previous fire time      Not available

Next fire time          2012-06-20 08:00:00

Cron expression         0 0 8 * * ?
```

11.3.47 Updating a subscription (updatesub)

For SMR only: The `updatesub` command updates certain metadata of a given subscription. This metadata allows the subscription server to find the host running the publication server that manages the publication associated with the subscription.

Synopsis

```
-updatesub subname
  -subsvrfile subsvrfile
  -pubsvrfile pubsvrfile
  -host newpubsvr_ipaddress
  -port newpubsvr_port
```

The `updatesub` command allows you to update the subscription metadata. The metadata consists of the IP address and port number identifying the publication server that is the parent of the publication associated with the subscription.

This metadata is essential to the proper operation of the replication system. It is the means by which the subscription server locates the publication server whenever a replication must be performed on a given subscription. The replication process is always carried out by the publication server that manages the publication associated with the subscription.

Use the `updatesub` command in the scenario in which you built your replication system using IP addresses that are valid at that point in time. At some later point, the IP address assigned to the host running the publication server changed.

You use the `host` and `port` parameters of the `updatesub` command to supply the new network address identifying the publication server.

See Updating a subscription for more information.

Parameters

`subname`

The name of the subscription whose metadata to update.

`subsvrfile`

The file containing the subscription server login information for the subscription server in which subscription `subname` was created.

`pubsvrfile`

The file containing publication server login information for the publication server that manages the publication associated with subscription `subname`. The values that you supply for `newpubsvr_ipaddress` and `newpubsvc_port` must be the same as the values set in fields `host` and `port` in the file `pubsvrfile`.

`newpubsvr_ipaddress`

The new IP address for the publication server that manages the publication associated with subscription `subname`. This value must be the same as the IP address specified for the field `host` in the file `pubsvrfile`.

`newpubsvr_port`

The new port number for the publication server that manages the publication associated with subscription `subname`. This value must be the same as the port number specified for the field `port` in the file `pubsvrfile`.

Examples

If the publication server host IP address changed to `192.168.2.7`, then make sure the publication server login information in file `pubsvrfile.prop` contains the new IP address, as shown by the following:

```
host=192.168.2.7
port=9051
user=enterprisedb
# Password is in encrypted
form.
password=ygJ9AxoJEX854elcVIJPTw==
```

To update the metadata for subscription `dept_emp_sub` so that its subscription server can find the new publication server host, run the following command:

```
$ java -jar edb-repcli.jar -updatesub dept_emp_sub \
>   -subsvrfile ~/subsvrfile.prop \
>   -pubsvrfile ~/pubsvrfile.prop \
>   -host 192.168.2.7 \
>   -port 9051
Updating subscription dept_emp_sub...

Subscription is updated successfully
```

11.3.48 Removing a subscription (removesub)

For SMR only: The `removesub` command removes a subscription.

Synopsis

```
-removesub subname -repsvrfile subsvrfile
```

See Removing a subscription for more information.

Parameters

`subname`

The name of the subscription to remove.

`subsvrfile`

The file containing the subscription server login information.

Examples

```
 A subscription named dept_emp_sub is removed.
$ java -jar edb-repcli.jar -removesub dept_emp_sub \
>   -repsvrfile ~/subsvrfile.prop
Removing subscription...
Subscription removed successfully.
```

11.3.49 Scheduling shadow table history cleanup (confcleanupjob)

The `confcleanupjob` command creates a schedule for when to delete shadow table history.

Synopsis

```
 -confcleanupjob pubdbid -repsvrfile pubsvrfile
{ -disable | -enable
  { -minutely no_of_minutes |
    -hourly no_of_hours |
    -daily hour |
    -weekly day_of_week hour |
    -cronexpr "cron_expression"
```

```
  }
}
```

Specifying the `disable` parameter deletes the schedule. In this case, specify only the `pubdbid` and `pubsvrfile` parameters. If you omit the `disable` parameter, then you must specify the `enable` parameter and one of parameters `minutely`, `hourly`, `daily`, `weekly,` or `cronexpr` along with the `pubdbid` and `pubsvrfile` parameters.

See Scheduling shadow table history cleanup for more information.

Parameters

`pubdbid`

Publication database ID of the publication database definition whose schedule for deleting shadow table history you want to enable or disable.

`pubsvrfile`

The file containing the publication server login information.

`-disable`

Specifying the `disable` paramter removes any existing shadow table history cleanup schedule from the publication database definition. If you don't spcify the `disable` parameter, then you must specify the `enable` parameter.

`-enable`

Establish a schedule for shadow table history cleanup.

`no_of_minutes`

The number of minutes between scheduled shadow table history cleanup jobs. Specify an integer from 1 through 59.

`no_of_hours`

The number of hours between scheduled shadow table history cleanup jobs. Specify an integer from 1 through 12.

`hour`

The hour of the day based on a 24-hour clock. Specify an integer from 0 through 23.

`day_of_week`

The day of the week. Specify any of the following values: `SUNDAY`, `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, or `SATURDAY`. This value is not case sensitive.

`cron_expression`

A cron expression. See Writing a cron expression for details.

Examples

This example schedules shadow table history cleanup to run once every three hours on the shadow tables created in the publication database definition with ID 1.

```
$ java -jar edb-repcli.jar -confcleanupjob 1 \
>   -repsvrfile ~/pubsvrfile.prop \
>   -enable -hourly 3
Configuring cleanup job ...
Cleanup job configured.
```

This example schedules a shadow table history cleanup to run once a day at 6:00 p.m. on the shadow tables created in the publication database definition identified by ID 1.

```
$ java -jar edb-repcli.jar -confcleanupjob 1 \
>   -repsvrfile ~/pubsvrfile.prop \
>   -enable -daily 18
Configuring cleanup job ...
Cleanup job configured.
```

This example schedules the shadow table history cleanup to run every Wednesday at 8:00 a.m. on the shadow tables created in the publication database definition identified by ID 1.

```
$ java -jar edb-repcli.jar -confcleanupjob 1 \
>   -repsvrfile ~/pubsvrfile.prop \
>   -enable –weekly WEDNESDAY 8
Configuring cleanup job ...
Cleanup job configured.
```

This example disables the shadow table history cleanup job on the publication database definition identified by ID 1.

```
$ java -jar edb-repcli.jar -confcleanupjob 1 \
>   -repsvrfile ~/pubsvrfile.prop -disable
Configuring cleanup job ...
Cleanup job removed.
```

11.3.50 Cleaning up shadow table history (cleanshadowhistforpub)

The `cleanshadowhistforpub` command deletes the shadow table history for the specified publication.

Synopsis

```
-cleanshadowhistforpub pubname
  -repsvrfile pubsvrfile
[ -mmrdbid dbid_1[,dbid_2 ] ...]
```

See Cleaning up shadow table history for more information.

Parameters

`pubname`

The name of the publication whose shadow table history to delete.

`pubsvrfile`

The file containing the publication server login information.

`dbid_n`

For MMR only: The publication database ID of the primary node whose shadow table history to delete. This parameter is required for a multi-master replication system specifying one or more comma-separated publication database IDs. Don't include white space between the comma and publication database IDs.

Examples

This example deletes the shadow table history for publication `dept_emp`.

```
$ java -jar edb-repcli.jar -cleanshadowhistforpub dept_emp \
>   -repsvrfile ~/pubsvrfile.prop
Removing shadow table's transaction history ...

Shadow table's transaction history removed successfully.
```

11.3.51 Cleaning up replication history (cleanrephistoryforpub)

The `cleanrephistoryforpub` command deletes the replication history for the specified publication.

Synopsis

`-cleanrephistoryforpub pubname -repsvrfile pubsvrfile`

See Cleaning up replication history for more information.

Parameters

`pubname`

The name of the publication whose replication history to delete.

`pubsvrfile`

The file containing the publication server login information.

Examples

This example deletes replication history for publication `dept_emp.`

```
$ java -jar edb-repcli.jar -cleanrephistoryforpub dept_emp \
>   -repsvrfile ~/pubsvrfile.prop
Removing publication's replication history ...

Replication history has been removed.
```

11.3.52 Cleaning up all replication history (cleanrephistory)

The `cleanrephistory` command deletes the replication history for all publications in the specified publication server.

Synopsis

`-cleanrephistory -repsvrfile pubsvrfile`

See Cleaning up replication history for more information.

Parameters

`pubsvrfile`

The file containing the publication server login information.

Examples

This example deletes replication history for all publications in the publication server identified by the content of the file `pubsvrfile.prop`.

```
$ java -jar edb-repcli.jar -cleanrephistory -repsvrfile ~/pubsvrfile.prop
Removing all publication's replication history ...

Replication history has been removed.
```

11.3.53 Reloading the publication or subscription server configuration file (reloadconf)

The `reloadconf` command applies any changes in the publication or subscription server configuration file (`XDB_HOME/etc/xdb_pubserver.conf` or `XDB_HOME/etc/xdb_subserver.conf`) without restarting the server.

Synopsis

```
-reloadconf
 -repsvrfile svrfile
```

Parameters

`svrfile`

The file containing the publication or subscription server login information.

The table shows whether a configuration property can be reloaded.

Property name	Property can be reloaded
Properties shared by publication server and subscription server	
`java.rmi.server.hostname`	No
`logging.level`	Yes
`logging.file.size`	No
`logging.file.count`	No
`importPartitionAsTable`	Yes
`skipCheckConst`	Yes
`snapshotParallelLoadCount`	Yes
`snapshotParallelLoadRowLimit`	Yes
`importPartitionAsTable`	Yes
`sslTrustStore`	Yes
`sslTrustStorePassword`	Yes
`sslTrustStoreType`	Yes
`sslKeyStore`	Yes
`sslKeyStoreType`	Yes
`sslKeyStorePassword`	Yes
Properties used by subscription server	
`skipTablePrivileges`	Yes
Properties used by publication server	
`skipTablePrivileges`	Yes
`replaceNullChar`	Yes
`nullReplacementChar`	Yes
`escapeTabDelimiter`	Yes
`mtkCopyDelimiter`	Yes
`enableConstBeforeDataLoad`	Yes
`cpBatchSize`	Yes
`batchSize`	Yes

Property name	Property can be reloaded
copyViaDBLinkOra	Yes
skipAnalyze	Yes
pgdbschedule	Yes
defaultBatchUpdateMode	No
switchBatchUpdateMode	No
syncBatchSize	Yes
busBatchThresholdCount	Yes
bupBatchThresholdCount	Yes
bupBatchThresholdRepeatLimit	Yes
offlineSnapshot	Yes
uniquenessConflictDetection	Yes
txSetMaxSize	Yes
skipConflictDetection	Yes
enablePerformanceStats	Yes
syncLoadThreadLimit	Yes
targetDBQueryTimeout	Yes
ddlChangeTableLock	Yes
lobBatchSize	Yes
batchInitialSync	Yes
deadlockRetryCount	Yes
deadlockWaitTime	Yes
mtk.logging.file.size	Yes
mtk.logging.file.count	Yes
persistZeroTxRepEvent	Yes
oraJDBCCustomURL	Yes
walStreamQueueLimit	Yes
walTxSetCreationInterval	Yes
dataSyncThreadCount	Yes
jdbc.pool.validationQueryTimeout	Yes
historyCleanupDaysThreshold	Yes

Example

This example reloads the configuration file.

> **Note**
>
> When you execute the `reloadconf` command, if any configuration options were changed from their default values, the output includes the configuration option and its new value.

```
java -jar edb-repcli.jar -reloadconf -repsvrfile subsvr.prop
```

```
 Reloading Subscription Server configuration file...
Reloaded configuration options from ../etc/xdb_subserver.conf...
The conf option 'snapshotParallelTableLoaderLimit' set to '1'
The conf option 'skipCheckConst' set to 'false'
The conf option 'snapshotParallelLoadCount' set to '1'

Configuration was reloaded successfully.
```

12 Data Validator

The Data Validator is a utility that compares the rows of one or more tables in a schema of a database against the rows of the tables with the same names in a schema of another database. The Data Validator generates a summary of the comparison, noting the number of rows whose column values differ. A file containing detailed information regarding any differences is also generated.

The two databases being compared are referred to as the source database and the target database. The source database can be of type Oracle, EnterpriseDB, SQL Server, Sybase, or MySQL. The target database must be either Oracle or EnterpriseDB.

An EnterpriseDB database type means either an EDB Postgres Advanced Server database or a PostgreSQL database.

The tables available for comparison are those found in the schema of the source database. Tables in the target database that don't exist in the source database schema are ignored.

> **Note**
>
> The Data Validator doesn't validate columns having the following data types. Tables containing one or more columns of these types are only partially validated.

- BFILE
- STRUCT
- REF
- ARRAY
- BLOB
- CLOB
- RAW
- LONG RAW

> **Note**
>
> Regarding the use of the Data Validator with tables in an Replication Server single-master or multi-master replication system, be sure all synchronization replication between the source and target Replication Server tables is complete before using the Data Validator. If synchronization replication is still in progress, the Data Validator will likely report differences in table content.

12.1 Installing and configuring the data validator

When you install the Replication Server product, the components for the Data Validator are installed as well. Also, when you uninstall the Replication Server product, the Data Validator components are uninstalled.

The following components that you use to run the Data Validator are installed when you install the Replication Server product.

File name	Location	Description
`datavalidator.properties`	*XDB_HOME*/etc	Data Validator Properties file

File name	Location	Description
`runValidation.sh (Linux)`	*XDB_HOME*/bin	Data Validator execution script
`runValidation.bat (Windows)`	*XDB_HOME*\bin	Data Validator execution script

> **Note**
>
> `XDB_HOME` is the directory where Replication Server is installed. This might not be the same as the Postgres home directory depending on how Replication Server is installed.

1. If you plan to use an Oracle database as the source or target database, download the Oracle JDBC driver and place it in the `JAVA_HOME/jre/lib/ext` directory.

2. Edit the `datavalidator.properties` file located in the `XDB_HOME/etc` directory and specify the connection information for the source and target databases you want to compare.

   You can override any of these parameters with an option when you invoke the Data Validator script. See Performing data validation for information on invoking the Data Validator.

   The following are the parameters in the `datavalidator.properties file`.

Parameter	Descripton
`source_dbms`	Type of the source database. Values can be enterprisedb, oracle, sqlserver, sybase, or mysql.
`source_host`	IP address or server name of the host running the database server of the source database
`source_port`	Port number on which the database server of the source database listens for requests
`source_database`	Database name of the source database
`source_user`	Database user name of the source database
`source_password`	Unencrypted password of the source database user
`target_dbms`	Type of the target database. Values can be enterprisedb or oracle.
`target_host`	IP address or server name of the host running the database server of the target database
`target_port`	Port number on which the database server of the target database listens for requests
`target_database`	Database name of the target database
`target_user`	Database user name of the target database
`target_password`	Unencrypted password of the target database user

   The following is the initial content of the `datavalidator.properties` file after installation:

```
 ################################################################
        Source database connection
################################################################

#source_dbms=(enterprisedb | oracle | sqlserver | sybase | mysql)

source_dbms=oracle
source_host=localhost
source_port=1521
source_database=xe
source_user=hr
source_password=hr
```

```
    #source_dbms=mysql
    #source_host=localhost
    #source_port=3306
    #source_database=test
    #source_user=root
    #source_password=

    #source_dbms=sqlserver
    #source_host=localhost
    #source_port=1433
    #source_database=pubs
    #source_user=sa
    #source_password=

    #source_dbms=sybase
    #source_host=localhost
    #source_port=5004
    #source_database=test
    #source_user=sa
    #source_password=

    ###############################################################
            Target database connection
    ###############################################################

    #target_dbms=(enterprisedb | oracle)

    target_dbms=enterprisedb
    target_host=localhost
    target_port=5444
    target_database=edb
    target_user=enterprisedb
    target_password=edb
```

3. Before invoking the Data Validator for the first time, determine the location of the Data Validator `logs` directory.

   The Data Validator generates a log file with a name formatted as `datavalidator_yymmdd-hhmiss.log` in the logs directory for each run.

   If there are row differences between the source and target tables, a file with a name formatted as `datavalidator_yymmdd-hhmiss.diff` is also generated that contains output of the errors in diff format. Use a graphical diff tool like Kompare to view this file to highlight the specific differences.

   The Data Validator attempts to create a subdirectory named `logs` in the `XDB_HOME/bin` directory the first time you invoke the Data Validator without the `-ld` option. If you don't invoke the Data Validator as the root account, the run will likely fail as it attempts to create subdirectory `logs` in the `XDB_HOME/bin` directory. Typically only the root account has this privilege.

   Windows hosts have the same situation, as the account you're using must have the permission to create a subdirectory in the `XDB_HOME\bin` location. Choices for determining and setting the Data Validator directory for the log and diff files are the following:

- Run the Data Validator as the root account. This approach enables the Data Validator to create the `logs` subdirectory in the `XDB_HOME/bin` directory and then to create the log and diff files in the `logs` subdirectory.

- Create the `XDB_HOME/bin/logs` directory structure before running the Data Validator. Modify the permissions on directory `XDB_HOME/bin/logs` so the operating system account you use to run the Data Validator has the privilege to create files in the directory.

- Use the `-ld log_directory_path` option to allow the Data Validator to create the log and diff files in the specified directory location `log_directory_path`. Be sure the operating system account you use to run the Data Validator has the privileges to either create the lowest level subdirectory specified by `log_directory_path` if it doesn't already exist or to create files in the specified directory if the full directory path already does exist.

Once you have determined and verified that the operating system account you plan to use to run the Data Validator can create files in the log directory, you can proceed with performing data validation.

12.2 Performing data validation

The current working directory from which you invoke the Data Validator script `runValidation.sh` (`runValidation.bat` for Windows) must be the `bin` subdirectory containing the script (that is, `XDB_HOME/bin`).

For example, if the Replication Server is installed in its default directory location, then issue the following command before invoking the Data Validator:

```
cd /usr/edb/xdb/bin
```

Similarly for Windows hosts, issue the following:

```
cd C:\Program Files\edb\EnterpriseDB-xDBReplicationServer\bin
```

The general command format for invoking the Data Validator is the following:

```
./runValidation.sh { -ss | --source-schema } schema_name
  [ option ] ...
```

`schema_name` is the name of the schema in the source database containing the tables to validate.

For Windows hosts, the command format is the following:

```
runValidation { -ss | --source-schema } schema_name
  [ option ] ...
```

The following option displays the Data Validator version:

```
./runValidation.sh { -v | --version }
```

On Linux the version is displayed as follows:

```
$ ./runValidation.sh --version
EnterpriseDB DataValidator Build 3
```

On Windows the version is displayed as follows:

```
C:\Program Files\edb\EnterpriseDB-xDBReplicationServer\bin>runValidation -v
EnterpriseDB DataValidator Build 3
```

The following option displays the help information.

```
./runValidation.sh { -h | --help }
```

For example,

```
$ ./runValidation.sh --help
Usage:
```

```
   runValidation.sh (-v  | --version) | (-h | --help)
   runValidation.sh (-ss | --source-schema) SOURCE_SCHEMA [OPTIONS] CONNECTION_INFO_FILE

OPTIONS:

   (-ts | --target-schema)   target-schema-name
   (-it | --include-tables)  comma-seperated-tables-name
   (-et | --exclude-tables)  comma-seperated-tables-name

   (-ld | --logging-dir)     logging-dir-path
   (-ds  | --display-summary) (true|false)
   (-srs | --skip-rowsonlyin-source) (true|false)
   (-srt | --skip-rowsonlyin-target) (true|false)
   (-srb | --skip-rowsin-both) (true|false)
   (-fs  | --fetch-size) row count
   (-bs  | --batch-size) row count

   (-sdbms | --source-dbms)     source database type
   (-sh | --source-host) source database server name/IP
   (-sp | --source-port) source database server port
   (-sdb | --source-database)     source database name
   (-su | --source-user) source database user id
   (-spw | --source-password) source database user password
   (-tdbms | --target-dbms)     target database type
   (-th | --target-host) target database server name/IP
   (-tp | --target-port) target database server port
   (-tdb | --target-database)     target database name
   (-tu | --target-user) target database user id
   (-tpw | --target-password) target database user password
   (-uoc | --use-ora-case) use oracle (upper) case for table name
```

> **New enhancement**
>
> The --use-ora-case option is available in Replication Server 7.1 and later.

> **Note**
>
> Data Validator does not support column mapping. With custom column mapping, table columns in the source and target databases can have different data types. As a result, the Data Validator is not able to validate the data when column mapping is applied.

The general syntax for all options except for `--version` and `--help` is shown by the following:

```
./runValidation.sh -ss schema
  [ -ts schema ]
  [ -it table_1 [,table_2 ] ... ]
  [ -et table_1 [,table_2 ] ... ]
  [ -srs { true | false } ]
  [ -srt { true | false } ]
  [ -srb { true | false } ]
  [ -ld log_directory_path ]
  [ -ds { true | false } ]
  [ -sdbms database_type ]
  [ -sh host ]
  [ -sp port ]
  [ -sdb dbname ]
  [ -su user ]
  [ -spw password ]
  [ -tdbms database_type ]
  [ -th host ]
```

```
  [ -tp port ]
  [ -tdb dbname ]
  [ -tu user ]
  [ -tpw password ]
  [ -bs row_count ]
  [ -fs row_count ]
  [ -uoc ]
```

For clarity, the syntax diagram shows only the single-character form of the option. The description of each option lists both the single-character and multi-character forms.

Specifying any database connection option (`-sdbms` through `-tpw` listed in the syntax diagram) overrides the corresponding parameter in the `datavalidator.properties` file. See Installing and configuring the data validator for information on the `datavalidator.properties` file.

Options

`-ss, --source-schema schema`

The schema of the source database containing the tables to compare against the target database.

`-ts, --target-schema schema`

The schema of the target database containing the tables to compare against the source database. If omitted, the schema of the target database is the same schema as specified for the source database with the `-ss` option.

`-it, --include-tables table_1 [,table_2 ] ...`

The tables in the source schema to include for comparison. If omitted, all tables in the source schema are compared against tables in the target schema. The exception is the tables excluded from comparison using the `-et` option. Don't use white space between the comma and table names.

`-et, --exclude-tables table_1 [,table_2 ] ...`

The tables in the source schema to exclude from comparison. If omitted, only those tables specified with the `-it` option are included for comparison. If both the `-it` and `-et` options are omitted, all source schema tables are included for comparison. Don't use white space between the comma and table names.

`-srs, --skip-rowsonlyin-source { true | false }`

When you specify `true` , the logging of differences for rows that exist only in the source database table are skipped. The default is `false` .

`-srt, --skip-rowsonlyin-target { true | false }`

When you specify `true` , the logging of differences for rows that exist only in the target database table are skipped. The default is `false` .

`-srb, --skip-rowsin-both { true | false }`

When you specify `true` , the logging of differences for rows:

- That exist both in the source and target database tables
- Have the same primary key
- Have different non-primary key values

are skipped. The default is `false` .

`-ld, --logging-dir log_directory_path`

Directory path to create and store the Data Validator log and diff files. If `log_directory_path` doesn't exist, Data Validator attempts to create it. If a full directory path isn't specified `log_directory_path` is created or assumed to be located relative to the `XDB_HOME/bin` subdirectory where the `runValidation.sh` script is invoked. (That is, the `logs` directory is `XDB_HOME/bin/log_directory_path`.) Be sure the operating system account used to invoke the `runValidation.sh` script has the privileges to create the directory if needed or to create files in the specified directory. The default is the `XDB_HOME/bin/logs` directory.

`-ds, --display-summary { true | false }`

Specify `true` to display only the Data Validator summary. This value omits the source and target database connection information as well as the detailed breakdown of the results by source database table. Specify `false` to display all of the Data Validator results. The type and amount of information that is displayed at the command line console when the Data Validator is invoked is the same information that is also stored in the log file for that run. The default is `false`.

`-sdbms, --source-dbms database_type`

The type of the source database server. Supported types are `oracle`, `enterprisedb`, `sqlserver`, `sybase`, and `mysql`.

`-sh, --source-host host`

The IP address or server name of the host where the source database server is running.

`-sp, --source-port port`

The port number on which the source database server is listening for connections.

`-sdb, --source-database dbname`

The name of the source database.

`-su, --source-user user`

The database user name for connecting to the source database.

`-spw, --source-password password`

The password of the source database user in unencrypted form.

`-tdbms, --target-dbms database_type`

The type of the target database server. Supported types are `enterprisedb` and `oracle`.

`-th, --target-host host`

The IP address or server name of the host whereh the target database server is running.

`-tp, --target-port port`

The port number on which the target database server is listening for connections.

`-tdb, --target-database dbname`

The name of the target database.

`-tu, --target-user user`

The database user name for connecting to the target database.

`-tpw, --target-password password`

The password of the target database user in unencrypted form.

`-bs, --batch-size row_count`

The `-bs` option specifies the number of rows to group in a batch to use for comparison across the source and target database tables. For example, if a table contains 1000 rows, then a `-bs` setting of 100 requires 10 batch iterations to complete the comparison across the source and target databases. The Data Validator reads 100 rows, both from the source and target tables, and adds them in source and target buffers. The validation thread then reads the 100 rows from the source and target buffers and performs the comparison. It then moves to read and prepare the next 100 rows for comparison, and so on. The actual database round trips required to bring in 100 rows from the database depends on the `-fs` option for the fetch size. For example, an `-fs` setting of 100 needs just one round trip and an `-fs` setting of 10 requires 10 database round trips.

`-fs, --fetch-size row_count`

Performing data validation for tables that are quite large can cause the Data Validator to stop with an out-of-heap-space error when using the default fetch size of 5000 rows. Use the `-fs` option to specify a smaller fetch size to help avoid the out-of-heap-space issue. The result set iteration brings in as many rows as represented by the `row_count` value in a single database round trip.

Examples

The following examples use an Oracle source database and an EDB Postgres Advanced Server target database to compare the tables in schema `EDB` on Oracle against the tables in schema `public` in EDB Postgres Advanced Server.

The following lists the tables in schema EDB along with the content of tables `DEPT` and `EMP` in the Oracle source database:

```
SQL> SELECT table_name FROM user_tables;
```

```
TABLE_NAME
------------------------------
ORATAB
DEPT
EMP
JOBHIST
```

```
SQL> SELECT * FROM dept;
```

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 FINANCE        CHICAGO
```

```
SQL> SELECT * FROM emp;
```

```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
      9001 SMITH      ANALYST         7566                 8500                    20
      9002 ROGERS     SALESMAN        7698                 8000       4000         30

16 rows selected.
```

The following lists the tables in schema public along with the content of tables dept and emp in the EDB Postgres Advanced Server `edb` database:

```
edb=#
\dt
           List of relations
```

```
 Schema |  Name   | Type  |    Owner
--------+---------+-------+--------------
 public | dept    | table | enterprisedb
 public | emp     | table | enterprisedb
 public | jobhist | table | enterprisedb
(3 rows)
```

```
edb=# SELECT * FROM dept;
```

```
 deptno |   dname    |   loc
--------+------------+----------
     10 | ACCOUNTING | NEW YORK
     20 | RESEARCH   | DALLAS
     30 | SALES      | CHICAGO
     40 | OPERATIONS | BOSTON
(4 rows)
```

```
edb=# SELECT * FROM emp;
```

```
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm   | deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 17-DEC-80 00:00:00  |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 00:00:00  | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN  | 7698 | 22-FEB-81 00:00:00  | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER   | 7839 | 02-APR-81 00:00:00  | 2975.00 |         |     20
  7654 | MARTIN | SALESMAN  | 7698 | 28-SEP-81 00:00:00  | 1250.00 | 1400.00 |     30
  7698 | BLAKE  | MANAGER   | 7839 | 01-MAY-81 00:00:00  | 2850.00 |         |     30
  7782 | CLARK  | MANAGER   | 7839 | 09-JUN-81 00:00:00  | 2450.00 |         |     10
  7788 | SCOTT  | ANALYST   | 7566 | 19-APR-87 00:00:00  | 3000.00 |         |     20
  7839 | KING   | PRESIDENT |      | 17-NOV-81 00:00:00  | 5000.00 |         |     10
  7844 | TURNER | SALESMAN  | 7698 | 08-SEP-81 00:00:00  | 1500.00 |    0.00 |     30
  7876 | ADAMS  | CLERK     | 7788 | 23-MAY-87 00:00:00  | 1100.00 |         |     20
```

```
  7900 | JAMES  | CLERK     | 7698 | 03-DEC-81 00:00:00 |  950.00 |         |     30
  7902 | FORD   | ANALYST   | 7566 | 03-DEC-81 00:00:00 | 3000.00 |         |     20
  7934 | MILLER | CLERK     | 7782 | 23-JAN-82 00:00:00 | 1300.00 |         |     10
  9001 | SMITH  | SALESMAN  | 7698 |                    | 8000.00 | 4000.00 |     30
  9002 | ROGERS | SALESMAN  | 7698 |                    | 9500.00 | 4000.00 |     30
(16 rows)
```

Note the following differences:

- The Oracle `EDB` schema contains one moew table named `ORATAB` that doesn't exist in the EDB Postgres Advanced Server public schema.
- The Oracle DEPT table contains one extra row with `DEPTNO 50` that doesn't exist in the EDB Postgres Advanced Server dept table.
- The rows in the EMP table with `EMPNO` values `9001` and `9002` have column values that differ between the Oracle and EDB Postgres Advanced Server tables.
- In this example, the JOBHIST table contains identical rows for both the Oracle and Advanced Server tables.

The content of the `datavalidator.properties` file is set as follows:

```
################################################################
            Source database connection
################################################################

#source_dbms=(enterprisedb | oracle | sqlserver | sybase | mysql)

source_dbms=oracle
source_host=192.168.2.23
source_port=1521
source_database=xe
source_user=edb
source_password=password

################################################################
         Target database connection
################################################################

#target_dbms=(enterprisedb | oracle)

target_dbms=enterprisedb
target_host=localhost
target_port=5444
target_database=edb
target_user=enterprisedb
target_password=password
```

The following example compares all tables in the Oracle `EDB` schema against the EDB Postgres Advanced Server public schema.

The Data Validator log files are created in directory `/home/user/datavalidator_logs` as specified with the `-ld` option. The operating system account used to invoke the `runValidation.sh` script has write access to the `/home/user` directory so the Data Validator can create the `datavalidator_logs` subdirectory.

```
 $ cd /usr/edb/xdb/bin
$ pwd
/usr/edb/xdb/bin
$ ./runValidation.sh -ss edb -ts public -ld /home/user/datavalidator_logs

EnterpriseDB DataValidator Build 3
-------------------------------------------------------------------------------
Source and target databases connection information
-------------------------------------------------------------------------------
```

```
Source database:

     DBMS:     ORACLE
     Host:     192.168.2.23
     Port:     1521
     Database: xe
     User:     edb

Target database:

     DBMS:     ENTERPRISEDB
     Host:     localhost
     Port:     5444
     Database: edb
     User:     enterprisedb

------------------------------------------------------------------------------
Databases data validation process started...
------------------------------------------------------------------------------

Validating Table DEPT
  Rows validated: 5
  Finished validating table DEPT with 1 errors.
  Logging errors details in the diff file...

Validating Table EMP
  Rows validated: 16
  Finished validating table EMP with 2 errors.
  Logging errors details in the diff file...

Validating Table JOBHIST
  Rows validated: 17
  Finished validating table JOBHIST with 0 errors.

Validating Table ORATAB
  Table not validated as it does not exist on the target database.


DataValidator found 3 errors across source and target databases.
For detailed error report see datavalidator_20150713-144417.diff file.

------------------------------------------------------------------------------
Data validation process has completed.
------------------------------------------------------------------------------

*************************************************************
                          DataValidator Summary
*************************************************************

       All tables count: 4

       Validated tables count: 3
       Rows count: 38
       Errors count: 3

       Missing tables on the target database count: 1
       Tables list:
           - EDB.ORATAB

       Tables having only unsupported datatypes count: 0
```

```
       Tables having primary key limitation count: 0

       Total time(s): 0.678
       Rows per second: 56

**************************************************************
```

The Data Validator output indicates the following:

- There is one error in the DEPT table (the missing row).
- There are two errors in the EMP table (the two rows with mismatching column values).
- The JOBHIST table contains no errors.
- The ORATAB table doesn't exist on the target database.

The following shows the files created in the Data Validator `logs` directory:

```
 $ pwd
/home/user/datavalidator_logs
$ ls -l
total 24
-rw-rw-r-- 1 user user 18999 Aug 13 15:44 datavalidator_20150713-144417.diff
-rw-rw-r-- 1 user user  2133 Aug 13 15:44 datavalidator_20150713-144417.log
```

The log file contains the same content as displayed when the Data Validator is invoked. The diff file compares the differences where errors were detected.

The following example includes only tables dept and emp with the `-it` option when comparing the Oracle EDB schema against the Advanced Server public schema.

```
 $ cd /usr/edb/xdb/bin
$ pwd
/usr/edb/xdb/bin
$ ./runValidation.sh -ss edb -ts public -ld /home/user/datavalidator_logs -it dept,emp

EnterpriseDB DataValidator Build 3
--------------------------------------------------------------------------------
Source and target databases connection information
--------------------------------------------------------------------------------

Source database:

     DBMS:     ORACLE
     Host:     192.168.2.23
     Port:     1521
     Database: xe
     User:     edb

Target database:

     DBMS:     ENTERPRISEDB
     Host:     localhost
     Port:     5444
     Database: edb
     User:     enterprisedb

--------------------------------------------------------------------------------
Databases data validation process started...
--------------------------------------------------------------------------------
```

```
Validating Table DEPT
  Rows validated: 5
  Finished validating table DEPT with 1 errors.
  Logging errors details in the diff file...

Validating Table EMP
  Rows validated: 16
  Finished validating table EMP with 2 errors.
  Logging errors details in the diff file...


DataValidator found 3 errors across source and target databases.
For detailed error report see ``datavalidator_20150714-123353.diff`` file.

------------------------------------------------------------------------------
Data validation process has completed.
------------------------------------------------------------------------------

**************************************************************
                           DataValidator Summary
**************************************************************

       All tables count: 2

       Validated tables count: 2
       Rows count: 21
       Errors count: 3

       Missing tables on the target database count: 0

       Tables having only unsupported datatypes count: 0

       Tables having primary key limitation count: 0

       Total time(s): 0.539
       Rows per second: 39

**************************************************************
```

The following example excludes tables ORATAB and jobhist with the `-et` option when comparing the Oracle EDB schema against the EDB Postgres Advanced Server public schema. The `-ds true` option results in the display of only the Data Validator summary.

```
 $ ./runValidation.sh -ss edb -ts public -ld /home/user/datavalidator_logs -et ORATAB,jobhist -ds true
Databases data validation process started...

**************************************************************
                           DataValidator Summary
**************************************************************

       All tables count: 2

       Validated tables count: 2
       Rows count: 21
       Errors count: 3

       Missing tables on the target database count: 0

       Tables having only unsupported datatypes count: 0

       Tables having primary key limitation count: 0
```

```
      Total time(s): 0.535
      Rows per second: 39

************************************************************
```

For this run, the corresponding log file contains only the Data Validator summary, omitting the source and target database connection information along with the error breakdown by table.

13 Appendix

This appendix covers miscellaneous topics.

13.1 Resolving problems

Try these tips for locating and correcting various problems that can occur.

13.1.1 Error messages

These error messages can appear from the Replication Server console. The messages are listed in alphabetical order.

This list includes only the messages that typically involve initial configuration operations requiring additional information for resolving the problem.

Error Messages and Resolutions

Problem

```
Authentication failed. Reason: Invalid user name/password.
```

Resolution

Occurs when registering a publication server or subscription server. Verify the user name and password you enter matches the admin user name and password in the Replication Server configuration file on the host you're running the publication server or subscription server. See Replication Server configuration file.

Problem

```
Cannot register database because it is already registered by a publication service.
```

Resolution

You can create only one publication database definition for any given database. (Oracle is the exception whereby more than one publication database definition can be created for the same Oracle database if different Oracle user names are specified in each publication database definition.)

Problem

```
The connection could not be established with the server. Verify that the server is running and accepting connections. Reason: Connection refused to host: *xxx*.\ *xxx*.\ *xx*.\ *xxx*; nested exception is: java.net.ConnectException: Connection refused
```

Resolution

Occurs whenever a Java RMI connection can't be made to the publication server, the subscription server, or a database server. Can occur when registering a publication or subscription server, adding a publication database or a subscription database, or identifying the publication server for a new subscription. Verify you that have entered the correct host IP address and port number of the server. Verify the server is running (see Starting the publication server or subscription server). If the server is running on Linux, verify that in the `/etc/hosts` file the host name is mapped to the correct network IP address, which matches the IP address returned by the Linux `/sbin/ifconfig` command and also matches the IP address you entered in the **Host** field of the dialog box. Alternatively, instead of modifying the `/etc/hosts` file, set configuration option `java.rmi.server.hostname` to the IP address of the publication or subscription server (see Assigning an IP address for remote method invocation Don't use the loopback address `127.\ *x*.\ *x*.\ *x*` for this entry.

Problem

```
Connection refused. Check that the hostname and port are correct and that the postprimary is accepting TCP/IP connections.
```

Resolution

Occurs when attempting to save a publication database definition. The publication server can't connect to the database server network location given in the Add Database dialog box. Verify that the correct IP address and port for the database server are given. Verify that the database server is running and is accessible from the host running the publication server.

Problem

```
Could not connect to the database server. Reason: FATAL: number of requested standby connections exceeds max_wal_senders (currently *n*)
```

Resolution

Occurs when attempting a snapshot replication from a publication database configured with the log-based method of synchronization replication (that is, WAL-based logical replication), and the additional concurrent connection for logical replication exceeds the current setting, `n`, of the `max_wal_senders` configuration parameter in the `postgresql.conf file`. Increase the value of `max_wal_senders` in the `postgresql.conf` file of the database server running the publication database. Restart the database server containing the publication database. See Synchronization Replication with the Log-Based Method.

Problem

```
Currently no publication exists on the publication server. Please create at least one publication on the server and then retry.
```

Resolution

Occurs when attempting to create a subscription. If there are no publications in the specified publication server, then this error message appears.

Problem

```
The database cannot be registered because a partial schema already exists. A manual cleanup is required to proceed. For help with manual cleanup please check out our product documentation.
```

Resolution

The metadata database objects from a prior publication already exist in the schema under which the publication server is attempting to create new metadata database objects. Perform the operation described in Deleting the control schema and control schema objects.

Problem

```
Database cannot be removed. Reason: Publication database connection cannot be removed as one or more publications are defined against it.
```

Resolution

Make sure all publications subordinate to the publication database definition are removed. If no publications appear under the Publication Database node in the Replication Server console replication tree and the error persists, there might be a problem with the control schema objects. Perform the operation described in Deleting the control schema and control schema objects.

Problem

```
Database cannot be removed. Reason: Publication service failed to clean up replication control schema tables.
```

Resolution

The control schema objects under the Oracle publication database user schema or under the Postgres or SQL Server schemas `\_edb_replicator_pub`, `\_edb_replicator_sub`, or `\_edb_scheduler` can't be deleted by the publication server. The control schema objects or schemas might have already been deleted. The publication database definition can't be removed using the Replication Server console. Perform the operation described in Deleting the control schema and control schema objects.

Problem

```
Database cannot be removed. Reason: The target publication database is currently set as the Controller database and is being referenced by one or more dependent nodes.
```

Resolution

Occurs when attempting to remove the publication database currently set as the controller database. Select another publication database to use as the controller database. Use the **Set As Controller** option in the publication database's context menu to set this database as the controller database. You can then remove the original publication database. See Switching the controller database.

Problem

```
Database cannot be set as controller. Reason: Connection refused. Check that the hostname and port are correct and that the postprimary is accepting TCP/IP connections.
```

Resolution

Occurs when attempting to set a publication database as the controller database and the database isn't accessible by the publication server. Verify that the correct IP address and port was defined in the publication database definition. Verify that the database server is running and is accessible from the host running the publication server.

Problem

```
Database connection cannot be added. Connection refused. Check that the hostname and port are correct and that the postprimary is accepting TCP/IP connections.
```

Resolution

Occurs when attempting to save a subscription database definition. The subscription server can't connect to the database server network location given in the Add Database dialog box. Verify that the correct IP address and port for the database server are given. Verify that the database server is running and is accessible from the host running the subscription server.

Problem

`Database connection cannot be added. FATAL: no pg_hba.conf entry for host "*xxx*.\ *xxx*.\ *xx*.\ *xxx*", user "*user_name*", database "*db_name*", SSL off`

Resolution

Occurs when attempting to save a subscription database definition. The subscription server isn't permitted to connect to the database at the network location given in the Add Database dialog box. Verify that the database host IP address, port number, database user name, password, and database identifier are correct. Verify there is an entry in the `pg_hba.conf` file permitting access to the database by the given user name originating from the IP address where the subscription server is running.

Problem

`Database connection cannot be added. Controller database is not initialized yet.`

Resolution

Occurs when attempting to add a subscription database. Verify that the Replication Server configuration file on the host running the subscription server contains an entry for a valid controller database. Verify that a publication database was defined under the publication server as the controller database and its connection information is recorded in the Replication Server configuration file. See Replication Server configuration file.

Problem

`The database type for the selected database is different than that of the MDN database. Each database should be of the same type in a MMR cluster.`

Resolution

All database servers in a multi-master replication system must be of the same type—either all PostgreSQL (or EDB Postgres Advanced Server installed in PostgreSQL-compatible configuration mode) or all EDB Postgres Advanced Server installed in Oracle-compatible configuration mode. This error message appears when attempting to add a primary node and the database server type differs from the database server type of the primary definition node. See Permitted MMR database server configurations.

Problem

`An error occurred while removing tables from other Primary node(s). Please refer to the user manual for instructions on how to remove shadow tables and triggers from Primary node(s).`

Resolution

When a primary node of a multi-master replication system is deleted using the Replication Server console or the Replication Server CLI, the control schema objects that were created in the primary node are also dropped. These include schemas `\_edb_replicator_pub`, `\_edb_replicator_sub`, and `\_edb_scheduler`. For the log-based method of synchronization replication, there are shadow tables and triggers on the publication tables as well. If any of these control schema objects fail to be dropped, this error message appears. See Dropping replication slots for log-based synchronization replication for directions on how to remove these control schema objects.

Problem

`FATAL: no pg_hba.conf entry for host "*xxx*.\ *xxx*.\ *xx*.\ *xxx*", user "*user_name*", database "*db_name*", SSL off`

Resolution

Occurs when attempting to save a publication database definition. The publication server isn't permitted to connect to the database at the network location given in the Add Database dialog box. Verify that the database host IP address, port number, database user name, password, and database identifier are correct. Verify there is an entry in the `pg_hba.conf` file permitting access to the database by the given user name originating from the IP address where the publication server is running.

Problem

```
Filter cannot be defined for Binary data type column(s)e.g. BYTEA, BLOB, RAW.
```

Resolution

Occurs when attempting to define a filter rule on a column with a binary data type in a publication table. Filter rules aren't permitted on such columns. See Table settings and restrictions for table filters.

Problem

```
Filter with same name/clause already exist on table/view: *schema*.\ *table_name*
```

Resolution

When adding a filter rule on a publication table, you can't use the same filter name or the same filter clause (WHERE clause) more than once on a given table. Modify the duplicate filter name or filter clause so it is unique for the table.

Problem

```
The initial snapshot is not performed for this subscription. Please take the snapshot first and then
proceed with the synchronize operation.
```

Resolution

A snapshot replication must be performed before the first synchronization replication. Perform an on-demand snapshot replication.

Problem

```
It is recommended to use a network IP address, the loopback address may result in connectivity issues.
```

Resolution

This warning is given when localhost or 127.0.0.1 is specified as the host address of a replication system component. We strongly recommend that all replication system components are identified by their specific IP address on the network.

Problem

```
The log triggers creation failed for one or more publication tables. Make sure the database is in valid
state and user is granted the required privileges.
```

Resolution

Either the user doesn't have the trigger creation privilege or there is a database server problem. The database server message is displayed as part of the error.

Problem

```
The MMR mode is currently not supported for *database_type* database.
```

Resolution

A database server of type *database_type* can't be used in a multi-master replication system. You can use only EDB Postgres Advanced Server or PostgreSQL database servers as primary nodes in a multi-master replication system.

Problem

```
Multiple filters of same table are not allowed.
```

Resolution

When creating a subscription in a single-master replication system or creating a primary node other than the primary definition node in a multi-master replication system, you can select only one filter for a given table. Clear the additional boxes in the Apply column under the **Filter Rules** tab if more than one box is selected.

Problem

```
No JDBC Client driver is configured for the Oracle data source.
```

Resolution

Occurs when creating an Oracle publication or subscription database definition. Copy the Oracle JDBC driver file `ojdbc\ *x*.jar` to subdirectory `lib/jdbc` of where the publication server or subscription server is installed on the host running the publication server or subscription server. Restart the publication server or subscription server.

Problem

```
No Publication found on MDN node, additional Primary node cannot join MMR cluster.
```

Resolution

Occurs when attempting to add a second primary node to a multi-master replication system but no publication was defined under the primary definition node. Create a publication under the primary definition node, and then add the additional primary nodes. See Adding a publication.

Problem

```
None of the target master/subscription databases is accessible, hence the replication process failed to complete.
```

Resolution

Synchronization replication failed due to the unavailability of a target database. See the publication server log file for details. See Where to look for errors.

Problem

```
One or more primary database node(s) are defined against this publication. Removing the publication will invalidate the MDN.
```

Resolution

Primary nodes are still defined in a multi-master replication system in which an attempt is being made to delete the publication from the primary definition node. You must delete all primary nodes (other than the primary definition node) first before deleting the publication from the primary definition node. Perform this deletion process with the Replication Server console or Replication Server CLI.

Problem

```
One or more subscriptions are defined against this publication. Removing the publication will invalidate the subscription. Do you want to continue?
```

Resolution

Warning issued when you attempt to remove a publication with subscriptions associated with it. You can remove the publication, but the subscriptions are no longer usable and you must remove them as well.

Problem

```
Multiple Publications creation is currently not supported.
```

Resolution

Only one publication is supported in a multi-master replication system, and only one such multi-master replication system can exist for an Replication Server installation.

Problem

```
Only subscription which has subscribed against a publication with transactional replication type, can be synchronized.
```

Resolution

You can't perform synchronization replication on a snapshot-only publication. Perform snapshot replication instead.

Problem

```
The Oracle/MS SQL Server cannot be registered if the active Controller database is a non-PG/PPAS database.
```

Resolution

Occurs when creating an Oracle or SQL Server publication database definition and the current controller database is not a Postgres database (that is, the controller database is an Oracle or SQL Server database). /To create an Oracle or SQL Server publication database, create and designate a Postgres publication database as the controller database. See Switching the controller database.

Problem

```
Parent table *table_name* is not selected when its child tables are part of the publication list.
```

Resolution

Table selected for a publication has a foreign key referencing a parent table that hasn't been chosen for the publication. This is only a warning that the parent table will not be part of the subscription.

Problem

```
Problem occurred in publish process. Reason: Connection refused. Check that the hostname and port are correct and that the postprimary is accepting TCP/IP connections.
```

Resolution

Occurs when attempting synchronization replication and the controller database isn't accessible by the publication server. Verify that the correct IP address and port was defined in the publication database definition of the controller database. Verify that the database server is running and is accessible from the host running the publication server.

Problem

```
Problem occurred in publish process. Reason: ERROR: permission denied for schema \_edb_replicator_pub
```

Resolution

For a Postgres publication, verify that the publication database user has `CREATE ON DATABASE` privilege on the publication database, or the database user is a superuser.

Problem

```
Publication cannot be created. One or more tables have no attributes defined and cannot be published.
Unselect the specific tables and retry.
```

Resolution

In Postgres, it is possible to create a table with no columns. A publication can't include a Postgres table with no columns since the corresponding subscription table can't be created in Oracle.

Problem

```
Publication cannot be created. Publication *publication_name* already exists on the publisher server.
Please choose a different name and then proceed.
```

Resolution

Publication names must be unique in a publication server. Enter a different publication name.

Problem

```
Publication cannot be created. Table *schema*.\ *table_name* replica identity is set to
*replica_identity_setting*. To define a Filter, the table replica identity should be set to FULL.
```

Resolution

Occurs when you attempt to define a table filter is attempted on a publication table used in a log-based replication system. Use the `ALTER TABLE` statement to change `REPLICA IDENTITY` to `FULL`. See Table settings and restrictions for table filters.

Problem

```
Publication cannot be created. Replica Identity for table table_name is DEFAULT, but it does not contain a
Primary Key or Unique columns. Transactional replication is not supported for this table.
```

Resolution

All tables used for synchronization replication must have primary keys or unique columns. Create a primary key or unique column on the table, or add the table to a snapshot-only publication.

Problem

```
Publication cannot be created. Replica Identity for table table_name is INDEX, but it does not have Unique
Columns. Transactional replication is not supported for this table.
```

Resolution

All tables used for synchronization replication must have primary keys or unique columns. Create a primary key or unique column on the table, or add the table to a snapshot-only publication.`

Problem

```
Publication cannot be created. Replica Identity for table table_name is FULL and it has no Primary Key and
no Unique Columns. Transactional replication is not supported for this table yet.
```

Resolution

All tables used for synchronization replication must have primary keys or unique columns. Create a primary key or unique column on the table, or add the table to a snapshot-only publication.

Problem

```
Publication cannot be created. Replica Identity for table table_name is NOTHING. Log-based transactional replication is not supported for this table.
```

Resolution

With replica identity NOTHING, no WAL records are produced, so log-based transactional replication isn't possible. Change the replica identity of this table and provide primary key and no unique columns.

Problem

```
Publication cannot be created. Replica Identity for table table_name is NOTHING and it does not contain a Primary Key or Unique columns. Tigger-based transactional replication is not supported for this table.
```

Resolution

All tables used for synchronization replication must have primary keys or unique columns. Create a primary key or unique column on the table, or add the table to a snapshot-only publication.

Problem

```
Publication cannot be created. The publication creation process timed out as one or more tables may be locked by another session. Please retry later.
```

Resolution

For a Postgres publication that isn't for snapshot-only, the publication database user must be able to create triggers on the publication tables. To do this, the publication database user must have the privilege to execute the `ALTER TABLE` statement on the publication tables, and the publication database user must have `CREATE` and `USAGE` privileges on the schema containing the publication tables. Verify that one of the following is true: 1) All the tables in the publication are owned by the publication database user and the user has `CREATE` and `USAGE` privileges on the publication tables' schemas, or 2) the publication database user is a superuser.

Problem

```
Publication cannot be removed. Reason: Publication *publication_name* cannot be removed. Reason: Error: cannot drop table \_edb_replicator_pub.rrst\_\ *schema_table_name* because other objects depend on it.
```

Resolution

PL/pgSQL custom conflict handler functions might exist in the primary definition node that depend on the publication's shadow tables. Drop the custom conflict handler functions before deleting the publication.

Problem

```
Publication cannot be updated. Reason: The parent table *schema*.\ *table_name* is selected for removal while it has one or more child tables in the publication list. Make sure that parent-child dependency holds in the publication tables.
```

Resolution

Choose the child tables for removal as well as the parent table.

Problem

```
Publication defined in MMR cluster cannot be subscribed in SMR cluster.
```

Resolution

A given publication can't be used in both a multi-master replication system and a single-master replication system.

Problem

```
Publication does not exist on the publication server. It might have been removed.
```

Resolution

The publication doesn't exist for a given subscription. The subscription is no longer usable and must be removed.

Problem

```
Publication having subscription against it, cannot be updated by removing tables from it.
```

Resolution

Remove the subscription, remove tables from the publication, and then add the subscription.

Problem

```
The publication schema cannot be created. Reason: ERROR: Permission denied for database *db_name*.
```

Resolution

Occurs when attempting to create the publication database definition and the specified publication database user doesn't have the privilege to create a schema in database `db_name`. Grant the `CREATE` privilege on the database to the publication database user.

Problem

```
Publication Service connection failure.
```

Resolution

Verify that the publication server is running. See Starting the publication server or subscription server. Verify that the database server hosting the controller database specified in the Replication Server configuration file is running and the publication server is connected to it. See Replication Server configuration File.

Problem

```
The replication process could not be completed due to a database failure. Check the database state and retry.
```

Resolution

Might be caused by characters in the publication data that are illegal for the character set of the subscription database. Check the snapshot replication failure log file or the database server log file. See Replacing null characters.

Problem

```
Replication server does not support Oracle to Oracle replication.
```

Resolution

See Permitted configurations and combinations for supported database server configurations. Use Oracle products for Oracle-to-Oracle replication.

Problem

```
A replication slot is not available on the target database server. Please configure the
max_replication_slots GUC on the database server.
```

Resolution

Occurs when attempting to add a publication database definition with the log-based method of synchronization replication, and the `max_replication_slots` configuration parameter in the `postgresql.conf` file isn't set to a large enough value to accommodate the added database. Increase the value of the `max_replication_slots` parameter and restart the database server. See Synchronization replication with the log-based method for more information.

Problem

```
Subscription *subscription_name* already exists on the subscriber server. Please choose a different name
and then proceed.
```

Resolution Subscription names must be unique in a subscription server. Enter a different subscription name.

Problem

```
Subscription *subscription_name* cannot be removed. Reason: Publication does not exist on the publication
server.
```

Resolution

Warning issued if the subscription you're attempting to remove doesn't have an associated publication. You can still remove the subscription. Subscription database connection can't be removed as one or more subscriptions are defined against it.

You can't remove a subscription database definition if there are subordinate subscriptions. Remove the subscriptions first.

Problem

```
Subscription does not exist on the subscription service. It might have been removed by some other user.
```

Resolution The Subscription node you are trying to select no longer represents an existing subscription. The subscription might have been removed by a concurrent Replication Server console or Replication Server CLI session. Select **Refresh** in the Replication Server console toolbar to display the current replication tree.

Problem

```
Subscription Service connection failure.
```

Resolution Verify that the subscription server is running. See Starting the publication server or subscription server

Problem

```
Synchronize Publication process failed for one or more primary nodes. Please see logs for more details.
```

Resolution

Synchronization replication failed to complete for all target databases in the multi-master replication system due to the unavailability of some target

database. See the publication server log file for details. See Where to look for errors. A table with large object type PK attribute can't be published for (synchronize) incremental replication.

Oracle doesn't log changes for a large object column. Such a column can't be referenced in the triggers that log changes to the shadow tables. Use snapshot-only replication instead.

Problem

```
Test result: Failure
```

```
Database connection information test failed. Connection refused. Check that the hostname and port are correct and that the postprimary is accepting TCP/IP connections.
```

Resolution

Occurs when testing the connection of a publication or subscription database definition. The publication or subscription server can't connect to the database server network location given in the Add Database dialog box. Verify that the correct IP address and port for the database server are given. Verify that the database server is running and is accessible from the host running the publication or subscription server.

Problem

```
Test result: Failure
```

```
Database connection information test failed. FATAL: no pg_hba.conf entry for host "*xxx*.\ *xxx*.\ *xx*.\ *xxx*", user "*user_name*", database "*db_name*", SSL off
```

Resolution

Occurs when testing the connection of a publication or subscription database definition. The publication or subscription server isn't permitted to connect to the database at the network location given in the Add Database dialog box. Verify that the database host IP address, port number, database user name, password, and database identifier are correct. Verify there is an entry in the `pg_hba.conf` file permitting access to the database by the given user name originating from the IP address where the publication or subscription server is running.

Problem

```
Test result: Failure
```

```
Database connection information test failed. IO exception: The Network Adapter could not establish the connection.
```

Resolution

Verify that the database server is running. For Oracle, verify that the Oracle listener program `lsnrctl` is running.

Problem

```
Test result: Failure
```

```
The target database server cannot be registered for WAL based logical replication. Reason: The database server is not configured for logical replication. Reason: FATAL: must be superuser or replication role to start walsender.
```

Resolution Occurs when attempting to add a publication database definition with the log-based method of synchronization replication (that is, WAL-based logical replication), and the publication database user isn't a superuser or doesn't have `REPLICATION` privilege. Grant the publication database user the appropriate privilege, or specify a different database user who has the appropriate privilege for logical replication as the publication database user. See Synchronization replication with the log-based method.

Problem

```
Test result: Failure
```

```
The target database server cannot be registered for WAL based logical replication. Reason: The database server is not configured for logical replication. Reason: FATAL: no pg_hba.conf entry for replication connection from host "*xxx*.\ *xxx*.\ *xx*.\ *xxx*", user "*user_name*", SSL off
```

Resolution

Occurs when attempting to add a publication database definition with the log-based method of synchronization replication (that is, WAL-based logical replication), and there's no entry in the `pg_hba.conf` file where the `DATABASE` field is set to replication for `user_name`. The `pg_hba.conf` file of the target database server must contain a replication entry for the publication database user name specified when creating the publication database definition. See Synchronization replication with the log-based method.

Problem

```
Test result: Failure
```

```
The target database server cannot be registered for WAL based logical replication. Reason: The database server is not configured for logical replication. Reason: FATAL: number of requested standby connections exceeds max_wal_senders (currently *n*)
```

Resolution

Occurs when attempting to add a publication database definition with the log-based method of synchronization replication (that is, WAL-based logical replication), and the additional concurrent connection for logical replication exceeds the current setting, `n`, of the `max_wal_senders` configuration parameter in the `postgresql.conf` file. Increase the value of `max_wal_senders` in the `postgresql.conf` file of the database server running the publication database. Restart the database server containing the publication database. See Synchronization replication with the log-based method.

Problem

```
Test result: Failure
```

```
The target database server cannot be registered for WAL based logical replication. Reason: The target database server version x.x does not support WAL logical decoding.
```

Resolution

Occurs when attempting to create a publication database definition with the log-based method of synchronization replication (that is, WAL-based logical replication), and the Postgres database server isn't version 9.4 or later. Only Postgres database servers of version 9.4 or later support the log-based method of synchronization replication. See Synchronization replication with the log-based method.

Problem

```
Unable to apply DDL changes.
```

Resolution The DDL statements in the text file specified for the DDL change replication feature contain syntax errors or aren't supported by the DDL change replication feature. See Replicating DDL changes.

Problem

```
Unable to communicate with remote server.
```

Resolution

Occurs when attempting an operation such as performing synchronization replication or creating a schedule on a publication or subscription database that can't be accessed by the Replication Server console. Verify that the publication and/or subscription servers are running. Verify that the database servers of the publication or subscription databases are running.

Problem

```
Unable to create schema tables in target primary database.
```

```
Unable to create publication shadow tables.
```

```
Unable to create subscription schema tables.
```

```
DB-42501: com.edb.util.PSQLException: ERROR: permission denied for relation pg_class.
```

Resolution Occurs when attempting to create an MMR publication database definition and the publication server can't create the control schema objects in the new publication database. This typically results when creating a second publication database definition and the publication server can't copy by snapshot the control schema objects from the controller database to the new publication database. The publication database user of the new publication database must be a superuser. In addition, in system catalog table `pg_catalog.pg_authid`, column `rolcatupdate` must be set to true for this superuser. See Disabling foreign key constraints for snapshot replications.

Problem

```
Unable to create Subscription subscription_name. Reason: Connection rejected: FATAL: no pg_hba.conf entry
for host "*xxx*.\ *xxx*.\ *xxx*.\ *xxx*" user "*user_name*", database "*db_name*", SSL off
```

Resolution

Occurs when creating a subscription. The subscription server running on host `xxx.xxx.xx.xxx` can't access the controller database. Verify that the `pg_hba.conf` file on the controller database server permits access from the subscription server host.

Problem

```
Unable to create subscription schema tables. Org.postgresql.util.PSQLException: FATAL: no pg_hba.conf entry
for host "*xxx*.\ *xxx*.\ *xxx*.\ *xxx*" user "*user_name*", database "*db_name*", SSL off
```

Resolution

Occurs when creating a subscription. The subscription server running on host `xxx.xxx.xx.xxx` can't access the publication database. Verify that the `pg_hba.conf` file on the publication database server permits access from the subscription server host.

Problem

```
Unable to create subscription schema tables. The database type is not supported.
```

Resolution

The subscription database type isn't supported for the intended publication database type. See Permitted SMR source and target configurations.

Problem

```
Unable to create subscription schema tables. The target database schema already contains one or more tables
with the same name as the table(s) in the source database.
```

Resolution

The subscription server can't create a subscription table definition in the intended target schema. Typically, the reason is that a table with the same name

already exists in the target schema of the subscription database. This can occur if you create a subscription, remove it but fail to drop the table definitions created under the target schema, and then try to create the subscription a second time.

Problem

```
Unable to create publication shadow tables.
```

```
Unable to create subscription schema tables.
```

```
DB-42501: com.edb.util.PSQLException: ERROR: permission denied for relation pg_class.
```

Resolution

Occurs when attempting to create an SMR publication database definition and the publication server can't create the control schema objects in the new publication database. This typically results when creating a second publication database definition and the publication server can't copy by snapshot the control schema objects from the controller database to the new publication database. The publication database user of the new publication database must be a superuser. In addition, in system catalog table `pg_catalog.pg_authid`, column `rolcatupdate` must be set to `true` for this superuser. See Disabling foreign key constraints for snapshot replications.

Problem

`Unable to perform snapshot for subscription` subscription_name `. Reason: DB-42501: com.edb.util.PSQLException: ERROR: permission denied for relation pg_class.`

Resolution

Occurs when attempting a snapshot replication. The database user of the database receiving the snapshot must be a superuser. In addition, in system catalog table `pg_catalog.pg_authid`, column `rolcatupdate` must be set to `true` for this superuser. See Disabling foreign key constraints for snapshot replications.

Problem

```
Unable to perform snapshot for subscription subscription_name. Reason: org.postgresql.util.PSQLException: FATAL: no pg_hba.conf entry for host "*xxx*.\ *xxx*.\ *xx*.\ *xxx*", user "*user_name*", database "*db_name*", SSL off
```

Resolution

Occurs when attempting a snapshot replication. The publication server running on host `xxx.xxx.xx.xxx` can't access the subscription database. Verify that the `pg_hba.conf` file on the subscription database server permits access from the publication server host.

Problem

```
Unable to synchronize. Reason: FATAL: no pg_hba.conf entry for host "xxx.xxx.xx.xxx", user "user_name", database "db_name", SSL off
```

Resolution Occurs during an implicit synchronization following snapshot replication. The publication server running on host `xxx.xxx.xx.xxx` can't access the subscription server's controller database. Verify that the `pg_hba.conf` file on the subscription server permits access from the publication server host using network address `xxx.xxx.xx.xxx`.

Problem

```
Unable to update publication database information. Reason: Publication control schema does not exist on target database.
```

Resolution

The control schema objects in the publication database might have been deleted or corrupted. For an Oracle publication database, the control schema objects are located in the publication database user's schema. For a Postgres or SQL Server publication database, the metadata database objects are located in schemas `\_edb_replicator_pub`, `\_edb_replicator_sub`, and `\_edb_scheduler`. See Dropping replication slots for log-based synchronization replication.

Problem

```
The user has insufficient privileges to manage publications. Grant required privileges as listed below and then proceed with operation.
```

Resolution

An Oracle publication database user must have `CONNECT`, `RESOURCE`, and `CREATE ANY TRIGGER` privileges.

13.1.2 Where to look for errors

You can look in a number of places to find more detailed information about a replication error.

General replication status

In the Replication Server console, view the replication history. See Viewing replication history.

Snapshot replication failures

View the log file found in the following path:

For Linux:

```
/var/log/xdb-x.x/mtk.log
```

For Windows:

```
POSTGRES_HOME\.enterprisedb\xdb\x.x\mtk.log
```

`POSTGRES_HOME` is the home directory of the Windows postgres account (enterprisedb account for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode). The specific location of `POSTGRES_HOME` depends on your version of Windows. The Replication Server version number is represented by x.x.

See Controlling logging level, log file sizes, and rotation count for more information on setting log file options.

Synchronization replication failures

Check the database server log file.

The typical default location of these files is:

`POSTGRES_INSTALL_HOME/data/pg_log`

Publication and subscription server startup failures

View the publication server and subscription server log files `pubserver.log[.n]` and `subserver.log[.n]` in the following directory:

For Linux:

`/var/log/xdb-x.x`

For Windows:

`POSTGRES_HOME\.enterprisedb\xdb\x.x`

`[.n]` is an optional integer suffix whose presence depends on the `logging.file.count` configuration option described in Controlling logging level, log file sizes, and rotation count.

`POSTGRES_HOME` is the home directory of the Windows postgres account (enterprisedb account for EDB Postgres Advanced Server installed in Oracle-compatible configuration mode). The specific location of `POSTGRES_HOME` depends on your version of Windows. The Replication Server version number is represented by `x.x`.

> **Note**
>
> You can control the severity level of messages logged in these files using a configuration option. SeeControlling logging level, log file sizes, and rotation count.

For Linux only: View the publication service and subscription service startup log files `edb-xdbpubserver.log` and `edb-xdbsubserver.log` as well as the service script log files `edb-xdbpubserver_script.log` and `edb-xdbsubserver_script.log` in directories `/var/log/edb/xdbpubserver` and `/var/log/edb/xdbsubserver`. These log files contain the output from the scripts used to start the publication server and subscription server and can typically be used to confirm the port number on which the publication and subscription servers were started.

> **Note**
>
> The publication service and subscription service startup log files aren't generated for Windows and Mac OS X operating systems.

If there's an entry for a controller database in the Replication Server configuration file, verify that this controller database is accessible with the designated connection information. The controller database parameters are `host, port, type, user`, and `password`.

The following is an example of the content of a Replication Server configuration file with an Oracle database as the controller database:

```
#xDB Replication Server Configuration
Properties
#Tue May 26 13:45:37 GMT-05:00
2015
port=1521
admin_password=ygJ9AxoJEX854elcVIJPTw\=\=
user=pubuser
admin_user=admin
type=oracle
password=ygJ9AxoJEX854elcVIJPTw\=\=
database=xe
host=192.168.2.23
```

See Replication Server configuration file

for information on this file.

Also check the database server log file of the controller database.

Database server errors

Check the database server log file.

The typical default location of these files is:

```
POSTGRES_INSTALL_HOME/data/pg_log
```

Oracle Errors

For problems in Oracle, first find the directory locations of the log files by issuing the following commands in SQL*Plus:

```
SQL> SHOW PARAMETER USER_DUMP_DEST;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
_
user_dump_dest                       string
/usr/lib/oracle/xe/app/oracle/
                                                 admin/XE/udump
```

The directory given by parameter `USER_DUMP_DEST` contains errors given by user processes.

```
SQL> CONNECT system/password
Connected.
SQL> SHOW PARAMETER
BACKGROUND_DUMP_DEST;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
_
background_dump_dest                 string
/usr/lib/oracle/xe/app/oracle/
                                                 admin/XE/bdump
```

The directory given by parameter `BACKGROUND_DUMP_DEST` contains errors given by the Oracle background processes.

Find the latest log file in the preceding directories to investigate the problem.

13.1.3 Common problems checklist

Use the following checklist to verify that the proper configuration steps were followed. Omitting one or more of these steps is a common source of errors.

1. Verify that the database server of the publication database, the database server of the subscription database (for single-master replication systems), and the database servers of the primary nodes (for multi-master replication systems) are all running.

2. When viewing information in the Replication Server console, select **Refresh** in the toolbar to ensure you are viewing the most current information,

especially after making a configuration change to your replication system.

3. Verify that the publication server and the subscription server (for single-master replication systems) are running. If they aren't running and can't be started, see Starting the publication server or subscription server.

4. If you're using an Oracle publication or subscription database, verify that the Oracle JDBC driver file was copied to the `XDB_HOME/lib/jdbc` directory. `XDB_HOME` is the location where you installed Replication Server.

   See Enabling access to Oracle.

5. Verify that the necessary privileges were granted to the publication database user.

   For an Oracle publication database, verify that the publication database user has `CONNECT`, `RESOURCE`, and `CREATE ANY TRIGGER` privileges.

   See Oracle publication database.

   For a SQL Server publication database, verify the following:

   - In the msdb database, verify that the database user mapped to the SQL Server login given in the publication database definition has `EXECUTE` and `SELECT` privileges on schema `dbo`.
   - In the publication database, verify that the database user mapped to the SQL Server login given in the publication database definition has its default schema set to the schema containing the Replication Server metadata database objects.
   - For the same database user, verify that this database user is either the owner of the schema containing the Replication Server metadata database objects or has the following privileges on this schema: `ALTER`, `EXECUTE`, `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
   - For the same database user, verify that this database user has `CREATE TABLE` and `CREATE PROCEDURE` privileges.
   - For the same database user, verify that this database user has `ALTER` privilege on the publication tables.
   - For any database user that will be updating the publication tables, verify that these database users have `EXECUTE`, `SELECT`, and `INSERT` privileges on the schema containing the Replication Server metadata database objects.

   See SQL Server publication database.

   For a Postgres publication database in a single-master replication system, verify that the publication database user is a superuser and has the privilege to modify `pg_catalog` tables. See Postgres publication database.

   For the primary definition node in a multi-master replication system, verify that the publication database user is a superuser and has the privilege to modify `pg_catalog` tables. See Preparing the primary definition node.

   For a primary node other than the primary definition node in a multi-master replication system, verify that the primary node database user is a superuser and has the privilege to modify `pg_catalog` tables. See Preparing more primary nodes.

6. Verify that the necessary privileges were granted to the subscription database user.

   For an Oracle subscription database, verify that the subscription database user has `CONNECT` and `RESOURCE` privileges.

   For a Postgres subscription database, verify that the subscription database user is a superuser and has the privilege to modify `pg_catalog` tables. See Preparing the Subscription Database.

7. **For Linux only:** Verify that the network IP address returned by the `/sbin/ifconfig` command either matches the IP address associated with the host name in the `/etc/hosts` file (see Network IP addresses) or matches the IP address specified with the `java.rmi.server.hostname` configuration option in the publication and subscription server configuration files (see Assigning an IP address for remote method invocation).

13.1.4 Troubleshooting areas

The following topics provide information on specific problem areas you may encounter.

Removing Java can remove Replication Server

When Replication Server is installed on a machine where Java is not present, the JDK is installed as part of the installation process. In this situation the JDK is installed as a dependency of the Replication Server. If you subsequently remove the JDK, Replication Server is also removed.

If Java 1.8 or greater existed before Replication Server was installed, removing the JDK does not remove the Replication Server.

Java runtime errors

If errors are encountered regarding the Java Runtime Environment such as the Java program cannot be found or Java heap space errors, check the parameters set in the Replication Server configuration file `xdbReplicationServer-xx.config`. See Replication Server configuration file for information on this file.

The following is an example of the content of the Replication Server Configuration file:

```
#!/bin/sh

JAVA_EXECUTABLE_PATH="/usr/bin/java"
JAVA_MINIMUM_VERSION=1.8
JAVA_BITNESS_REQUIRED=64
JAVA_HEAP_SIZE="-Xms2048m -
Xmx4096m"
PUBPORT=9051
SUBPORT=9052
```

After you make any changes to the parameters in the Replication Server configuration file, be sure to restart the publication server and subscription server.

Starting the publication server or subscription server

> **Note**
>
> The subscription server applies only to single-master replication systems.

If you can't start the publication server or the subscription server:

1. Check the `pubserver.log` and `subserver.log` files for errors.
2. Check the log file of the database server running the controller database for errors.
3. Verify that the user name and password in the Replication Server configuration file on the hosts running the publication server and subscription server match a database user name and password in the database server running the controller database that the publication server and subscription server are attempting to access.
4. If the controller database is a Postgres database, verify that the `pg_hba.conf` file of its Postgres database server has entries that allow access to the controller database from the IP addresses of the hosts running the publication server and subscription server by the user name in the Replication Server configuration file.

Deleting the control schema and control schema objects

The control schema completely describes the replication system. The control schema and its control schema objects must be complete and correct for replication to occur properly. In addition, the configuration and maintenance operations performed through the Replication Server console or the Replication Server CLI can't be accomplished properly unless the control schema is complete and correct.

There might be occasions in which the control schema becomes corrupted. Either one or more control schema tables containing metadata are inadvertently deleted, or the data in the control schema tables becomes corrupted. Typically, corruption occurs in the form of the first case: one or more control schema tables were deleted, or the entire control schema and its contents were deleted manually using an SQL utility rather than through the operation of the Replication Server console or Replication Server CLI.

In these situations, there might be no other choice but to remove all of the remaining control schema objects using the database management system's deletion functions, which effectively deletes all replication systems managed by the control schema.

The same control schema deletion procedure must be performed in all publication databases that share the same control schema information as the current controller database given in the Replication Server configuration file.

From the viewpoint of the Replication Server console replication tree, a publication server that connects to the controller database has subordinate to it the publication databases sharing the same control schema information.

In the following example, the SMR publication database edb as well as the three MMR primary node databases `mdnnode` , `MMRnode_a` , and `MMRnode_b` are all managed by the same publication server, which connects to the controller database designated in the Replication Server configuration file. Thus, all publication databases `edb` , `mdnnode` , `MMRnode_a` , and `MMRnode_b` contain what should be the same control schema information.

The control schema must be removed from all four publication databases if it is determined that the control schema is corrupted in any of the four publication databases.

Finally, the subscription databases of SMR systems contain a control schema object, which must be deleted as well.

In the preceding example, subscription database subdb contains a control schema object that might have to be deleted if control schema deletion is performed on the publication database.

These instructions describe how to completely remove all control schema objects created by the Replication Server product, leaving just your original publication tables and any replicated subscription tables or publication tables of multi-master system nodes. Hence, the definition and framework for all existing single-master and multi-master replication systems are deleted. In effect, this simulates the situation when you install the Replication Server product for the first time.

After you perform this deletion process, you must recreate single-master replication systems following the directions inCreating a publication onward. You must recreate a multi-master replication system following the directions in Creating a publication onward.

> **Warning**
>
> Don't attempt this if any replication systems are running in production. All replication systems will become inoperable. This section describes what to look for to tell if the control schema isn't complete, and, if so, what to delete to completely remove the replication system. This section doesn't discuss the internal contents of the control schema objects. If all of the control schema objects are present, then review the checklist in Common problems checklist before proceeding with deleting the control schema, as it is fairly unlikely that the content of a control schema table becomes corrupted.

If you decide that you must delete all of the control schema objects:

1. Stop the publication server.

2. Stop the subscription server.

3. Look for the control schema objects contained in a publication database. In the example used in this section, pubuser is the publication database user name. The publication consists of two tables: dept and emp.

For Oracle only: See Oracle control schema objects for a list of Oracle control schema objects.

For SQL Server only: See SQL control schema objects for a list of SQL Server control schema objects.

For Postgres only: See PostgreSQL control schema objects for a list of Postgres control schema objects.

4. If the schema that is supposed to contain the control schema objects (the publication database user name for Oracle or the control schema you created or selected when configuring a SQL Server publication database along with `_edb_replicator_pub`, `_edb_replicator_sub`, and `_edb_scheduler`, or `_edb_replicator_pub, _edb_replicator_sub`, and `_edb_scheduler` for Postgres) is missing, or there are missing database objects under the control schema, then you might need to complete the process of removing all remaining control schema objects.

   If you decide to undergo this procedure, you must remove the control schema objects from all publication databases. You must also remove all subscription metadata objects from the subscription databases. Proceed with Step 7 and repeat Step 7 for all publication databases. Then proceed with Step 8 and repeat Step 8 for all subscription databases.

   If the control schema objects look intact, repeat Step 3 for all other publication databases. If the control schema objects of all publication databases appear intact, then proceed with Step 5.

5. For single-master replication systems, the subscription database contains a single control schema object in the form of a table named `rrep_txset_health`. See Subscription metadata object for a listing of this control schema object for each type of subscription database.

   For each subscription database, verify the presence of this subscription metadata object.

6. If, at this point, all control schemas and control schema objects appear intact in all publication databases and all subscription databases, then chances are that the problem lies elsewhere. Don't proceed with any further steps in this section. Instead, recheck the checklist in Common problems checklist.

   If it was determined that incomplete control schema objects exist, and you decide to go ahead with the deletion process, proceed with Step 7.

7. Repeat this step for every publication database to delete its control schema and control schema objects.

   For Oracle only: If the publication user name still exists, then log onto SQL*Plus or any other Oracle database administration utility and drop all control schema objects owned by the publication user. Alternatively, you can drop the publication database user along with its database objects using the cascade option, but the publication database user must be recreated and privileges reassigned if you intend to rebuild your replication systems. See Preparing the publication database for directions on creating the publication database user. The following example illustrates use of the cascade option:

```
SQL> CONNECT system/password
Connected.
SQL> DROP USER pubuser CASCADE;

User
dropped.
```

   For SQL Server only: If any of the control schema objects listed in Step 3 still exist, then log onto the SQL Server command line program, sqlcmd, or SQL Server Management Studio and drop these objects. The following example assumes some of the control schema objects were created under schema `pubuser`. The other control schema objects are created under `_edb_replicator_pub, _edb_replicator_sub,` and `_edb_scheduler`. The publication tables are `dept` and `emp` located in schema `edb`.

   The following example shows how to delete the jobs in the `msdb` database:

```
1> USE msdb;
2> GO
Changed database context to 'msdb'.
1> EXEC sp_delete_job @job_name = 'rrep_cleanup_job_edb';
2> GO
1> EXEC sp_delete_job @job_name = 'rrep_txset_job_edb';
2> GO
```

```
The next example shows the deletion of the triggers on the non-snapshot only publication
tables:
1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP TRIGGER edb.rrpd_edb_dept;
2> DROP TRIGGER edb.rrpi_edb_dept;
3> DROP TRIGGER edb.rrpu_edb_dept;
4> DROP TRIGGER edb.rrpd_edb_emp;
5> DROP TRIGGER edb.rrpi_edb_emp;
6> DROP TRIGGER edb.rrpu_edb_emp;
7> GO
```

The control schema objects under the `_edb_replicator_pub` schema are dropped as shown by the following:

```
 1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP TABLE _edb_replicator_pub.rrep_lock;
2> DROP TABLE
_edb_replicator_pub.rrep_MMR_pub_group;
3> DROP TABLE _edb_replicator_pub.rrep_MMR_txset;
4> DROP TABLE _edb_replicator_pub.rrep_properties;
5> DROP TABLE _edb_replicator_pub.rrep_publication_subscriptions;
6> DROP TABLE
_edb_replicator_pub.rrep_publication_tables;
7> DROP TABLE _edb_replicator_pub.rrep_tables;
8> DROP TABLE _edb_replicator_pub.rrep_tx_monitor;
9> DROP TABLE
_edb_replicator_pub.rrep_txset;
10> DROP TABLE _edb_replicator_pub.rrep_txset_health;
11> DROP TABLE _edb_replicator_pub.rrep_txset_log;
12> DROP TABLE
_edb_replicator_pub.xdb_cleanup_conf;
13> DROP TABLE
_edb_replicator_pub.xdb_conflicts;
14> DROP TABLE
_edb_replicator_pub.xdb_conflicts_options;
15> DROP TABLE
_edb_replicator_pub.xdb_events;
16> DROP TABLE _edb_replicator_pub.xdb_events_status;
17> DROP TABLE _edb_replicator_pub.xdb_MMR_pub_group;
18> DROP TABLE
_edb_replicator_pub.xdb_pub_database;
19> DROP TABLE _edb_replicator_pub.xdb_pub_table_replog;
20> DROP TABLE _edb_replicator_pub.xdb_pub_replog;
21> DROP TABLE _edb_replicator_pub.xdb_publication_filter;
22> DROP TABLE _edb_replicator_pub.xdb_publication_filter_rule;
23> DROP TABLE _edb_replicator_pub.xdb_publication_subscriptions;
24> DROP TABLE
_edb_replicator_pub.xdb_publications;
25> DROP TABLE _edb_replicator_pub.xdb_pubtables_ignoredcols;
26> DROP TABLE _edb_replicator_pub.xdb_sub_servers;
27> GO
```

For SQL Server 2014 only: Drop the following control schema objects when the publication database is SQL Server 2014:

```
 1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP SEQUENCE _edb_replicator_pub.rrep_tx_seq;
2> DROP SEQUENCE _edb_replicator_pub.rrep_txset_seq;
3> DROP SEQUENCE _edb_replicator_pub.rrep_common_seq;
```

```
4> GO
```

Drop the `_edb_replicator_pub` control schema:

> 1> USE edb; 2> GO
>
> Changed database context to `edb`.
>
> 1> DROP SCHEMA _edb_replicator_pub; 2> GO

The control schema objects under the `_edb_replicator_sub` schema as well as the schema itself are dropped as shown by the following.

> **Note**
>
> **(For SQL Server 2014):** When the publication database is SQL Server 2014, the first table in the following list, `rrep_common_seq`, does not exist. Therefore don't issue the first `DROP TABLE`
>
> `_edb_replicator_sub.rrep_common_seq` command.

```
1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP TABLE _edb_replicator_sub.rrep_common_seq;
2> DROP TABLE
_edb_replicator_sub.xdb_sub_database;
3> DROP TABLE
_edb_replicator_sub.xdb_subscription_tables;
4> DROP TABLE _edb_replicator_sub.xdb_subscriptions;
5> DROP TABLE
_edb_replicator_sub.xdb_tables;
6> DROP SCHEMA _edb_replicator_sub;
7> GO
```

The control schema objects under the `_edb_scheduler` schema as well as the schema itself are dropped as shown by the following:

```
1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP TABLE
_edb_scheduler.sch_pub_BLOB_TRIGGERS;
2> DROP TABLE _edb_scheduler.sch_pub_CALENDARS;
3> DROP TABLE
_edb_scheduler.sch_pub_CRON_TRIGGERS;
4> DROP TABLE
_edb_scheduler.sch_pub_SIMPLE_TRIGGERS;
5> DROP TABLE _edb_scheduler.sch_pub_TRIGGER_LISTENERS;
6> DROP TABLE _edb_scheduler.sch_pub_FIRED_TRIGGERS;
7> DROP TABLE _edb_scheduler.sch_pub_TRIGGERS;
8> DROP TABLE
_edb_scheduler.sch_pub_JOB_LISTENERS;
9> DROP TABLE _edb_scheduler.sch_pub_JOB_DETAILS;
10> DROP TABLE
_edb_scheduler.sch_pub_LOCKS;
11> DROP TABLE _edb_scheduler.sch_pub_PAUSED_TRIGGER_GRPS;
12> DROP TABLE
_edb_scheduler.sch_pub_SCHEDULER_STATE;
13> DROP TABLE
_edb_scheduler.sch_sub_BLOB_TRIGGERS;
14> DROP TABLE _edb_scheduler.sch_sub_CALENDARS;
15> DROP TABLE
_edb_scheduler.sch_sub_CRON_TRIGGERS;
```

```
16> DROP TABLE
_edb_scheduler.sch_sub_SIMPLE_TRIGGERS;
17> DROP TABLE _edb_scheduler.sch_sub_TRIGGER_LISTENERS;
18> DROP TABLE _edb_scheduler.sch_sub_FIRED_TRIGGERS;
19> DROP TABLE _edb_scheduler.sch_sub_TRIGGERS;
20> DROP TABLE
_edb_scheduler.sch_sub_JOB_LISTENERS;
21> DROP TABLE _edb_scheduler.sch_sub_JOB_DETAILS;
22> DROP TABLE
_edb_scheduler.sch_sub_LOCKS;
23> DROP TABLE _edb_scheduler.sch_sub_PAUSED_TRIGGER_GRPS;
24> DROP TABLE
_edb_scheduler.sch_sub_SCHEDULER_STATE;
25> DROP SCHEMA _edb_scheduler;
26> GO
```

The control schema objects under the `pubuser` schema are dropped as shown by the following:

```
 1> USE edb;
2> GO
Changed database context to 'edb'.
1> DROP FUNCTION pubuser.getPackageVersionNumber;
2> DROP PROCEDURE pubuser.CleanupShadowTables;
3> DROP PROCEDURE pubuser.ConfigureCleanUpJob;
4> DROP PROCEDURE pubuser.ConfigureCreateTxSetJob;
5> DROP PROCEDURE pubuser.CreateMultiTxSet;
6> DROP PROCEDURE pubuser.CreateTableLogTrigger;
7> DROP PROCEDURE pubuser.CreateTxSet;
8> DROP PROCEDURE
pubuser.CreateTxSet_old;
9> DROP PROCEDURE pubuser.CreateUniTxSet;
10> DROP PROCEDURE pubuser.GetNewTxsCount;
11> DROP PROCEDURE
pubuser.JobCleanup;
12> DROP PROCEDURE pubuser.JobCreateTxSet;
13> DROP PROCEDURE pubuser.LoadPubTableList;
14> DROP PROCEDURE pubuser.RemoveCleanupJob;
15> DROP PROCEDURE
pubuser.RemoveCreateTxSetJob;
16> DROP TABLE pubuser.rrst_edb_dept;
17> DROP TABLE
pubuser.rrst_edb_emp;
18> GO
```

For Postgres only: If any of the schemas `_edb_replicator_pub`, `_edb_replicator_sub`, or `_edb_scheduler` still exist in the publication database, drop the schema and all of its database objects. The following example shows a connection established in psql to the publication database edb. The `DROP SCHEMA CASCADE` statement is then used to drop the schemas.

```
 edb=# \c edb
 enterprisedb
You are now connected to database "edb" as user
"enterprisedb".

edb=# DROP SCHEMA _edb_replicator_pub
CASCADE;
NOTICE:  drop cascades to 51 other
objects
DETAIL:  drop cascades to sequence
_edb_replicator_pub.rrep_common_seq
drop cascades to sequence
_edb_replicator_pub.rrep_tx_seq

.

.
```

```
.
DROP SCHEMA

edb=# DROP SCHEMA _edb_replicator_sub
CASCADE;
NOTICE:  drop cascades to 9 other
objects
DETAIL:  drop cascades to sequence
_edb_replicator_sub.rrep_common_seq
drop cascades to table
_edb_replicator_sub.xdb_sub_database

.

.

.
DROP SCHEMA

edb=# DROP SCHEMA _edb_scheduler
CASCADE;
NOTICE:  drop cascades to 40 other
objects
DETAIL:  drop cascades to table
_edb_scheduler.sch_pub_job_details
drop cascades to table
_edb_scheduler.sch_pub_job_listeners

.

.

.
DROP SCHEMA
```

For synchronization replication with the trigger-based method, in the schema containing the publication tables, drop the triggers and trigger functions associated with the publication tables:

```
 edb=# SET search_path TO
 edb;
SET
edb=# DROP FUNCTION rrpd_edb_dept_tgfunc() CASCADE;
NOTICE:  drop cascades to trigger rrpd_edb_dept on table
dept
DROP FUNCTION
edb=# DROP FUNCTION rrpi_edb_dept_tgfunc() CASCADE;
NOTICE:  drop cascades to trigger rrpi_edb_dept on table
dept
DROP FUNCTION
edb=# DROP FUNCTION rrpu_edb_dept_tgfunc() CASCADE;
NOTICE:  drop cascades to trigger rrpu_edb_dept on table
dept
DROP FUNCTION
edb=# DROP FUNCTION rrpd_edb_emp_tgfunc()
CASCADE;
NOTICE:  drop cascades to trigger rrpd_edb_emp on table
emp
DROP FUNCTION
edb=# DROP FUNCTION rrpi_edb_emp_tgfunc()
CASCADE;
NOTICE:  drop cascades to trigger rrpi_edb_emp on table
emp
DROP FUNCTION
edb=# DROP FUNCTION rrpu_edb_emp_tgfunc()
CASCADE;
```

```
NOTICE:  drop cascades to trigger rrpu_edb_emp on table
emp
DROP FUNCTION
```

8. Repeat this step for every subscription database to delete its control schema and control schema object.

   For single-master replication systems, the subscription database contains a single control schema object in the form of a table named `rrep_txset_health`. Delete this table in all subscription databases. For SQL Server and Postgres subscription databases, delete the parent schema _edb_replicator_sub as well.

   For Oracle subscription databases, the parent schema isn't generated by Replication Server, so it's your decision as to whether to keep or delete the parent schema.

   For Oracle only: The `RREP_TXSET_HEALTH` table is created in the subscription database user's schema. Drop this table.

```
SQL> CONNECT subuser/password
Connected.
SQL> DROP TABLE rrep_txset_health;

Table
dropped.
```

   For SQL Server only: The `rrep_txset_health` table is created in the schema named _edb_replicator_sub. Drop this table and schema.

```
1> USE
subdb;
2> GO
Changed database context to 'subdb'.
1> DROP TABLE _edb_replicator_sub.rrep_txset_health;
2> GO
1> DROP SCHEMA _edb_replicator_sub;
2> GO
```

   For Postgres only: The `rrep_txset_health` table is created in the schema named `_edb_replicator_sub`. Drop this table and schema.

```
edb=# \c subdb enterprisedb
You are now connected to database "subdb" as user
"enterprisedb".
subdb=# DROP SCHEMA _edb_replicator_sub
CASCADE;
NOTICE:  drop cascades to table
_edb_replicator_sub.rrep_txset_health
DROP SCHEMA
```

9. In the Replication Server configuration file, delete the lines containing the following parameters: `user`, `password`, `host`, `port`, `database`, and `type`.

   Keep the lines with the following parameters: `admin_user`, `admin_password`, and `license_key` (if it exists).

   See Replication Server configuration file for information on the Replication Server configuration file. See Installation details for the file system location of the Replication Server configuration file.

   The absence of these parameters prevents the publication server and subscription server from attempting to connect to this database upon publication and subscription server startup.

   The Replication Server configuration file appears as follows without the controller database connection and authentication information:

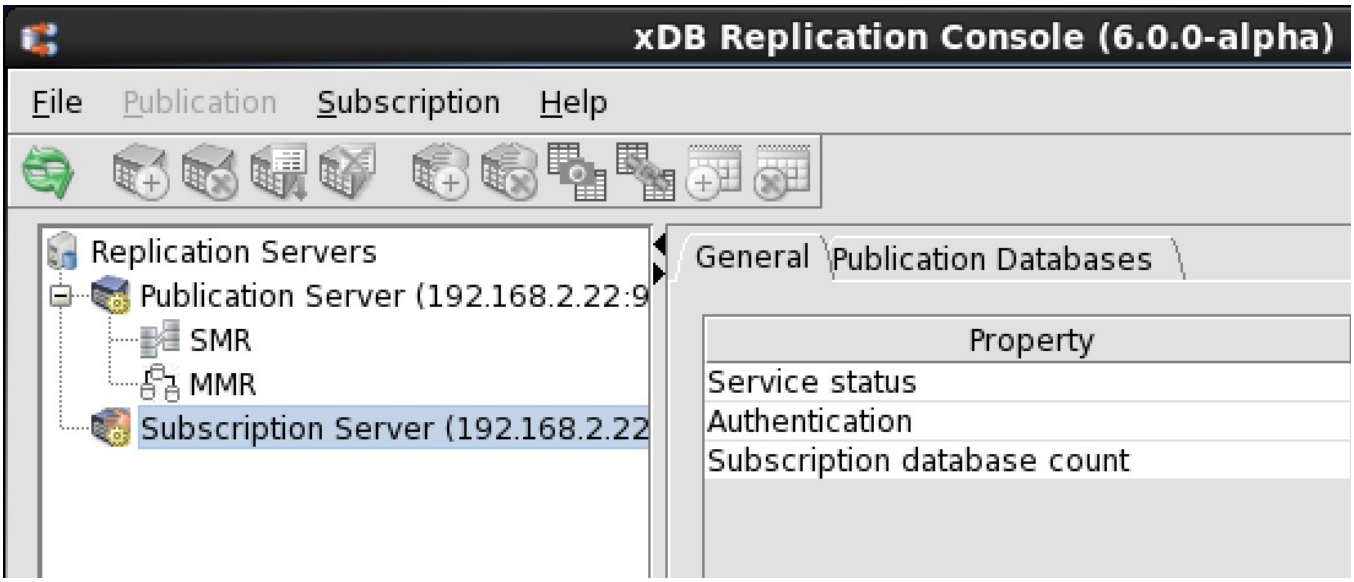
```
#xDB Replication Server Configuration
Properties
```

```
#Fri Jan 30 17:34:06 GMT-05:00
2015
admin_password=ygJ9AxoJEX854elcVIJPTw\=\=
admin_user=enterprisedb
```

10. Start the publication server.

11. Start the subscription server.

12. The replication tree appears as follows:



All the nodes under the SMR and MMR type nodes beneath the Publication Server node and under the Subscription Server node no longer appear.

13. Recreate the replication system as described in Creating a publication onward for a single-master replication system. See Creating a publication for a multi-master replication system.

Dropping replication slots for log-based synchronization replication

As described in Logical replication slots logical replication slots are used for the log-based method of synchronization replication. While a log-based replication system is in use, these replication slots remain connected to the Postgres databases. When the replication system is removed, these replication slots are also deleted.

There are circumstances when you want to drop a Postgres database used in a replication system but the replication system can't be removed according to the normal procedure of using the Replication Server console or the Replication Server CLI. In such cases, it is assumed that the replication system has somehow become corrupted, and you want to delete the replication system components, including some of the databases used in the replication system.

When the log-based method is used, certain additional procedures might be required to remove the replication slots before dropping the databases. Postgres doesn't permit a database to be dropped if a replication slot is connected to it. The following describes how you cam remove the replication slots to drop a database.

> **Warning**
>
> Don't attempt this if any replication systems are running in production. All replication systems will become inoperable.

Replication slots can be displayed by the following query on the database server containing the databases to be dropped:

```
edb=# SELECT slot_name, slot_type, database, active, active_pid FROM
pg_replication_slots;
```

```
  slot_name   | slot_type | database | active | active_pid
--------------+-----------+----------+--------+------------
 xdb_14793_5 | logical   | edb      | t      |       5288
 xdb_79910_5 | logical   | MMRnode  | t      |       5327
(2 rows)
```

The active column indicates whether the replication slot is active. To deactivate an active replication slot, first stop the publication server. If the active column of the replication slot now displays f for false then you can remove the replication slot.

If the replication slot is still active, then you can deactivate it by terminating the process shown in the `active_pid` column with the following command:

```
edb=# SELECT
pg_terminate_backend(5327);
```

```
 pg_terminate_backend
----------------------
 t
(1 row)
```

The following now shows that replication slot `xdb_79910_5` for database `MMRnode` was deactivated:

```
edb=# SELECT slot_name, slot_type, database, active, active_pid FROM
pg_replication_slots;
```

```
  slot_name   | slot_type | database | active | active_pid
--------------+-----------+----------+--------+------------
 xdb_14793_5 | logical   | edb      | t      |       5288
 xdb_79910_5 | logical   | MMRnode  | f      |
(2 rows)
```

Drop the replication slot with the following command by specifying the slot name:

```
edb=# SELECT pg_drop_replication_slot('xdb_79910_5');
```

```
 pg_drop_replication_slot
--------------------------


(1 row)
```

Now, the dropped replication slot doesn't appear when the `pg_replication_slots` directory is queried:

```
edb=# SELECT slot_name, slot_type, database, active, active_pid FROM
pg_replication_slots;
```

```
  slot_name   | slot_type | database | active | active_pid
--------------+-----------+----------+--------+------------
 xdb_14793_5 | logical   | edb      | t      |       5288
(1 row)
```

The database can now be successfully dropped:

```
edb=# DROP DATABASE
MMRnode;
```

```
DROP DATABASE
```

In addition, you can display replication origins with the following command:

```
edb=# SELECT * FROM pg_replication_origin;
```

```
 roident |        roname
---------+-----------------------
       1 | xdb_MMRnode_emp_pub_1
       2 | xdb_edb_emp_pub_6
(2 rows)
```

You can use the following command to remove a replication origin:

```
edb=# SELECT pg_replication_origin_drop('xdb_MMRnode_emp_pub_1');
```

```
 pg_replication_origin_drop
----------------------------

(1 row)
```

The following shows this replication origin was removed:

```
edb=# SELECT * FROM pg_replication_origin;
```

```
 roident |        roname
---------+-----------------------
       2 | xdb_edb_emp_pub_6
(1 row)
```

For more information on logical decoding functions see *9.26.6 Replication Functions* under *Section 9.26 System Administration Functions* in the PostgreSQL core documentation.

After performing this process, it is unlikely that you can remove the entire replication system with the Replication Server console or Replication Server CLI. You must remove the remaining replication system components manually. Part of this process is removing the control schema and control schema objects from the publication databases. See Dropping replication slots for log-based synchronization replication for information on this procedure.

13.2 Miscellaneous Replication Server processing topics

These topics include:

- Handling special characters in replication data
- Replicating Oracle partitioned tables
- Performing an offline snapshot and subsequent synchronization
- Generating an encrypted password
- Writing cron expressions

13.2.1 Publication and subscription server configuration options

The publication server and the subscription server support various configuration options for purposes such as the following:

- Optimize synchronization performance based on the types of transactions affecting the publication. (See Optimizing synchronization replication for details on these options.)
- Utilize alternate loading methods in snapshot replications. (See Optimizing snapshot replication for details on these options.)

- Special configuration options for multi-master replication. (See Optimizing performance for details on these options.)
- Adjust memory usage and transaction size for replications.
- Replicate certain Oracle partitioned table types.
- Replicate special characters found in publication data.
- Set special configuration options for the log-based method of synchronization replication. (See Quoted identifiers and default case translation for details on these options.)

Most options apply only to the publication server, although a few are used by the subscription server.

Set and pass the configuration options for the publication server in a text file called the publication server configuration file: `xdb_pubserver.conf`.

Set and pass the configuration options for the subscription server in a text file called the subscription server configuration file: `xdb_subserver.conf`.

See Installation details for the locations of these files.

Modified publication server configuration options take effect after you restart the publication server. Similarly, modified subscription server configuration options take effect after you restart the subscription server. The configuration options that were explicitly put into effect by overriding their defaults in the configuration files are logged in the publication server log file and the subscription server log file. Installation details contains the locations of these log files.

To set the configuration options:

1. The publication and subscription server configuration files are created during Replication Server installation and already contain all of the configuration options as comments with their default settings.

   To change the setting of a configuration option, edit the publication server or subscription server configuration file by removing the comment symbol (#) from the option and substituting the desired value in place of the currently coded value.

   The following example shows a portion of the publication server configuration file in which replacement of null characters in the publication data is activated and the replacement character is set to the question mark character.

   ```
   replaceNullChar = true

   #Null Replacement
   Character
   nullReplacementChar =
   ?
   ```

2. Restart the publication or subscription server.

   Use the following command for CentOS 7 or RHEL 7 and Rocky Linux 8 or AlmaLinux 8 or RHEL 8:

   ```
   systemctl restart edb-xdbpubserver
   ```

   Use the following command for previous Linux versions:

   ```
   service edb-xdbpubserver restart
   ```

13.2.1.1 Controlling logging level, log file sizes, rotation count, and locale

> **Note**
>
> These options apply to the publication server and the subscription server unless otherwise specified.

The following options control various aspects of message logging in the publication server log file, the subscription server log file, and the Migration

Toolkit log file.

See Publication and subscription server startup failures and Snapshot replication failures for more information.

`logging.level`

Set the `logging.level` option to control the severity of messages written to the publication server log file and the subscription server log file.

```
logging.level={OFF | SEVERE | WARNING | INFO | CONFIG | FINE | FINER | FINEST | ALL}
```

The default value is `WARNING`.

`logging.file.size`

Set the `logging.file.size` option to control the maximum file size (in megabytes) of the publication server log file and the subscription server log file.

> **Note**
>
> If `logging.file.count` is set to `0`, the setting of `logging.file.size` is ignored. The log file is allowed to grow without limit.

```
logging.file.size=n
```

The default value is `50`, in megabytes.

`logging.file.count`

Set the `logging.file.count` option to control the number of files in the log file rotation history of the publication server log file and the subscription server log file.

```
logging.file.count=n
```

The default value is `20`.

A non-zero value of n specifies the maximum number of log files to create.

> **Note**
>
> The publication server log file named `pubserver.log` is used here as an example. For the subscription server, the log file is named `subserver.log`.

- Specify a value of `0` to disable log file rotation and create a single, unlimited size log file named `pubserver.log`. This log file will grow to an unlimited size, ignoring any setting of `logging.file.size`.
- Specify a value of `1` to disable log file rotation and create a single, limited-size log file named `pubserver.log`. The log file is deleted and a new one is created each time the log file reaches the size limit set by `logging.file.size`.
- Specify a value of `2` or greater to enable log file rotation. All log file names have an integer suffix (for example, `pubserver.log.0`, `pubserver.log.1`, `pubserver.log.2`).

When log file rotation is enabled, the log file with the greatest integer suffix contains the oldest messages. When there are enough messages to generate every file in the history rotation, the oldest messages are in `pubserver.log.n-1`, where n is the setting of `logging.file.count`. Log file `pubserver.log.0` is the current, active log file containing the most recent messages.

When log file rotation is enabled and the current, active log file (`pubserver.log.0`) reaches the size specified by `logging.file.size`, then the following events occur:

- The log file containing the oldest messages (`pubserver.log.n-1`) is deleted.
- Each remaining log file is renamed with the next greater integer suffix (`pubserver.log.m` is renamed to `pubserver.log.m+1`, with m varying from `0` to `n-2`).
- A new, active log file is created (`pubserver.log.0`).

`logging.default.locale`

Set the `logging.default.locale` option to use either the current system locale or English (en) for publication and subscription logs.

```
logging.default.locale={system | en}
```

The default value is `system`.

> **Note**
>
> This option is applicable only for publication and subscription logs and isn't supported for `mtk.log`.
>
> The RepCLI and RepConsole logs continue showing text in the default locale.

`mtk.logging.file.size`

> **Note**
>
> This option applies only to the publication server.

Set the `mtk.logging.file.size` option to control the maximum file size (in megabytes) of the Migration Toolkit log file.

```
mtk.logging.file.size=n
```

The default value is `50`, in megabytes.

`mtk.logging.file.count`

> **Note**
>
> This option applies only to the publication server.

Set the `mtk.logging.file.count` option to control the number of files in the log file rotation history of the Migration Toolkit log file.

```
mtk.logging.file.count=n
```

The default value for n is `20`.

A non-zero value of n specifies the maximum number of history log files to create.

- Specify a value of `0` to disable log file rotation and create a single, limited-size log file named `mtk.log`. The log file is deleted and a new one is created each time the log file reaches the size limit set by `mtk.logging.file.size`.
- Specify a value of `1` or greater to enable log file rotation. All log file names have an integer suffix (for example, `mtk.log.1, mtk.log.2`).

When log file rotation is enabled, the log file with the greatest integer suffix contains the oldest messages. When there are enough messages to generate every file in the history rotation, the oldest messages are in `mtk.log.n`, where `n` is the setting of `mtk.logging.file.count`.

Log file `mtk.log` is the current, active log file containing the most recent messages.

When the current, active log file (`mtk.log`) reaches the size specified by `mtk.logging.file.size`, then the following events occur:

- The log file containing the oldest messages (`mtk.log.n`) is deleted.
- Each remaining log file with a suffix is renamed with the next greater integer suffix (`mtk.log.m` is renamed to `mtk.log.m+1`, with `m` varying from `1` to `n-1`).
- Log file `mtk.log` is renamed to `mtk.log.1`.
- A new, active log file is created (`mtk.log`).

13.2.1.2 Replacing null characters

> **Note**
>
> These options apply only to the publication server.

A character consisting of binary zeros (also called the null character string) and represented as 000 in octal or 0x00 in hexadecimal can result in errors when attempting to load such data into a Postgres character column.

You might get the following error in the Migration Toolkit log file when performing a snapshot replication of an Oracle table that contains the null character string:

```
 Loading Table Data in 8 MB batches...
Disabling FK constraints & triggers on edb.null_test before truncate...
Truncating table NULL_TEST before data load...
Disabling indexes on edb.null_test before data load...
Loading Table: NULL_TEST ...
Error Loading Data into Table: NULL_TEST: ERROR: invalid byte sequence for encoding "UTF8": 0x00
  Where: COPY null_test, line 2
```

The same circumstance might also produce the following error in the Migration Toolkit log file:

```
 Loading Table Data in 8 MB batches...
Disabling FK constraints & triggers on edb.null_clob before truncate...
Disabling indexes on edb.null_clob before data load...
Loading Large Objects into table: NULL_CLOB ...
[NULL_CLOB] Migrated 1 rows.
com.edb.util.PSQLException: Zero bytes may not occur in string parameters., Skipping Batch
```

If any of these errors occur, you can set an option that converts each null character encountered in character columns of the source tables to a space character or to any other character of your choice before loading the target tables.

> **Note**
>
> This option doesn't alter null characters encountered in columns with binary data types such as Oracle RAW and BLOB data types.

Set the following option:

```
replaceNullChar=true
```

This option results in the substitution of a space character for each null character encountered in the source character data. If you want to use a character other than a space character to replace each null character, use the following option in addition to `replaceNullChar=true`.

```
nullReplacementChar=<char>
```

`<char>` is a single character you want to substitute for the null character. For example, the following combination replaces each null character with the hash symbol `#`.

```
replaceNullChar=true
```

```
nullReplacementChar=#
```

13.2.1.3 Schema migration options

> **Note**
>
> This option applies only to the subscription server.

This option controls how certain aspects of the publication database schema are migrated to the subscription database.

skipCheckConst

By default, column `CHECK` constraints from publication tables are migrated to the subscription table definitions when the subscription is created. Set this option to `true` if you don't want `CHECK` constraints as part of the subscription table definitions.

Setting this option to `true` is useful if the `CHECK` constraint is based on a built-in function supported by the publication database server, and this built-in function doesn't exist in the subscription database server.

```
skipCheckConst={true | false}
```

The default value is `false`.

13.2.1.4 Replicating Oracle partitioned tables

> **Note**
>
> You must set this option to the same value for both the publication server and the subscription server.

> **Note**

> This feature applies only for subscriptions in an EDB Postgres Advanced Server database. It doesn't apply to subscriptions in a PostgreSQL database.

In Oracle, table partitioning allows you to store table rows in different physical locations (tablespaces) according to a rule defined on the table.

The most common types of Oracle table partitioning are the following:

- Range partitioning. Ranges of values defined on a column determine the tablespace in which a row is stored.
- List partitioning. A list of values defined on a column determines the tablespace in which a row is stored.
- Hash partitioning. An algorithm on a column generates a hash key, which determines the tablespace in which a row is stored.

Limitations for hash partitioning

- Data is consistent only at the parent table across Oracle and EDB Postgres Advanced Server nodes but not across child partition tables. This is due to the difference in hashing algorithms in Oracle and EDB Postgres Advanced Server that distribute data in the partition.
- Truncate operation on child partition table: You can truncate child partitions in Oracle using the `ALTER` command. Replication Server doesn't support the `ALTER` command when the source database type is Oracle or SQL server. In this case, don't truncate child partition tables outside the Replication Server, leading to data consistency across nodes.

> **Note**
>
> If you're using EDB Postgres Advanced Server, table partitioning using Oracle-compatible table partitioning syntax is an available feature. See the section on table partitioning in the Database compatibility for Oracle developers for information. See Replicating Postgres partitioned tables for information on including Postgres partitioned tables in a replication system. The `importPartitionAsTable` option described here applies only to table partitioning in an Oracle database.

The `importPartitionAsTable` option controls what happens when an Oracle partitioned table is part of the publication.

```
importPartitionAsTable={true | false}
```

The default value is `false`.

Depending on the Oracle partitioned table type and the setting of the `importPartitionAsTable` option, one of the following can occur:

- A set of inherited tables is created in EDB Postgres Advanced Server to which the Oracle partitioned table is replicated. The rows can be stored in different physical locations.
- A plain, single table with no inheritance is created in EDB Postgres Advanced Server to which the Oracle partitioned table is replicated. All rows are stored in one physical location.
- No table is created in EDB Postgres Advanced Server for the Oracle partitioned table, and an error message is written to the Migration Toolkit log file.

When `importPartitionAsTable=false` (the default setting), the following occurs:

- A list partitioned table is replicated as a set of inherited EDB Postgres Advanced Server tables.
- A range partitioned table is replicated as a set of inherited EDB Postgres Advanced Server tables.
- A hash partitioned table is replicated as a set of inherited EDB Postgres Advanced Server tables.

> **Note**
>
> If subscription tables are created as sets of EDB Postgres Advanced Server inherited tables, then you must also set the `enableConstBeforeDataLoad` option in the publication server configuration file to `true`. See Specifying a custom URL for an Oracle JDBC connection for information on the `enableConstBeforeDataLoad` option.

When `importPartitionAsTable=true`, the following occurs:

- A list partitioned table is replicated as a single EDB Postgres Advanced Server table with no inheritance.
- A range partitioned table is replicated as a single EDB Postgres Advanced Server table with no inheritance.
- A hash partitioned table is replicated as a single EDB Postgres Advanced Server table with no inheritance.

Setting the `importPartitionAsTable` option to `true` allows you to replicate a broader range of Oracle partitioned table types but as normal EDB Postgres Advanced Server tables without simulating partitions by using inheritance.

13.2.1.5 Specifying a custom URL for an Oracle JDBC connection

> **Note**
>
> This option applies only to the publication server.

By default, Replication Server supports the basic thin-client URL pattern for an Oracle JDBC connection. If you need to specify custom connectivity credentials, specify the advanced URL using the following option.

`oraJDBCCustomURL=customURL_string`

The following is an example of custom connectivity to an Oracle database.

```
oraJDBCCustomURL=jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=$HOST)(PORT=$PORT))
(CONNECT_DATA=(SERVICE_NAME =$SERVICE_NAME)(SERVER=DEDICATED)))
```

The parameters prefixed with a dollar sign `($)` are dynamically replaced based on the actual connection values specified when adding the Oracle publication database (see Adding a publication database). Alternatively, you can replace the parameters prefixed with a dollar sign with hardcoded values in the URL string, in which case these hardcoded values override what is specified when adding the publication database.

13.2.1.6 Snapshot replication options

> **Note**
>
> These options apply only to the publication server unless otherwise specified.

The server configuration options apply to snapshot replications.

`escapeTabDelimiter`

When `JDBC COPY` is used in snapshot replication, the data delimiter between column values is an escaped tab character (t). Set this option to `false` if you don't want to escape the tab delimiter character.

`escapeTabDelimiter={true | false}`

The default value is `true` .

`mtkCopyDelimiter`

When `JDBC COPY` is used in snapshot replication, the data delimiter between column values is an escaped tab character (t). Set this option to change the data delimiter character.

`mtkCopyDelimiter=c`

`c` denotes the single replacement character for the data delimiter.

The default value is `\t.`

`enableConstBeforeDataLoad`

The `enableConstBeforeDataLoad` option controls whether table constraints, including triggers, are reenabled before loading data into target tables. The default process is that the tables are loaded first, and then the constraints are enabled afterwards.

Activate this option if triggers affect how data is loaded into the target tables.

If target tables are created as sets of Postgres-inherited tables resulting from partitioned Oracle source tables, then you must enable this option.

`enableConstBeforeDataLoad={true | false}`

The default value is `false.`

13.2.1.7 Assigning an IP address for remote method invocation

> **Note**
>
> This option applies to the publication server and the subscription server.

For Linux only:

An alternative method to modifying the `/etc/hosts` file so that the host name is associated with a non-loopback IP address, as discussed in Network IP addresses, is to specify the network IP address using the `java.rmi.server.hostname` option.

In the publication server configuration file, set this option to the network IP address of the host running the publication server.

In the subscription server configuration file, set this option to the network IP address of the host running the subscription server.

`java.rmi.server.hostname=xxx.xxx.xx.xxx`

For example, instead of modifying the `/etc/hosts` file to look like the following for a publication or subscription server running on host `192.168.2.19`:

```
#127.0.0.1              localhost.localdomain localhost
192.168.2.19             localhost.localdomain localhost
```

You can set the IP address in the server configuration file as shown by the following:

```
#On Linux machines, the localhost to real IP may not give correct results.
Hence
#users are advised to override the following property with server IP
address
java.rmi.server.hostname=192.168.2.19
```

> **Note**
>
> If you're still facing issues while accessing the publication server and subscription server via the remote method invocation, add `-Djava.rmi.server.hostname=<IP address>` in `runPubServer.sh` and `runSubServer.sh`, respectively, before launching the servers. The `<IP address>` is the actual IP address where the publication and subscription servers are running.

13.2.1.8 Using pgAgent job scheduling

> **Note**
>
> This option applies only to the publication server.

> **Note**
>
> Using pgAgent job scheduling is useful only if Postgres is the publication database.

> **Note**
>
> You must have pgAgent installed and running on the host where the publication database resides.

When the `pgdbschedule` option is set to `true`, Replication Server uses the pgAgent job scheduler instead of the default Quartz job scheduler.

When activated, pgAgent takes over the following scheduling tasks from Quartz:

- Scheduling shadow table history cleanup in the publication database. See Scheduling shadow table history cleanup.
- Scheduling transaction set creation. A transaction set creation job is scheduled to run every hour to create transaction sets from the updates on the source tables. Transaction sets are applied to the target tables.

Unlike the Quartz scheduler, pgAgent can run and perform its tasks even if the publication server isn't running.

```
pgdbschedule={true | false}
```

The default value is `false`.

13.2.1.9 Forcing immediate shadow table cleanup

> **Note**
>
> This option applies only to the publication server.

A cleanup job is provided that can run on demand or on a schedule to remove dead (processed) tuples from the shadow tables (see Managing history).

However, to perform even quicker cleanup scheduling, turn on this option to force the cleanup of shadow tables after every synchronization replication.

```
postSyncShadowTableCleanup={true | false}
```

The default value is `false`.

13.2.1.10 Setting event history cleanup threshold

The event history cleanup job is scheduled to run every day at 12 a.m. to remove completed, historical, event and replication history data from the control schema `xdb_events`, `xdb_events_status`, `xdb_pub_replog`, and `xdb_pub_table_replog` tables that are older than `n` days. By default, the history data older than seven days is removed.

Specify a value of `0` to clean up all completed event history and replication history data, regardless of its age.

See Cleaning up event history for information on cleaning up event and replication history.

```
historyCleanupDaysThreshold=n
```

The default value is `7`.

13.2.1.11 DDL change replication table locking

> **Note**
>
> This option applies only to the publication server.

When the DDL change replication process is invoked, each affected table in the replication system is acquired in turn with an exclusive lock before the DDL change is applied to the table.

Set `ddlChangeTableLock` to `false` if you don't want an exclusive lock placed on the table before applying the DDL change. Set this option to `false` only if there are no write transactions expected on the target table. If write transactions do occur, they might not be recorded by the replication system.

See Replicating DDL changes for more information.

```
ddlChangeTableLock={true | false}
```

The default value is `true`.

13.2.1.12 Persisting zero transaction count replication history

> **Note**
>
> This option applies only to the publication server.

If you want to maintain zero transaction count records in the replication history after the publication server restarts, set `persistZeroTxRepEvent` to `true`. Otherwise, zero transaction count records are no longer available once the publication server restarts.

See Viewing replication history.

```
persistZeroTxRepEvent={true | false}
```

The default value is `false`.

13.2.1.13 Skipping grants of table-level user privileges on MMR target tables

> **Note**
>
> This option applies only to the publication server.

When creating non-MDN nodes in a multi-master replication system, the publication server creates the publication tables and their corresponding shadow tables in the non-MDN node database.

When `skipTablePrivileges` is set to `false`, which is the default value, the database user privileges on the publication tables in the primary definition node are granted to the same database users on the publication tables in the newly created non-MDN node.

The required privileges are also granted to these database users on the corresponding shadow tables in the non-MDN node so these database users can perform updates on the publication tables. The changes are recorded in the corresponding shadow tables. This enables proper synchronization replication of any such changes.

This granting of privileges occurs only for database users with privileges on the primary definition node publication tables at the time the non-MDN node is defined using the Replication Server console or CLI. If you don't want the publication server to grant these database user privileges to the non-MDN publication tables and shadow tables when defining the non-MDN node, set `skipTablePrivileges` to `true`. In this case, you must explicitly grant the privileges on the publication tables and corresponding shadow tables in the non-MDN node for any database user that you want to provide update access to on these tables. See Step 2 of Postgres publication database for information about the required privileges.

This usage is typically for the case when database users who access the non-MDN node publication tables are different from the database users who access the primary definition node publication tables.

```
skipTablePrivileges={true | false}
```

The default value is `false`.

13.2.1.14 Applying grants of table-level user privileges on SMR target tables

> **Note**
>
> This option applies only to the subscription server. This option also applies only when both the publication database and the subscription database are Postgres databases.

When creating a subscription in a single-master replication system, the subscription server creates the subscription tables in the subscription database.

When `skipTablePrivileges` is set to `true`, which is the default value, no database user privileges are granted on these subscription tables to any database user. By default, the subscription database user specified when the subscription database definition is created (see Adding a subscription database) is the owner of the subscription tables.

This is the typical, expected scenario since the data in subscription tables should be updated only by Replication Server.

Database users that require access to the subscription tables must be explicitly granted such privileges.

If, however, you do want the subscription server to grant database user privileges to the subscription tables for the same database users that already have access privileges to the publication tables, set `skipTablePrivileges` to `false` in the subscription server configuration file. (The setting of `skipTablePrivileges` in the publication server configuration file is ignored for this process in a single-master replication system.)

In this case, the same access privileges are granted on the subscription tables to database users with privileges on the publication tables whwn the subscription is defined using the Replication Server console or CLI.

```
skipTablePrivileges={true | false}
```

The default value is `true.`

13.2.1.15 Log-based method of synchronization options

> **Note**
>
> This option applies only to the publication server.

`walTxSetCreationInterval`

When using the log-based method of synchronization replication, the `walTxSetCreationInterval` option controls the time interval between creations of the transaction sets. This time interval affects the size of the transaction set (that is, the batch size). The default setting results in creating a transaction set every 5,000 milliseconds (5 seconds), assuming changes to the publication tables to be replicated are available.

Adjust this value based on the workload, that is, the transaction per minute (TPM) rate on the target publication tables.

If the TPM rate is on a higher end, set the `walTxSetCreationInterval` option to a relatively low value.

If the TPM rate is on a lower end, set the option to a higher value. Doing so ensures that a transaction set is large enough to allow an average batch size in the range of 100 to 500 transactions.

`walTxSetCreationInterval=n` The default value is `5000`, in milliseconds.

`walStreamQueueLimit`

The `walStreamQueueLimit` option defines the upper limit for the number of WAL entries that can be held in the queue pending for processing at a point in time. Once the queue becomes full, the WAL stream receiver blocks additions until space becomes available in the queue as transaction entries are popped out of the queue for processing.

A value of `0` indicates there is no upper limit. Too high a setting can result in Java heap space out-of-memory errors. SeeSetting heap memory size for the publication and subscription servers for information on adjusting the Java heap memory size.

`walStreamQueueLimit=n`

The default value is `10000`.

`pendingTxSetThreshold`

The `pendingTxSetThreshold` option defines the upper threshold limit for the number of pending transaction sets that, when reached, causes the extraction of transaction data from the WAL stream and its parsing to be put on hold until the pending transactions are processed.

This threshold is to avoid a situation in which the data is continuously pushed over the WAL stream channel but isn't being processed and applied due to some failure or lack of scheduling of the synchronization process. This can result in a Java heap space out-of-heap memory error. See Setting heap memory size for the publication and subscription servers for information on adjusting the Java heap memory size.

`pendingTxSetThreshold=n`

The default value is `10`.

13.2.1.16 Setting the Apache DBCP connection validation query timeout

> **Note**
>
> This option applies only to the publication server.

The Apache Commons Database Connection Pooling (DBCP) component is the Apache pooling framework used by the publication server for establishing JDBC connections.

The `jdbc.pool.validationQueryTimeout` option controls the timeout setting when a validation query is executed when allocating a connection from the pool. This is the amount of time in seconds before an exception is returned if the connection validation query doesn't succeed.

The default timeout value is 30 seconds. In situations in which network connections aren't reliable, you can increase the timeout value to allow more time for the connection attempt. Specify a value of `0` if you don't want a timeout value.

```
jdbc.pool.validationQueryTimeout=n
```

The default value is `30`.

13.2.2 Encrypting the password in the Replication Server configuration file

If you need to change the password in the Replication Server configuration file, you must first encrypt the password. Use the `encrypt` command of the Replication Server CLI to generate the encrypted form of the password from its plain text form given in an input file.

1. Create a text file with the password you want to encrypt. Don't leave any white space before or after the password.

   The following example shows the text `newpassword` in the input file `passfile`:

   ```
   $ cat passfile
   newpassword
   $
   ```

2. Use the `edb-repcli.jar` file to execute the Replication Server CLI with the `encrypt` command by first including the Java bin directory in your `PATH` environment variable and making `XDB_HOME/bin` your current working directory.

   For example, assuming `/usr/bin` contains the Java executable program and Replication Server is installed into the `POSTGRES_INSTALL_HOME` directory, perform the following:

   ```
   $ export PATH=/usr/bin:$PATH
   $ cd /opt/PostgresPlus/9.4AS/bin
   $ java -jar edb-repcli.jar -encrypt -input ~/passfile -output ~/encrypted
   The following shows the encrypted form of the password in the output file encrypted:
   $ cat ~/encrypted
   4mKq/4jQQoV2IypCSmPpTQ==
   $
   ```

3. Copy and paste the encrypted password into the Replication Server configuration file.

   ```
   #xDB Replication Server Configuration
   Properties
   #Thu Sep 03 11:13:27 GMT-05:00
   2015
   admin_password=4mKq/4jQQoV2IypCSmPpTQ==
   admin_user=admin
   ```

13.2.3 Writing a cron expression

A cron expression is a text string used to express a schedule of dates and times. The Linux cron tool uses a cron expression to schedule the execution of a job. Replication Server uses the Quartz job scheduling system for scheduling replications.

When creating a schedule for a Replication Server replication system, you can specify a cron expression. There are a number of formats for cron expressions. You must use the cron expression format supported by Quartz.

For a comprehensive treatment of cron expressions, refer to the Quartz documentation.

A Quartz cron expression consists of six mandatory fields followed by one optional field. Each field is separated from its neighbors by one or more consecutive space characters. The fields are order dependent and are listed as they must appear as follows:

```
ss mi hr dd mm dow [ yyyy ]
```

Field	Values	Description
ss	0 - 59	Second of the minute
mi	0 - 59	Minute of the hour
hr	0 - 23	Hour of the day
dd	1 - 31 or ?	Day of the month – if *dow* is given, then *dd* must be specified as ?
mm	1 - 12 or JAN - DEC	Month of the year (3-letter month abbreviations aren't case sensitive)
dow	1 – 7 or SUN – SAT or ?	Day of the week – if *dd* is given, then *dow* must be specified as ? (3-letter day of the week abbreviations are not case sensitive)
yyyy	1970 - 2099	Year – if omitted, then any year applies

A number of characters have special meaning that you can use in all fields unless noted.

Character	Meaning	
Character	Meaning	Example
,	Separates a list of values	MON,WED,FRI – Every Monday, Wednesday, and Friday
-	Separates the low and high end of a range of values	MON-FRI – Every Monday through Friday
*	Allows all legal values for the field	0 10 14 * * ? – Every day of every month at 2:10 PM
x/*i*	Specifies an increment, *i*, starting with *x*	0 0/10 * * * ? – Every 10 minutes starting on the hour for every day of every month (e.g., 8:00:00, 8:10:00, 8:20:00)
L	When used in the day of the month (dd) field, means the last day of the month	0 30 15 L 8 ? – Every August 31st at 3:30 PM
L	When used by itself in the day of the week field (dow), means Saturday	30 0 12 ? AUG L – The next Saturday in August at 30 seconds past 12:00 noon
*xxx*L	When used in the day of the week field (dow) following a day of the week, means the last *xxx* day of the month	30 0 12 ? AUG 6L – The last Friday in August at 30 seconds past 12:00 noon
*x*W	Used in the day of the month field (dd) following a day of the month, *x*, to specify the weekday closest to *x* without going over into the next or previous month.	1W – The weekday closest to the 1st of the month. If the 1st is a Wednesday, the result is Wednesday the 1st. If the 1st is a Sunday, the result is Monday the 2nd. If the 1st is a Saturday, the result is Monday the 3rd because the result does not go into the previous or following month.
xxx#*n*	Used in the day of the week field (dow) to specify the *n*th *xxx* day of the month	2#3 – The third Monday of the month (2 = Monday, 3 = third occurrence)

The following are some examples of cron expressions.

Cron expression	Meaning
Cron Expression	Meaning
0 0 12 20 AUG ? 2009	12:00:00 noon on August 20, 2009

Cron expression	Meaning
0 15 13 ? AUG WED	1:15:00 PM every Wednesday in August
30 30 8 ? * MON,WED,FRI	8:30:30 AM every Monday, Wednesday, and Friday of every month
0 0 8 ? * 2-6	8:00:00 AM Monday thru Friday of every month
0 0/30 8,9,10 15,L * ?	8:00:00 AM, 8:30:00 AM, 9:00:00 AM, 9:30:00 AM, 10:00:00 AM, 10:30:00 AM on the 15th and the last day of the month of every month
0 0 9 ? 9 L	9:00:00 AM each Saturday in September
0 0 1 ? * MonL	1:00:00 AM on the last Monday of the month of every month
0 30 16 15W sep ?	4:30:00 PM on the weekday of September closest to the 15th
0 30 16 ? * WED#2	4:30:00 PM on the second Wednesday of every month

13.2.4 Disabling foreign key constraints for snapshot replications

In a snapshot replication, the publication server calls EDB's Migration Toolkit, which disables foreign key constraints on tables so it can truncate the target tables before loading rows. In Postgres, foreign key constraints are implemented using triggers. So, in actuality, the Migration Toolkit disables triggers on the target tables by setting column `relhastriggers` of `pg_catalog.pg_class` to `false` for each target table.

No user (not even a superuser) is allowed to directly modify the data in a Postgres system catalog table unless the following conditions are satisfied:

- The database user attempting to modify the rows of a system catalog table is a superuser.
- In the Postgres system catalog table `pg_catalog.pg_authid` , the column `rolcatupdate` is set to `true` for the row identifying the superuser attempting to update a system catalog table. This requirement applies only to Postgres version 9.4 or earlier. The column `rolcatupdate` no longer exists in Postgres 9.5 or later.

To verify that a user has the privilege to update the system catalog tables, select the user name under the `Login Roles` node in pgAdmin (Postgres Enterprise Manager Client in Advanced Server). The **Update Catalogs** property must be set to **Yes**.

If the **Update Catalogs** property is set to **No**:

1. With the user name selected in the Object Browser, from the context menu, select **Properties**.
2. Select the **Role Privileges** tab.
3. Select **Can Modify Catalog Directly** and select **OK**.

13.2.5 Quoted identifiers and default case translation

A quoted identifier is an identifier created with its name enclosed in double quote characters `(")` . The text enclosed in double quotes is stored as the object identifier name exactly as given, with no default case translation of alphabetic characters. Quoted identifiers occur in both Oracle and Postgres.

For example, `CREATE TABLE "MyTable"` produces a table name that's stored in the database system's data dictionary as MyTable. References to this table must be made using an uppercase M, an uppercase T, and lowercase letters for the rest of the name.

If a database object is created without the double quotes surrounding its identifier name, default case translation of alphabetic characters occurs.

In Oracle, the default case translation is to upper case. For example, `CREATE TABLE MyTable` results in an object identifier name of `MYTABLE` .

In Postgres, the default case translation is to lower case. For example, `CREATE TABLE MyTable` results in an object identifier name of `mytable` .

13.2.6 Replicating the SQL server SQL_VARIANT data type

The `SQL_VARIANT` data type defines a column so that the individual values in that column can have different data types. For example, the same `SQL_VARIANT` column can store values that were explicitly cast as character, integer, numeric, and date/time.

However, if a table containing a `SQL_VARIANT` column is to be replicated to a Postgres database, the usage of the column in Postgres is restricted to a single data type to which all the values in the `SQL_VARIANT` column are implicitly convertible (that is, without the use of explicit casting). For example, an integer value is implicitly convertible to a `FLOAT` data type, but a floating point value is not implicitly convertible to an `INTEGER` data type.

The following restrictions apply to using replication on tables with the `SQL_VARIANT` data type:

- The values stored in the `SQL_VARIANT` columns of the table to be replicated must be implicitly convertible to the same data type in Postgres.
- If there's more than one table with `SQL_VARIANT` columns to be replicated to the same Postgres database, then all such `SQL_VARIANT` columns must contain values that are implicitly convertible to the same data type in Postgres.

In the Postgres subscription database, you define a domain named `sql_variant` that maps to an underlying data type to which all values in the `SQL_VARIANT` columns are implicitly convertible.

The following example shows how to set up replication for a table containing a `SQL_VARIANT` data type used to store numeric values, but of different data types. The SQL Server table definition is the following:

```
CREATE TABLE variant_tbl
(
    f1              INTEGER PRIMARY KEY,
    f2
SQL_VARIANT
);

INSERT INTO variant_tbl VALUES (1, CAST(1423.23 AS
NUMERIC(6,2)));
INSERT INTO variant_tbl VALUES (2, CAST(8001 AS
INTEGER));
INSERT INTO variant_tbl VALUES (3, CAST('4321' AS
CHAR(4)));
GO
```

The following query uses a function named `SQL_VARIANT_PROPERTY` to show the values stored in column `f2` and their data types.

```
 1> SELECT *,
2>     SQL_VARIANT_PROPERTY(f2,'BaseType') AS
basetype,
3>     SQL_VARIANT_PROPERTY(f2,'Precision') AS
precision,
4>     SQL_VARIANT_PROPERTY(f2,'Scale') AS
scale
5> FROM variant_tbl;
6> GO
f1          f2         basetype   precision
scale
----------- ---------- ---------- ---------- ---------
-
          1 1423.23    numeric    6          2
          2 8001       int        10         0
          3 4321       char       0          0

(3 rows affected)
```

In the Postgres subscription database, create a domain named `sql_variant` with an underlying data type compatible with the values that are stored in the SQL Server `SQL_VARIANT` column:

```
CREATE DOMAIN sql_variant AS NUMERIC(6,
2);
```

After replication occurs, the subscription table is created using the `sql_variant` domain in place of the `SQL_VARIANT` data type of the publication table.

13.3 Service pack maintenance

Maintenance items (bug fixes and enhancements) that were added to this version of Replication Server are listed below.

1. Registering your Replication Server product with an EnterpriseDB product license key is no longer required. Thus, all components related to registering the product were removed. The following are the removed components:

    - The `Product Registration` dialog box accessed from the EPRS Replication Console `Help` menu
    - The `license_key` parameter located in the EPRS Replication Configuration file
    - The Replication Server CLI `registerkey` command (`43230`)

2. Partitioned tables created using the declarative partitioning feature of PostgreSQL and EDB Postgres Advanced Server version 10 can now be replicated in a log-based single-master or multi-master replication system. (`43134`)

3. In an SMR system, removal of a table from a publication that has one or more existing subscriptions is now permitted. Previously, no tables from a publication in an SMR system could be removed if there are existing subscriptions. (`43110`)