Oracle® Database

Oracle GoldenGate Classic Architecture Documentation

(21c)
G18450-02
April 2025

Oracle Database Oracle GoldenGate Classic Architecture Documentation, (21c)

G18450-02

Primary Author: Oracle Corporation

Contents

Preface

1 Concepts

2 Install and Patch

3 Prepare

4 Manage

5 Extract

6 Replicat

7 Instantiate

8 Administer

9 Performance

10 Oracle GoldenGate Business Solutions

11 Autonomous Database

12 Upgrade

13 Appendix

Preface

The *Oracle GoldenGate Microservices Documentation* contains the Oracle GoldenGate Microservices concepts, tasks, advance tasks, security, and other reference information.

Audience

This guide is intended for system administrators and database users to learn about Oracle GoldenGate. It is assumed that readers are familiar with web technologies and have a general understanding of Windows and UNIX platforms.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Information

- Oracle GoldenGate Documentation
- Oracle GoldenGate Veridata

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, such as "From the File menu, select **Save**." Boldface also is used for terms defined in text or in the glossary.
italic *`italic`*	Italic type indicates placeholder variables for which you supply particular values, such as in the parameter statement: `TABLE` *`table_name`*. Italic type also is used for book titles and emphasis.
`monospace` `MONOSPACE`	Monospace type indicates code components such as user exits and scripts; the names of files and database objects; URL paths; and input and output text that appears on the screen. Uppercase monospace type is generally used to represent the names of Oracle GoldenGate parameters, commands, and user-configurable functions, as well as SQL commands and keywords.

Convention	Meaning
UPPERCASE	Uppercase in the regular text font indicates the name of a process or utility unless the name is intended to be a specific case. Keywords in upper case (`ADD EXTRACT`, `ADD EXTTRAIL`, `FORMAT RELEASE`).
LOWERCASE	Names of processes to be written in lower case. Examples: `ADD EXTRACT exte`, `ADD EXTRAIL ea`.
{ }	Braces within syntax enclose a set of options that are separated by pipe symbols, one of which must be selected, for example: {*option1* | *option2* | *option3*}.
[ ]	Brackets within syntax indicate an optional element. For example in this syntax, the `SAVE` clause is optional: `CLEANUP REPLICAT` *group_name* [, `SAVE` *count*]. Multiple options within an optional element are separated by a pipe symbol, for example: [*option1* | *option2*].
Sample Locations	Compass directions such as east, west, north, south to be used for demonstrating Extract and Replicat locations. Datacenters names to use the standard similar to `dc1`, `dc2`.
Group names	Prefixes for each process, as follows: • Extract: ext. Usage with location: `extn`, where ***n*** indicates 'north' compass direction. • Replicat: rep. Usage with location: `repn`, where ***n*** indicates 'north' compass direction. • Distribution Path: `dp`. Usage with location: `dpn`, where ***n*** indicates 'north' compass direction. • Checkpoint table: `ggs_checkpointtable` • Trail file names: e or d depending on whether the trail file is for the Extract or distribution path. Suffix derived in alphabetical order. Usage for an Extract trail file: `ea`, `eb`, `ec`. • Trail file subdirectory: The name will use compass directions to refer to the trail subdirectories. Example for trail subdirectory name would be `/east`, `/west`, `/north`, `/south`.

1
Concepts

Learn about the concepts of Oracle GoldenGate, its components, and Classic Architecture.

Oracle GoldenGate

Oracle GoldenGate is a comprehensive software package for real-time data integration and replication. It enables high availability solutions, real-time data integration, transactional change data capture, data replication, transformations, and verification between operational and analytical enterprise systems.

Using Oracle GoldenGate, you can move committed transactions across multiple systems in your enterprise. Oracle GoldenGate enables you to replicate data between Oracle databases to other supported Non-Oracle databases, and between Non-Oracle databases. In addition, you can replicate to Java Messaging Queues, Flat Files, and to Big Data targets in combination with Oracle GoldenGate for Big Data. To know more, see https://www.oracle.com/middleware/technologies/goldengate.html.

The product set enables high availability solutions, real-time data integration, transactional change data capture, data replication, transformations, and verification between operational and analytical enterprise systems. Oracle GoldenGate brings extreme performance with simplified configuration and management, support for cloud environments, expanded heterogeneity, and enhanced security.

This book is divided into parts so that you can easily find information that is relevant to your environment.

See *Installing Oracle GoldenGate* for system requirements and installation details for each of these databases.

See the *Using Oracle GoldenGate on Oracle Cloud Marketplace* to learn about provisioning and other configurations in the Oracle GoldenGate on Marketplace environment.

For Oracle GoldenGate 21c (21.1.0), you can use the following supported non-Oracle databases. This guide describes tasks related to these databases only.

- Db2 z/OS, Db2 for i, Db2 LUW
- MySQL
- SQL Server
- PostgreSQL
- Teradata
- TimesTen

For a full list of supported databases and variations, such as Amazon RDS and Azure database services, view the certification matrix available at:
https://www.oracle.com/middleware/technologies/fusion-certification.html
Each database that Oracle GoldenGate supports has its own requirements and configuration.

Parallel Replicat is supported by all databases available with Oracle GoldenGate 21c (21.1.0) and higher. This guide provides information about this feature for each database.

When Do You Use Oracle GoldenGate?

Oracle GoldenGate meets almost any data movement requirements you might have. Some of the most common use cases are described in this section.

You can use Oracle GoldenGate to meet the following business requirements:

Business Continuity and High Availability

Business Continuity is the ability of an enterprise to provide its functions and services without any lapse in its operations. High Availability is the highest possible level of fault tolerance. To achieve business continuity, systems are designed with multiple servers, storage, and data centers that provide high enough availability. To establish and maintain such an environment, data needs to be moved between these multiple servers and data centers, which is easily done using Oracle GoldenGate.

Consider a scenario where you are working in a multinational bank that has its headquarters in London, UK. You work in one of the banks' branches in Bangalore, India. This bank uses a specific account for its financial application that is used globally at all the branches. You have been asked by your manager to daily synchronize the transactions that have happened for this account in the database in the Bangalore branch with the centralized database situated at the UK. The volume of transactions is massive, and even the slightest delay can greatly impact the business. This same process is required at multiple destinations for every database in all the branches of the bank worldwide. This process has to be monitored continuously, preferably through some sort of GUI-based tool for the ease of management. Additionally, the bank has several other, non-critical applications used at all the branches. These applications are based on Non-Oracle databases, such as MySQL, but the transactions done over these databases also must be loaded into an Oracle Database located at the headquarters. The replication technology used must support both Oracle and Non-Oracle Databases so that they can talk to each other. Oracle GoldenGate is an apt solution in such a scenario.

Initial Load and Database Migration

Initial load is a process of extracting data records from a source database and loading those records onto a target database. Initial load is a data migration process that is performed only once. Oracle GoldenGate allows you to perform initial load data migrations without taking your systems offline.

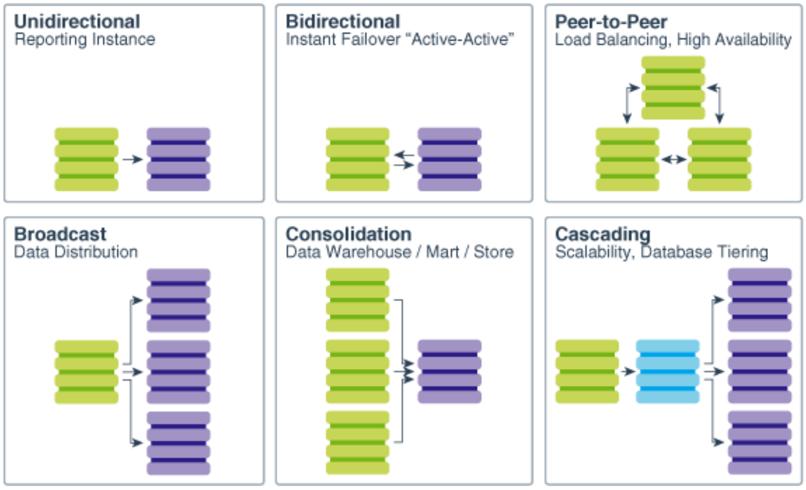
Data Integration

Data integration involves combining data from several disparate sources, which are stored using various technologies, and provide a unified view of the data. Oracle GoldenGate provides real-time data integration.

Topologies for Oracle GoldenGate

After installation, Oracle GoldenGate can be configured to meet your organization's business needs.

There are many different topologies that can be configured; which range from a simple unidirectional topology to the more complex peer-to-peer. No matter the architecture, Oracle GoldenGate provides similarities between them, making administration easier.



For full information about processing methodology, supported topologies and functionality, and configuration requirements, see the Oracle GoldenGate documentation for your database.

What is Oracle GoldenGate for Non-Oracle Databases?

Oracle GoldenGate is a comprehensive software package for real-time data capture and replication in non-Oracle IT environments.

The product set enables high availability solutions, real-time data integration, transactional change data capture, data replication, transformations, and verification between operational and analytical enterprise systems. Oracle GoldenGate brings extreme performance with simplified configuration and management, support for cloud environments, expanded heterogeneity, and enhanced security.

This book is divided into parts so that you can easily find information that is relevant to your environment.

See *Installing Oracle GoldenGate* for system requirements and installation details for each of these databases.

See the *Using Oracle GoldenGate on Oracle Cloud Marketplace* to learn about provisioning and other configurations in the Oracle GoldenGate on Marketplace environment.

What's New in This Guide

For Oracle GoldenGate 21c (21.1.0), you can use the following supported non-Oracle databases. This guide describes tasks related to these databases only.

- Db2 z/OS, Db2 for i, Db2 LUW
- MySQL
- SQL Server
- PostgreSQL
- Teradata
- TimesTen

For a full list of supported databases and variations, such as Amazon RDS and Azure database services, view the certification matrix available at:

https://www.oracle.com/middleware/technologies/fusion-certification.html
Each database that Oracle GoldenGate supports has its own requirements and configuration.

Parallel Replicat is supported by all databases available with Oracle GoldenGate 21c (21.1.0) and higher. This guide provides information about this feature for each database.

Oracle GoldenGate Product Family

There are numerous products in the Oracle GoldenGate product family.

- **Oracle GoldenGate Veridata** : Oracle GoldenGate Veridata compares one set of data to another and identifies data that is out-of-sync, and allows you to repair any data that is found out-of-sync.
- **Oracle GoldenGate Plug-in for EMCC**: The Enterprise Manager Plug-in for Oracle GoldenGate extends the Oracle Enterprise Manager Cloud Control and provides visual support for monitoring and managing Oracle GoldenGateprocesses.
- **Oracle GoldenGate Monitor**: Oracle GoldenGate Monitor is a real-time, Web-based monitoring console that delivers an at-a-glance, graphical view of all of the Oracle GoldenGate instances and their associated databases within your enterprise.
- **Oracle GoldenGate for Big Data**: Oracle GoldenGate for Big Data contains built-in support to write operation data from Oracle GoldenGate trail records into various Big Data targets (such as, HDFS, HBase, Kafka, Flume, JDBC, Cassandra, and MongoDB).
- **Oracle GoldenGate Application Adapters**: Oracle GoldenGate Application Adapters integrate with installations of the Oracle GoldenGatecore product to bring in Java Message Service (JMS) information or to deliver information as JMS messages or files.
- **Oracle GoldenGate for HP NonStop (Guardian)**: Oracle GoldenGate for HP NonStop enables you to manage business data at a transactional level by extracting and replicating selected data records and transactional changes across a variety of non-oracle applications and platforms.
- **Oracle GoldenGate Studio**: Oracle GoldenGate Studio enables you to design and deploy high-volume, real-time replication by automatically handling table and column mappings, allowing drag and drop custom mappings, generating best practice configurations from templates, and contains context sensitive help.

Getting Started with Oracle GoldenGate

Oracle GoldenGate supports two architectures, the Classic Architecture and the Microservices Architecture (MA).

Oracle GoldenGate can be configured for the following purposes:

- A static extraction of data records from one database and the loading of those records to another database.
- Continuous extraction and replication of transactional Data Manipulation Language (DML) operations and data definition language (DDL) changes (for supported databases) to keep source and target data consistent.
- Data extraction from supported database sources and replication to Big Data and file targets using Oracle GoldenGate for Big Data.

Oracle GoldenGate Architectures Overview

The following table describes the two Oracle GoldenGate architectures and when you should use each of the architectures.

X	Classic Architecture	Microservices Architecture
What is it?	Oracle GoldenGate Classic Architecture is the original architecture for enterprise replication. This architecture provides the processes and files required to effectively transfer transactional data across a variety of topologies. These processes and files form the main components of the classic architecture and was the main product installation method until the Oracle GoldenGate 12*c* (12.3.0.1) release.	Oracle GoldenGate Microservices Architecture is a microservices architecture that enables REST services as part of the Oracle GoldenGate environment. The REST-enabled services provide API end-points that can be leveraged for remote configuration, administration, and monitoring through web-based consoles, an enhanced command line interface, PL/SQL and scripting languages.

X	Classic Architecture	Microservices Architecture
When should I use it?	Oracle GoldenGate can be installed and configured to use the Oracle GoldenGate Classic Architecture only if MA release is not available for that platform, mentioned in the following scenarios: • A static extraction of data records from one database and the loading of those records to another database. • Continuous extraction and replication of transactional Data Manipulation Language (DML) operations and Data Definition Language (DDL) changes (for supported databases) to keep source and target data consistent. • Extraction from a database and replication to a file outside the database. • Capture from Non-Oracle Database sources.	Oracle GoldenGate can be installed and configured to use the Oracle GoldenGate Microservices Architecture for the following purposes: • Large scale and cloud deployments with fully-secure HTTPS interfaces and Secure WebSockets for streaming data. • Simpler management of multiple implementations of Oracle GoldenGate environments and control user access for the different aspects of Oracle GoldenGate setup and monitoring. • Support system managed database sharding to deliver fine-grained, multi-master replication where all shards are writable, and each shard can be partially replicated to other shards within a shardgroup. • Support the following features: – Thin and browser-based clients – Network security – User Authorization – Distributed deployments – Remote administration – Performance monitoring and orchestration – Coordination with other systems and services in an Oracle Database environment. – Custom embedding of Oracle GoldenGate into applications or to use secure, remote HTML5 applications.

X	Classic Architecture	Microservices Architecture
Which databases are supported?	Classic Architecture supports all supported databases as per the **certification matrix**.	MA only supports the Oracle database for an end-to-end MA-only topology. However, it is possible for a source Oracle GoldenGate Classic associated with Non-Oracle Databases to replicate to a target Oracle GoldenGate MA with Oracle, or a source Oracle GoldenGate MA with Oracle to replicate to a target Oracle GoldenGate legacy with Non-Oracle Databases.

Oracle GoldenGate Supported Processing Methods and Databases

Oracle GoldenGate enables the exchange and manipulation of data at the transaction level among multiple, Non-Oracle platforms across the enterprise. It moves committed transactions with transaction integrity and minimal overhead on your existing infrastructure. Its modular architecture gives you the flexibility to extract and replicate selected data records, transactional changes, and changes to DDL (data definition language) across a variety of topologies.

Note:

Support for DDL, certain topologies, and capture or delivery configurations varies by the database type.

Here is a list of the supported processing methods.

Database	Log-Based Extraction (capture)	Non-Log-Based Extraction [1] (capture)	Replication (delivery)
DB2 for i	N/A	N/A	X
DB2 LUW	X	N/A	X
DB2 z/OS	X	N/A	X
Oracle Database	X	N/A	X
MySQL	X	N/A	X
SQL Server	N/A	X	X
Terradata	N/A	N/A	X

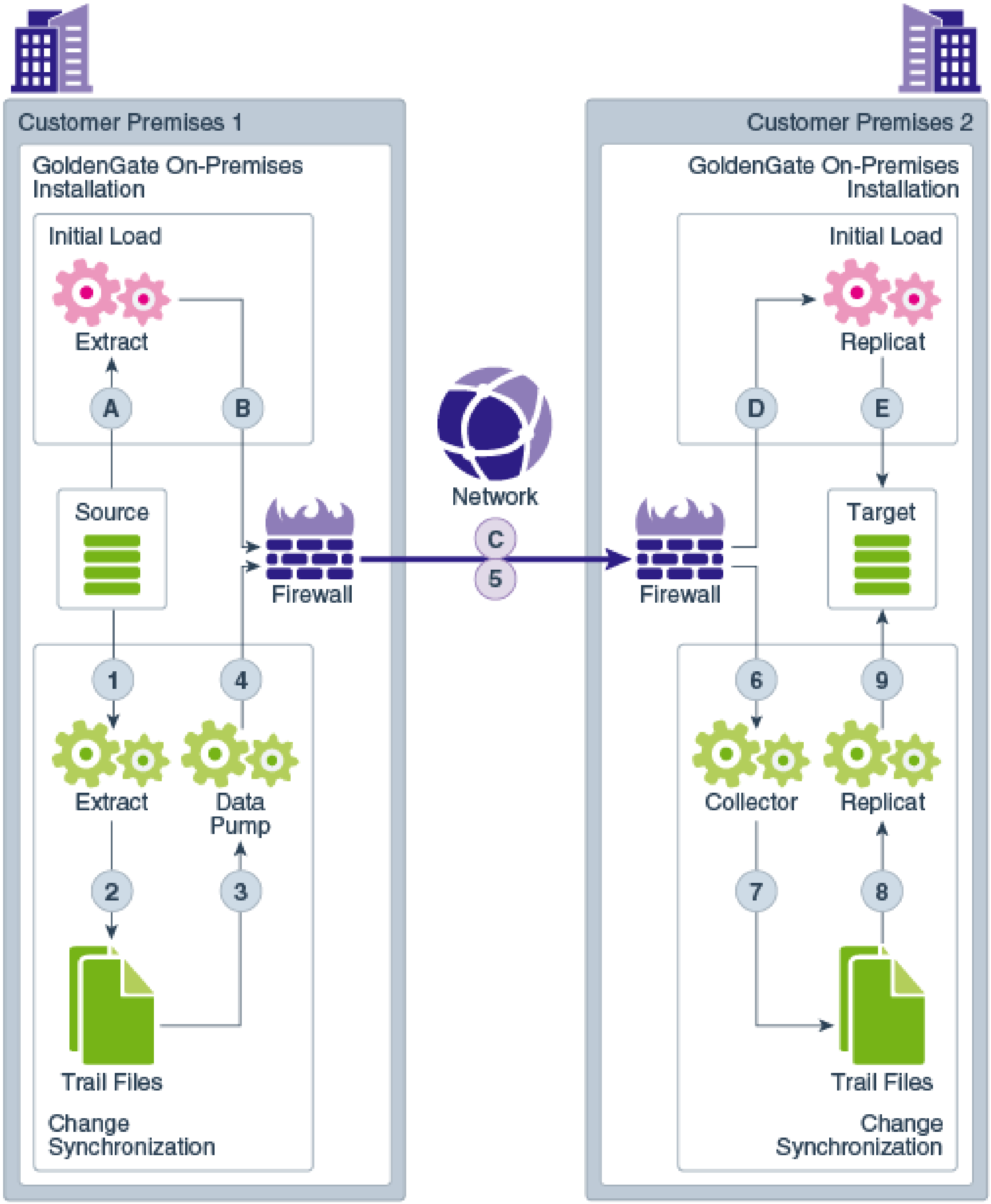
[1] Non-Log-Based Extraction uses a capture module that communicates with the Oracle GoldenGate API to send change data to Oracle GoldenGate.

Note:

Non-Log-Based Extraction uses a capture module that communicates with the Oracle GoldenGate API to send change data to Oracle GoldenGate.

Components of Oracle GoldenGate Classic Architecture

You can use the Oracle GoldenGate Classic Architecture to configure and manage your replications environments from the command line.



Note:

This is the basic configuration. Depending on your business needs and use case, you can configure different variations of this model.

What is a Manager?

Manager is the control process of Oracle GoldenGate. Manager must be running on each system in the Oracle GoldenGate configuration before the Extract or Replicat processes can be started.

Manager must also remain running while the Extract and Replicat processes are running so that resource management functions are performed. One Manager process can control many Extract or Replicat processes.

Manager performs the following functions:

- Starts Oracle GoldenGate processes
- Starts dynamic processes
- Maintains port numbers for processes
- Purges Trail files based on retention rules
- Creates event, error, and threshold reports

One Manager process can control many Extract or Replicat processes. On Windows systems, Manager can run as a service. See Configure the Manager Process for more information about the Manager process and configuring TCP/IP connections.

What is a Data Pump?

Data pump is a secondary Extract group within the source Oracle GoldenGate configuration.

If you configure a data pump, the Extract process writes all the captured operations to a trail file on the source database. The data pump reads the trail file on the source database and sends the data operations over the network to the remote trail file on the target database. Configuring a data pump is highly recommended for most configurations. If a data pump is not used, the Extract streams all the captured operations to a trail file on the remote target database. In a typical configuration with a data pump, however, the primary Extract group writes to a trail on the source system. The data pump reads this trail and sends the data operations over the network to a remote trail on the target. The data pump adds storage flexibility and also serves to isolate the primary Extract process from TCP/IP activity.

In general, a data pump can perform data filtering, mapping, and conversion

The data pump can be configured in two ways:

- Perform data manipulation: Data Pump can be configured to perform data filtering, mapping, and conversion.
- Perform no data manipulation: Data Pump can be configured in pass-through mode, where data is passively transferred as-is, without manipulation. Pass-through mode increases the throughput of the Data Pump, because all of the functionality that looks up object definitions is bypassed.

Though configuring a data pump is optional, Oracle recommends it for most configurations. Some reasons for using a data pump include the following:

- **Protection against network and target failures:** In a basic Oracle GoldenGate configuration, with only a trail on the target system, there is nowhere on the source system to store the data operations that Extract continuously extracts into memory. If the network or the target system becomes unavailable, Extract could run out of memory and abend. However, with a trail and data pump on the source system, captured data can be moved to disk, preventing the abend of the primary Extract. When connectivity is restored, the data pump captures the data from the source trail and sends it to the target system(s).

- **You are implementing several phases of data filtering or transformation.** When using complex filtering or data transformation configurations, you can configure a data pump to perform the first transformation either on the source system or on the target system, or even on an intermediary system, and then use another data pump or the Replicat group to perform the second transformation.
- **Consolidating data from many sources to a central target.** When synchronizing multiple source databases with a central target database, you can store extracted data operations on each source system and use data pumps on each of those systems to send the data to a trail on the target system. Dividing the storage load between the source and target systems reduces the need for massive amounts of space on the target system to accommodate data arriving from multiple sources.
- **Synchronizing one source with multiple targets.** When sending data to multiple target systems, you can configure data pumps on the source system for each target. If network connectivity to any of the targets fails, data can still be sent to the other targets.

What is a Collector?

The Collector is started by the manager process and is a process that runs in the background on the target system. It reassembles the transactional data into a target trail.

When the Manager receives a connection request from an Extract process, the Collector scans and binds to an available port and sends the port number to the Manager for assignment to the requesting Extract process. The Collector also receives the captured data that is sent by the Extract process and writes them to the remote trail file.

Collector is started automatically by the Manager when a network connection is required, so Oracle GoldenGate users do not interact with it. Collector can receive information from only one Extract process, so there is one Collector for each Extract that you use. Collector terminates when the associated Extract process terminates.

> **Note:**
>
> Collector can be run manually, if needed. This is known as a static Collector (as opposed to the regular, dynamic Collector). Several Extract processes can share one static Collector; however, a one-to-one ratio is optimal. A static Collector can be used to ensure that the process runs on a specific port.

By default, Extract initiates TCP/IP connections from the source system to Collector on the target, but Oracle GoldenGate can be configured so that Collector initiates connections from the target. Initiating connections from the target might be required if, for example, the target is in a trusted network zone, but the source is in a less trusted zone.

What is GGSCI?

You can use the Oracle GoldenGate Software Command Interface (GGSCI) commands to create data replications. This is the command interface between you and Oracle GoldenGate functional components.

To start either the Admin Client or GGSCI, you need to change the current working directory to the Oracle GoldenGate home directory (`OGG_HOME`).

> **Note:**
>
> The environment variable `OGG_HOME` and `OGG_VAR_HOME` must be set before starting the Admin Client or GGSCI.

Oracle GoldenGate Classic Architecture Programs and Utilities

This section describes programs installed in the Oracle GoldenGate installation directory.

> **Note:**
>
> Some programs may not exist in all installations. For example, if only capture or delivery is supported by Oracle GoldenGate for your platform, the Extract or Replicat program will not be installed, respectively.

Table 1-1 Oracle GoldenGate Installed Programs and Utilities

Program	Description
`convchk`	Converts checkpoint files to a newer release.
`convprm`	Converts parameter files that do not use SQL-92 rules for quoted names and literals to updated parameter files that use SQL-92 rules. SQL-92 format for quoted object names and literals was introduced as the default with the 12c release of Oracle GoldenGate.
`defgen`	Generates data definitions and is referenced by Oracle GoldenGate processes when source and target tables have dissimilar definitions.
`extract`	Performs capture from database tables or transaction logs or receives transaction data from a vendor access module.
`ggcmd`	Associated program of `ggsci`. Launches and monitors external applications, such as the JAgent of Oracle GoldenGate Monitor. Integrates those applications into the GGSCI environment.
`ggsci`	User interface to Oracle GoldenGate for issuing commands and managing parameter files.
`install`	Installs Oracle GoldenGate as a Windows service and provides other Windows-based service options.
`keygen`	Generates data-encryption keys.
`logdump`	A utility for viewing and saving information stored in extract trails or files.
`mgr`	(Manager) Control process for resource management, control and monitoring of Oracle GoldenGate processes, reporting, and routing of requests through the GGSCI interface.
`oggerr`	Manages Oracle GoldenGate error messages.
`replicat`	Applies data to target database tables.
`reverse`	A utility that reverses the order of transactional operations, so that Replicat can be used to back out changes from target tables, restoring them to a previous state.
`server`	The Collector process, an Extract TCP/IP server collector that writes data to remote trails.

Table 1-1 (Cont.) Oracle GoldenGate Installed Programs and Utilities

Program	Description
`vamserv`	Started by Extract to read the TMF audit trails generated by TMF-enabled applications. Installed to support the NonStop SQL/MX database.

Oracle GoldenGate Subdirectories

Learn about the subdirectories of the Oracle GoldenGate Classic Architecture installation directories; it does *not* apply to the Oracle GoldenGate Microservices Architecture.

Table 1-2 Oracle GoldenGate Classic Architecture Installed Subdirectories

Directory	Description
`br`	Contains the checkpoint files for the bounded recover feature.
`cfg`	Contains the property and XML files that are used to configure Oracle GoldenGate Monitor.
`dirdb`	Contains the data store that is used to persist information that is gathered from an Oracle GoldenGate instance for use by the Oracle GoldenGate Monitor application or within Oracle Enterprise Manager.

`dirchk`	Contains the checkpoint files created by Extract and Replicat processes, which store current read and write positions to support data accuracy and fault tolerance. Written in internal Oracle GoldenGate format.
	File name format is *`group_name+sequence_number.ext`* where *`sequence_number`* is a sequential number appended to aged files and *`ext`* is either `cpe` for Extract checkpoint files or `cpr` for Replicat checkpoint files.
	Do not edit these files.
	Examples:
	`ext1.cpe`
	`rep1.cpr`

`dircrd`	Contains credential store files.

`dirdat`	The default location for Oracle GoldenGate trail files and extract files that are created by Extract processes to store extracted data for further processing by the Replicat process or another application or utility. Written in internal Oracle GoldenGate format.
	File name format is a user-defined two-character prefix followed by either a 9-digit sequence number (trail files) or the user-defined name of the associated Extract process group (extract files).
	Do not edit these files.
	Examples:
	`rt000001`
	`finance`

Table 1-2 (Cont.) Oracle GoldenGate Classic Architecture Installed Subdirectories

Directory	Description
`dirdef`	The default location for data definitions files created by the `DEFGEN` utility to contain source or target data definitions used in a non-oracle synchronization environment. Written in external ASCII. File name format is a user-defined name specified in the `DEFGEN` parameter file. These files may be edited to add definitions for newly created tables. If you are unsure of how to edit a definitions file, contact Oracle GoldenGate technical support. Example: `defs.dat`
`dirdmp`	Contains trace, or dump, files that support the internal activity logging mechanism. This directory is only applicable to the Classic Architecture, see What is the Oracle GoldenGate Classic Architecture.
`dirjar`	Contains the Java executable files that support Oracle GoldenGate Monitor.

`dirpcs`	Default location for status files. File name format is *`group.extension`* where *`group`* is the name of the group and *`extension`* is either `pce` (Extract), `pcr` (Replicat), or `pcm` (Manager).
	These files are only created while a process is running. The file shows the program name, the process name, the port number, and the process ID.
	Do not edit these files.
	Examples:
	`mgr.pcm`
	`ext.pce`

`dirprm`	The default location for Oracle GoldenGate parameter files created by Oracle GoldenGate users to store run-time parameters for Oracle GoldenGate process groups or utilities. Written in external ASCII format. File name format is *`group name/user-defined name.prm`* or `mgr.prm` .
	These files may be edited to change Oracle GoldenGate parameter values after stopping the process. They can be edited directly from a text editor or by using the `EDIT PARAMS` command in GGSCI.
	Examples:
	`defgen.prm`
	`finance.prm`

`dirrec`	Not used by Oracle GoldenGate.

`dirrpt`	The default location for process report files created by Extract, Replicat, and Manager processes to report statistical information relating to a processing run. Written in external ASCII format.
	File name format is *`group name+sequence number.rpt`* where `sequence number` is a sequential number appended to aged files.
	Do not edit these files.
	Examples:
	`FIN2.rpt`
	`MGR4.rpt`

`dirsql`	Contains training scripts and any user-created SQL scripts that support Oracle GoldenGate.
`dirtmp`	The default location for storing transaction data when the size exceeds the memory size that is allocated for the cache manager. Do not edit these files.
`dirwlt`	Contains Oracle GoldenGate wallet files.

Table 1-2 (Cont.) Oracle GoldenGate Classic Architecture Installed Subdirectories

Directory	Description
`UserExitExamples`	Contains sample files to help with the creation of user exits.

Other Oracle GoldenGate Files

Learn about other files, templates, and objects created or installed in the `root` Oracle GoldenGate installation directory.

Table 1-3 Other Oracle GoldenGate Installed Files

Name	Description
`bcpfmt.tpl`	Template for use with Replicat when creating a run file for the Microsoft BCP/DTS bulk-load utility.
`bcrypt.txt`	Blowfish encryption software license agreement.
`cagent.dll`	Contains the Windows dynamic link library for the Oracle GoldenGate Monitor C sub-agent.
`category.dll`	Windows dynamic link library used by the `INSTALL` utility.
`chkpt_`*`db`*`_create.sql`	Script that creates a checkpoint table in the local database. A different script is installed for each database type.
`db2cntl.tpl`	Template for use with Replicat when creating a control file for the IBM `LOADUTIL` bulk-load utility.
`ddl_cleartrace.sql`	Script that removes the DDL trace file. (Oracle installations)
`ddl_ddl2file.sql`	Script that saves DDL from the marker table to a file.
`ddl_disable.sql`	Script that disables the Oracle GoldenGate DDL trigger.
`ddl_enable.sql`	Script that enables the Oracle GoldenGate DDL trigger.
`ddl_filter.sql`	Script that supports filtering of DDL by Oracle GoldenGate. This script runs programmatically; do not run it manually.
`ddl_nopurgeRecyclebin.sql`	Empty script file for use by Oracle GoldenGate support staff.
`ddl_ora11.sql` `ddl_ora12.sql`	Scripts that run programmatically as part of Oracle GoldenGate DDL support; do not run these scripts.
`ddl_pin.sql`	Script that pins DDL tracing, the DDL package, and the DDL trigger for performance improvements.
`ddl_purgeRecyclebin.sql`	Script that purges the Oracle recycle bin in support of the DDL replication feature.
`ddl_remove.sql`	Script that removes the DDL extraction trigger and package.
`ddl_session.sql`	Supports the installation of the Oracle DDL objects. This script runs programmatically; do not run it manually.
`ddl_setup.sql`	Script that installs the Oracle GoldenGate DDL extraction and replication objects.
`ddl_status.sql`	Script that verifies whether or not each object created by the Oracle GoldenGate DDL support feature exists and is functioning properly.

Table 1-3 (Cont.) Other Oracle GoldenGate Installed Files

Name	Description
`ddl_staymetadata_off.sql` `ddl_staymetadata_on.sql`	Scripts that control whether the Oracle DDL trigger collects metadata. This script runs programmatically; do not run it manually.
`ddl_trace_off.sql` `ddl_trace_on.sql`	Scripts that control whether DDL tracing is on or off.
`ddl_tracelevel.sql`	Script that sets the level of tracing for the DDL support feature.
`debug files`	Debug text files that may be present if tracing was turned on.

`demo_`*`db_scriptname`*`.sql`
`demo_`*`more_db_scriptname`*`.sql`
	Scripts that create and populate demonstration tables for use with tutorials and basic testing.

`.dmp files`	Dump files created by Oracle GoldenGate processes for tracing purposes.
`ENCKEYS`	User-created file that stores encryption keys. Written in external ASCII format.
`exitdemo.c`	User exit example.
`exitdemo_utf16.c`	User exit example that demonstrates how to use UTF16 encoded data in the callback structures for information exchanged between the user exit and the process.
`freeBSD.txt`	License agreement for FreeBSD.
`ggmessage.dat`	Data file that contains error, informational, and warning messages that are returned by the Oracle GoldenGate processes. The version of this file is checked upon process startup and must be identical to that of the process in order for the process to operate.
`ggserr.log`	File that logs processing events, messages, errors, and warnings generated by Oracle GoldenGate.
`ggsmsg.dll`	Windows dynamic link library used by the `install` program.
`GLOBALS`	User-created file that stores parameters applying to the Oracle GoldenGate instance as a whole.
`help.txt`	Help file for the GGSCI command interface.

`icudtxx.dll`
`icuinxx.dll`
`icuucxx.dll`
	Windows shared libraries for International Components for Unicode, where *`xx`* is the currently used version.

`jagent.bat`	Windows batch file for the JAgent for Oracle GoldenGate Monitor.
`jagent.log` `jagentjni.log`	Log files for the Oracle GoldenGate Monitor Agent.
`jagent.sh`	UNIX shell script for the JAgent for Oracle GoldenGate Monitor
`LGPL.txt`	Lesser General Public License statement. Applies to free libraries from the Free Software Foundation.
`libodbc.so`	ODBC file for Ingres 2.6 on UNIX.
`libodbc.txt`	License agreement for `libodbc.so`.

Table 1-3 (Cont.) Other Oracle GoldenGate Installed Files

Name	Description
`libxml2.dll`	Windows dynamic link library containing the XML library for the Oracle GoldenGate XML procedures.
`libxml2.txt`	License agreement for `libxml2.dll` .
`marker_remove.sql`	Script that removes the DDL marker table.
`marker_setup.sql`	Script that installs the Oracle GoldenGate DDL marker table.
`marker_status.sql`	Script that confirms successful installation of the DDL marker table.
`notices.txt`	Third-party software license file.
`odbcinst.ini`	Ingres 2.6 on UNIX ODBC configuration file.
`params.sql`	Script that contains configurable parameters for DDL support.
`ogg_cdc_cleanup_setup.bat`	Available for Oracle GoldenGate for SQL Server. Its used in creating the Oracle GoldenGate CDC Cleanup job for SQL Server.
`ogg_create_cdc_cleanup_job.sql`	Available for Oracle GoldenGate for SQL Server. Its used in creating the Oracle GoldenGate CDC Cleanup job for SQL Server.
`pthread-win32.txt`	License agreement for `pthread-VC.dll` .
`pthread-VC.dll`	POSIX threads library for Microsoft Windows.
`prvtclkm.plb`	Supports the replication of Oracle encrypted data.
`pw_agent_util.bat` `pw_agent_util.sh`	Script files that support the Oracle GoldenGate Monitor Agent.
`role_setup.sql`	Script that creates the database role necessary for Oracle GoldenGate DDL support.
`sampleodbc.ini`	Sample ODBC file for Ingres 2.6 on UNIX.
`sqlldr.tpl`	Template for use with Replicat when creating a control file for the Oracle SQL*Loader bulk-load utility.
`tcperrs`	File containing user-defined instructions for responding to TCP/IP errors.
`usrdecs.h`	Include file for user exit API.
`xerces-c_2_8.dll`	Apache XML parser library.
`zlib.txt`	License agreement for `zlib` compression library.

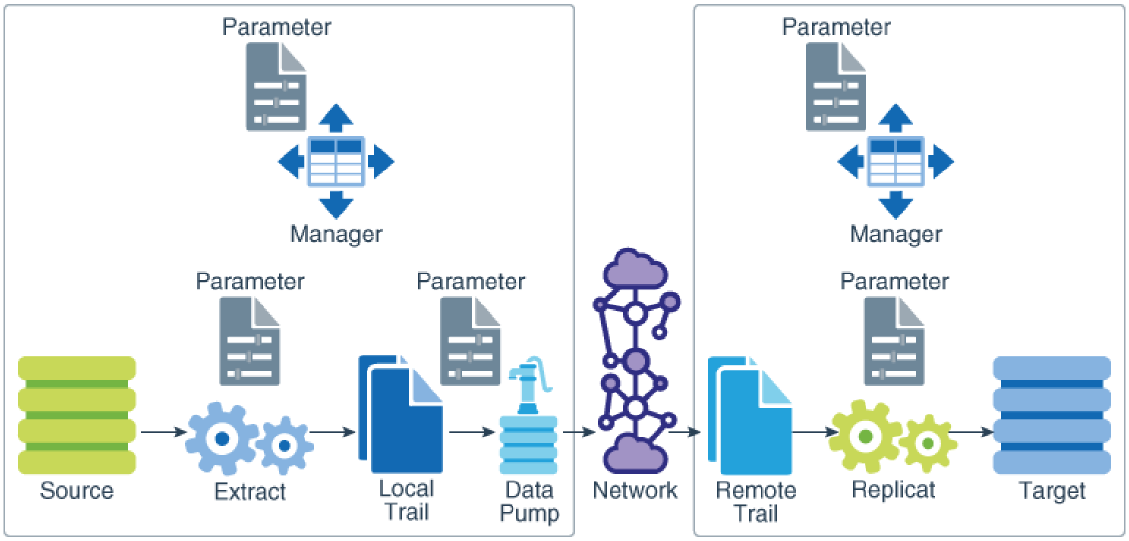
Overview of Oracle GoldenGate Processes

The Oracle GoldenGate capture process is known as *`Extract`*. Each instance of an Extract process is known as a *`group`*, which includes the process itself and the associated files that support it.

An additional Extract process, known as a *`data pump`*, is recommended to be used on the source system, so that captured data can be persisted locally to a series of files known as a *`trail`*. The data pump does not capture data but rather reads the local trail and propagates the data across the network to the target.

The Oracle GoldenGate apply process is known as *`Replicat`*. Each instance of a Replicat process is known as a *`group`*, which includes the process itself and the associated files that support it. Replicat reads data that is sent to local storage, known as a `trail`, and applies it to the target database.

The following diagram illustrates the basic Oracle GoldenGate process configuration.



> **Note:**
>
> Oracle Databases must be in `ARCHIVELOG` mode so that Extract can process the log files.

What is an Extract?

Extract is a process that is configured to run against the source database or configured to run on a downstream mining database (Oracle only) with capturing data generated in the true source database located somewhere else. This process is the extraction or the data capture mechanism of Oracle GoldenGate.

You can configure an Extract for the following use cases:

- **Initial Loads**: When you set up Oracle GoldenGate for initial loads, the Extract process captures the current, static set of data directly from the source objects.
- **Change Synchronization**: When you set up Oracle GoldenGate to keep the source data synchronized with another set of data, the Extract process captures the DML and DDL operations performed on the configured objects after the initial synchronization has taken place. Extracts can run locally on the same server as the database or on another server using the downstream Integrated Extract for reduced overhead. It stores these operations until it receives commit records or rollbacks for the transactions that contain them. If it receives a rollback, it discards the operations for that transaction. If it receives a commit, it persists the transaction to disk in a series of files called a trail, where it is queued for propagation to the target system. All the operations in each transaction are written to the trail as a sequentially organized transaction unit and are in the order in which they were committed to the database (commit sequence order). This design ensures both speed and data integrity.

> **Note:**
>
> Extract ignores operations on objects that are not in the Extract configuration, even though a transaction may also include operations on objects that are in the Extract configuration.

The Extract process can be configured to extract data from three types of data sources:

- Source tables: This source type is used for initial loads.
- Database recovery logs or transaction logs: While capturing from the logs, the actual method varies depending on the database type. An example of this source type is the Oracle Database redo logs.
- Third-party capture modules: This method provides a communication layer that passes data and metadata from an external API to the Extract API. The database vendor or a third-party vendor provides the components that extract the data operations and pass them to Extract.

What is a Trail?

A trail is a series of files on disk where Oracle GoldenGate stores the captured changes to support the continuous extraction and replication of database changes.

A trail can exist on the source system, an intermediary system, the target system, or any combination of those systems, depending on how you configure Oracle GoldenGate. On the local system, it is known as an Extract trail (or local trail). On a remote system, it is known as a remote trail. By using a trail for storage, Oracle GoldenGate supports data accuracy and fault tolerance. The use of a trail also allows extraction and replication activities to occur independently of each other. With these processes separated, you have more choices for how data is processed and delivered. For example, instead of extracting and replicating changes continuously, you could extract changes continuously and store them in the trail for replication to the target later, whenever the target application needs them.

In addition, trails allow Oracle Database to operate in non-oracle environment. The data is stored in a trail file in a consistent format, so it can be read by Replicat process for all supported databases. For more information , see About the Oracle GoldenGate Trail.

Processes that Write to the Trail File:

In Oracle GoldenGate Classic, the Extract and the data pump processes write to the trail. Only one Extract process can write to a given local trail. All local trails must have different full-path names though you can use the same trail names in different paths.

Multiple data pump processes can each write to a trail of the same name, but the physical trails themselves must reside on different remote systems, such as in a data-distribution topology. For example, a data pump named pumpm and a data pump named pumpn can both reside on sys01 and write to a remote trail named `aa`. Pumpm can write to trail aa on sys02, while pumpn can write to trail aa on sys03.

In Oracle GoldenGate MA, Distribution Server and distribution paths are used to write the remote trail.

Processes that Read from the Trail File:

The data pump, Replicat processes, and the Distribution Server read from the trail files. The data pump extracts DML and DDL operations from a local trail that is linked to an Extract process, performs further processing if needed, and transfers the data to a trail that is read by the next Oracle GoldenGate process downstream (typically Replicat, but could be another data pump if required). A Distribution Server process will read the trail file and send it across the network to a waiting Receiver Server process or collector.

The Replicat process reads the trail and applies the replicated DML and DDL operations to the target database.

Trail File Creation and Maintenance:

The trail files are created as needed during processing. You specify a two-character name for the trail when you add it to the Oracle GoldenGate configuration with the `ADD RMTTRAIL` or `ADD EXTTRAIL` command. By default, trails are stored in the `dirdat` sub-directory of the Oracle GoldenGate directory. You can specify a six or nine digit sequence number using the `TRAIL_SEQLEN_9D | TRAIL_SEQLEN_6D GLOBALS` parameter; `TRAIL_SEQLEN_9D` is set by default. It is recommended to use the 9-digit sequence number when possible.

As each new file is created, it inherits the two-character trail name appended with a unique nine digit sequence number from 000000000 through 999999999 (for example `c:\ggs\dirdat\tr000000001`). When the sequence number reaches 999,999,999 or 999,999 (depending on the prior setting) the Extract process will abend.

Refer to Doc ID 1060554.1 for details on how to reset the sequence number. Trail files can be purged on a routine basis by using the Manager parameter `PURGEOLDEXTRACTS`.

You can create more than one trail to separate the data from different objects or applications. You link the objects that are specified in a `TABLE` or `SEQUENCE` parameter to a trail that is specified with an `EXTTRAIL` or `RMTTRAIL` parameter in the Extract parameter file. To maximize throughput, and to minimize I/O load on the system, extracted data is sent into and out of a trail in large blocks. Transactional order is preserved.

Converting Existing Trails to 9 Digit Sequence Numbers

You can convert trail files from 6-digit to 9-digit checkpoint record for the named extract groups. Use `convchk` native command to convert to 9-digit trail by stopping your Extract gracefully then using `convchk` to upgrade as follows:

```
convchk extract trail seqlen_9d
```

Start your Extract

You can downgrade from a 9 to 6 digit trail with the same process using
this `convchk` command:

```
convchk extract trail seqlen_6d
```

> **Note:**
>
> **Extract Files**: You can configure Oracle GoldenGate to store extracted data in an extract file instead of a trail. The extract file can be a single file, or it can be configured to roll over into multiple files in anticipation of limitations on file size that are imposed by the operating system. It is similar to a trail, except that checkpoints are not recorded. The file or files are created automatically during the run. The same versioning features that apply to trails also apply to extract files.

What is a Replicat?

Replicat is a process that delivers data to a target database. It reads the trail file on the target database, reconstructs the DML or DDL operations, and applies them to the target database.

The Replicat process uses dynamic SQL to compile a SQL statement once and then executes it many times with different bind variables. You can configure the Replicat process so that it waits a specific amount of time before applying the replicated operations to the target database. For example, a delay may be desirable to prevent the propagation of errant SQL, to control data arrival across different time zones, or to allow time for other planned events to occur.

For the two common uses cases of Oracle GoldenGate, the function of the Replicat process is as follows:

- **Initial Loads**: When you set up Oracle GoldenGate for initial loads, the Replicat process applies a static data copy to target objects or routes the data to a high-speed bulk-load utility.
- **Change Synchronization**: When you set up Oracle GoldenGate to keep the target database synchronized with the source database, the Replicat process applies the source operations to the target objects using a native database interface or ODBC, depending on the database type.

You can configure multiple Replicat processes with one or more Extract processes and Data Pumps in parallel to increase throughput. To preserve data integrity, each set of processes handles a different set of objects. To differentiate among Replicat processes, you assign each one a group name

If you don't want to use multiple Replicat processes, you can configure a single Replicat process in parallel, coordinated, integrated mode.

- Parallel mode Parallel Replicat supports all databases using the non-integrated option. Parallel Replicat only supports replicating data from trails with full metadata, which requires the classic trail format. It takes into account dependencies between transactions, similar to Integrated Replicat. The dependency computation, parallelism of the mapping and apply is performed outside the database so can be off-loaded to another server. The transaction integrity is maintained in this process. In addition, parallel replicat supports the parallel apply of large transactions by splitting a large transaction into chunks and applying them in parallel. See About Parallel Replicat.
- Coordinated mode is supported on all databases that Oracle GoldenGate supports. In coordinated mode, the Replicat process is threaded. One coordinator thread spawns and coordinates one or more threads that execute replicated SQL operations in parallel. A coordinated Replicat process uses one parameter file and is monitored and managed as one unit. See About Coordinated Replicat Mode for more information.
- Integrated mode is supported for Oracle Database releases 11.2.0.4 or later. In integrated mode, the Replicat process leverages the apply processing functionality that is available within the Oracle Database. Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity. See About Integrated Replicat for more information about integrated mode.

You can delay Replicat so that it waits a specific amount of time before applying the replicated operations to the target database. A delay may be desirable, for example, to prevent the propagation of errant SQL, to control data arrival across different time zones, or to allow time for other planned events to occur. The length of the delay is controlled by the `DEFERAPPLYINTERVAL` parameter.

Oracle GoldenGate Processes and Key Terms

Oracle GoldenGate has common data replication processes and architecture-specific processes as well.

Specific components of Classic Architecture are discussed in Components of Oracle GoldenGate Classic Architecture.

Oracle GoldenGate Key Terms and Concepts

Apart from the two architectures and their components, there are some key terms that you should get familiar with.

About Process Types

Depending on the requirement, Oracle GoldenGate can be configured with the following processing types.

- An *online* Extract or Replicat process that runs until stopped by a user. Online processes maintain recovery checkpoints so that processing can resume after interruptions. You use online processes to continuously extract and replicate DML and DDL operations (where supported) to keep source and target objects synchronized. The `EXTRACT` and `REPLICAT` parameters apply to this process type.
- A *source-is-table* (also known as in *initial-load* Extract) Extract process extracts a current set of static data directly from the source objects in preparation for an initial load to another database. This process type does not use checkpoints. The `SOURCEISTABLE` parameter applies to this process type.
- A *special-run* Replicat process applies data within known begin and end points. You use a special-run Replicat for initial data loads, and it also can be used with an online Extract to apply data changes from the trail in batches, such as once a day rather than continuously. This process type does not maintain checkpoints because the run can be started over with the same begin and end points. The `SPECIALRUN` parameter applies to this process type.
- A *remote task* is a special type of initial-load process in which Extract communicates directly with Replicat over TCP/IP. Neither a Collector process nor temporary disk storage in a trail or file is used. The task is defined in the Extract parameter file with the `RMTTASK` parameter.

About Commit Sequence Number (CSN)

When working with Oracle GoldenGate, you might need to refer to a *Commit Sequence Number*, or CSN. A CSN is an identifier that Oracle GoldenGate constructs to identify a transaction for the purpose of maintaining transactional consistency and data integrity. It uniquely identifies a point in time in which a transaction commits to the database.

The CSN can be required to position Extract in the transaction log, to reposition Replicat in the trail, or for other purposes. It is returned by some conversion functions and is included in reports and certain command line output.

A CSN is a monotonically increasing identifier generated by Oracle GoldenGate that uniquely identifies a point in time when a transaction commits to the database. It purpose is to ensure transactional consistency and data integrity as transactions are replicated from source to target. Each kind of database management system generates some kind of unique serial number of its own at the completion of each transaction, which uniquely identifies the commit of that transaction. For example, the Oracle RDBMS generates a System Change Number, which is a monotonically increasing sequence number assigned to every event by Oracle RDBMS. The CSN captures this same identifying information and represents it internally as a series of bytes, but the CSN is processed in a platform-independent manner. A comparison of any two CSN numbers, each of which is bound to a transaction-commit record in the same log stream, reliably indicates the order in which the two transactions completed.

The CSN is cross-checked with the transaction ID (displayed as XID in Oracle GoldenGate informational output). The XID-CSN combination uniquely identifies a transaction even in cases where there are multiple transactions that commit at the same time, and thus have the same CSN. For example, this can happen in an Oracle RAC environment, where there is parallelism and high transaction concurrency.

The CSN value is stored as a token in any trail record that identifies the commit of a transaction. This value can be retrieved with the `@GETENV` column conversion function and viewed with the Logdump utility.

See *Administering Oracle GoldenGate* for more information about the CSN and a list of CSN values per database.

Overview of Groups

To differentiate among multiple Extract or Replicat processes on a system, you can create processing *groups*. For example, to replicate different sets of data, you would create two Replicat groups.

A processing group consists of a process (either Extract or Replicat), its parameter file, its checkpoint file, and any other files associated with the process. For Replicat, a group may also include an associated checkpoint table. You define groups by using the `ADD EXTRACT` and `ADD REPLICAT` commands in the Oracle GoldenGate command interface.

All files and checkpoints relating to a group share the name that is assigned to the group itself. Any time that you issue a command to control or view processing, you supply a group name or multiple group names by means of a wildcard.

2

Install and Patch

Learn about installation prerequisites for Oracle GoldenGate, steps to install Oracle GoldenGate for different databases, post-installation tasks, installing patches, and uninstalling Oracle GoldenGate.

Obtaining the Oracle GoldenGate Distribution

You can download Oracle GoldenGate from the Oracle Software Delivery Cloud site at:

https://edelivery.oracle.com/osdc/faces/SoftwareDelivery

This site hosts the first Generally Available (GA) releases of Oracle GoldenGate software versions and there may be a more recent patch version available on https://support.oracle.com.

It is extremely important to ensure that your Oracle GoldenGate installation is running the most recent release of the Oracle GoldenGate software. Oracle always recommends running the latest bundle patch for all Oracle GoldenGate products and this is crucial if you are installing GoldenGate for a new project or if you are upgrading an existing environment to the latest bundle patch.

For Oracle GoldenGate for the Oracle Database, both Classic Architecture and Microservices Architecture patches require the GA release to be installed first, followed by the patch to apply, unless there is a patch listed with a description of ‘Complete Install’ in which case that patch release can be installed directly without having to first install the GA release version.

In cases of Oracle GoldenGate Classic Architecture software for non-Oracle databases such as PostgreSQL or MySQL, any patch posted on https://support.oracle.com is a complete build which can be installed directly without having to apply the GA release first.

A list of the latest patches (both complete installs and incremental patches) for all GoldenGate for Oracle, non-Oracle, and Mainframe databases can be found on https://support.oracle.com, Knowledge Document - Primary Note for Oracle GoldenGate Core Product Patch Sets (Doc ID 1645495.1)

Verify Certification and System Requirements

Ensure that Oracle GoldenGate is installed on supported hardware and operating systems. For more information, see the Certification Matrix for the release.

Oracle tests and verifies the performance of your product on all certified systems and environments. As new certifications occur, they are added to the proper certification document. New certifications can occur at any time, and for this reason the certification documents are kept outside of the documentation libraries and are available on Oracle Technology Network.

Here are some additional details about the supported platforms:

- Cross Endian Support: Most Oracle GoldenGate products support cross endian replication, which means that the source and target database can be a different platform (or even endian) than the actual server where Oracle GoldenGate is installed.

- Fully Certified Criteria: Oracle GoldenGate certifications are often phased in, for a particular new release of the product. Oracle typically supports Oracle databases first and then the various non-Oracle and Big Data technologies. In some cases, Oracle GoldenGate may support the data store you are looking for, but you may need to check the certification matrix for a previous release. Platforms that are in the certification matrix are platforms where either full regression testing is done or where basic validation is performed for continuity purposes.
- Fully Supported by Inference: There are other technologies that are supported for Oracle GoldenGate that may not be explicitly listed in the certification matrix. For example, Oracle certify its technologies based on a combination of Chipset, Operating System, Data Store Type, and Data Store Version. As long as these four criteria are met, support is available.
- Fully Supported through Open Source Compatibility: There are a number of Open Source technologies that Oracle GoldenGate is certified with such as Big Data and non-Oracle databases. Sometimes, users may have open source environments and need Oracle GoldenGate to provide support with such unique infrastructures, such as Apache HBase on Azure Data Lake. In such cases, Oracle GoldenGate does support any unique open source environment if the Chipset, Operating System, Open Source Framework and Framework Version are certified by Oracle GoldenGate. For example, in case of Apache HBase, Oracle GoldenGate support needs to check the version of Apache HBase, for which Oracle GoldenGate is certified, and if that version happens to be running on some Cloud, then Oracle GoldenGate will be supported. In each of these Open Source examples (that are not explicitly certified), Oracle GoldenGate support is available using the base open source configurations, such as Apache on certified hardware. However, Oracle may not be obligated to support each possible infrastructure combination that users may select.
- Java JDBC Support: Many SQL, NoSQL and Big Data technologies support Java JDBC capabilities. Oracle GoldenGate for Distributed Applications and Analytics enables replication of transactions into any JDBC compliant drivers. Individual drivers may vary in terms of performance and metadata coverage, so there is no specific guarantee that Oracle GoldenGate JDBC support will work with every JDBC driver, but most common JDBC drivers and commercial implementations usually work with Oracle GoldenGate JDBC and these are supported. If you don’t find your technology in the certification matrix, but you know that there is a JDBC drive available, then it could be that you may have both technical compatibility and a supported configuration.
- Managed and Unmanaged Data Stores: With the advent of managed Cloud services such as native cloud services, many data stores are now available with automated lifecycle, patching, and other conveniences. In many cases, managed data stores are fully compatible and consistent with Oracle GoldenGate certifications and support. However, in some cases, a cloud vendor may turn-off or restrict access to features that Oracle GoldenGate requires for full features compatibility, particularly with Oracle GoldenGate Extract capabilities. If you have a question about a third party cloud managed service for a data store that Oracle GoldenGate may usually support, but you do not see that managed service listed in the Oracle GoldenGate certification matrix, directly contact Oracle GoldenGate product management.

Operating System Requirements

This section outlines the operating system resources that are necessary to support Oracle GoldenGate.

Memory Requirements

All Platforms

The amount of memory that is required for Oracle GoldenGate depends on the amount of data being processed, the number of Oracle GoldenGate processes running, the amount of RAM available to Oracle GoldenGate, and the amount of disk space that is available to Oracle GoldenGate for storing pages of RAM temporarily on disk when the operating system needs to free up RAM (typically when a low watermark is reached). This temporary storage of RAM to disk is commonly known as **swapping** or **paging** (herein referred to as swapping). Depending on the platform, the term *swap space* can be a swap partition, a swap file, a page file (Windows) or a shared memory segment (IBM for i).

Modern servers have sufficient RAM combined with sufficient swap space and memory management systems to run Oracle GoldenGate. However, increasing the amount of RAM available to Oracle GoldenGate may significantly improve its performance, as well as that of the system in general.

Typical Oracle GoldenGate installations provide RAM in multiples of gigabytes to prevent excessive swapping of RAM pages to disk. The more contention there is for RAM the more swap space that is used.

Excessive swapping to disk causes performance issues for the Extract process in particular, because it must store data from each open transaction until a commit record is received. If Oracle GoldenGate runs on the same system as the database, then the amount of RAM that is available becomes critical to the performance of both.

RAM and swap usage are controlled by the operating system, not the Oracle GoldenGate processes. The Oracle GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that the Oracle GoldenGate processes work in a sustained and efficient manner. In most cases, users need not change the default Oracle GoldenGate memory management configuration.

For more information about evaluating Oracle GoldenGate memory requirements, see the `CACHEMGR` parameter in the *Parameters and Functions Reference for Oracle GoldenGate*.

> **Note:**
>
> **On the DB2 for i host system** allocate approximately 10-50 MB of memory for each Oracle GoldenGate journal reader.

Windows Platforms

For Windows Server environments, the number of process groups that can be run are tightly coupled to the *non-interactive* Windows desktop heap memory settings. The default settings for Windows desktop heap may be enough to run very small numbers of process groups. As you approach larger amounts of process groups, more than 60 or so, you have two choices:

- Adjust the non-interactive value of the SharedSection field in the registry based on information from Microsoft (Windows desktop heap memory).
- Increase the number of Oracle GoldenGate homes and spread the total number of desired process groups across these homes.

For more information on modifying the Windows Desktop Heap memory, review the following Oracle Knowledge Base document (Doc ID 2056225.1).

Oracle GoldenGate for DB2 for i

Oracle GoldenGate for Db2 for i requires the following memory resources on the remote system, and the database host system.

Memory Requirements

Oracle GoldenGate requires the following memory resources on the Oracle GoldenGate remote system and the database host system.

On the remote system
The amount of memory that is required for Oracle GoldenGate depends on the amount of data being processed, the number of Oracle GoldenGate processes running, the amount of RAM available to Oracle GoldenGate, and the amount of disk space that is available to Oracle GoldenGate for storing pages of RAM temporarily on disk when the operating system needs to free up RAM (typically when a low watermark is reached). This temporary storage of RAM to disk is commonly known as **swapping** or **paging**. Depending on the platform, the term **swap space** can be a swap partition, a swap file, or a shared memory segment (IBM i platforms).
Modern servers have sufficient RAM combined with sufficient swap space and memory management systems to run Oracle GoldenGate. However, increasing the amount of RAM available to Oracle GoldenGate may significantly improve its performance, as well as that of the system in general.
Typical Oracle GoldenGate installations provide RAM in multiples of gigabytes to prevent excessive swapping of RAM pages to disk. The more contention there is for RAM the more swap space that is used.
Excessive swapping to disk causes performance issues for the Extract process in particular, because it must store data from each open transaction until a commit record is received. If Oracle GoldenGate runs on the same system as the database, the amount of RAM that is available becomes critical to the performance of both.
RAM and swap usage are controlled by the operating system, not the Oracle GoldenGate processes. The Oracle GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that the Oracle GoldenGate processes work in a sustained and efficient manner. In most cases, users need not change the default Oracle GoldenGate memory management configuration.
For more information about evaluating Oracle GoldenGate memory requirements, see the `CACHEMGR` parameter in the *Parameters and Functions Reference for Oracle GoldenGate*.

On the DB2 host system
Allocate approximately 10-50 MB of virtual memory for each Oracle GoldenGate log reader, `oggreadb`, that is invoked depending on the size of the log buffer. There is one invocation per Extract process on the remote system. To adjust the maximum log buffer size, use the `TRANLOGOPTIONS BUFSIZE` parameter in the Extract parameter file.
When setting up the Workload Manager (WLM) environment for the Extract log read components, it is recommended to set `NUMTCB` in the range of 10-40 depending on your environment. Refer to the IBM documentation for more information.

Disk Requirements

Disk space requirements vary based on the platform, database, and Oracle GoldenGate architecture to be installed.

Disk Requirements for Oracle GoldenGate Installation Files

The disk space requirements for a Oracle GoldenGate installation vary based on your operating system and database. Ensure that you have adequate disk space for the downloaded file, expanded files, and installed files, which can be up to 2GB.

To determine the size of the Oracle GoldenGate download file, view the Size column before downloading your selected build from Oracle Software Delivery Cloud. The value shown is the size of the files in compressed form. The size of the expanded Oracle GoldenGate installation directory is significantly larger on disk.

Temporary Disk Requirements

By default, the Oracle GoldenGate Classic Architecture maintains data that it writes to disk in the `dirtmp` sub-directory of the Oracle GoldenGate installation directory. When total cached transaction data exceeds the `CACHESIZE` setting of the `CACHEMGR` parameter, Extract will begin writing cache data to temporary files. The cache manager assumes that all of the free space on the file system is available. This directory can fill up quickly if there is a large transaction volume with large transaction sizes. To prevent I/O contention and possible disk-related Extract failures, dedicate a disk to this directory. You can assign a name to this directory with the `CACHEDIRECTORY` option of the `CACHEMGR` parameter.

> **Note:**
>
> `CACHEMGR` is an internally self-configuring and self-adjusting parameter. It is rare that this parameter requires modification. Doing so unnecessarily may result in performance degradation. It is best to acquire empirical evidence before opening an Oracle Service Request and consulting with Oracle Support.

It is typically more efficient for the operating system to swap to disk than it is for Extract to write temporary files. The default `CACHESIZE` setting assumes this. Thus, there should be sufficient disk space to account for this, because only after the value for `CACHESIZE` is exceeded Extract will write transaction cached data to temporary files in the file system name space. If multiple Extract processes are running on a system, the disk requirements can multiply. Oracle GoldenGate writes to disk when there is not enough memory to store an open transaction. Once the transaction has been committed or rolled back, committed data is written to trail files and the data are released from memory and Oracle GoldenGate no longer keeps track of that transaction. There are no minimum disk requirements because when transactions are committed after every single operation these transactions are never written to disk.

> **Important:**
>
> Oracle recommends that you do not change the `CACHESIZE` because performance can be adversely effected depending on your environment.

Other Disk Space Considerations

In addition to the disk space required for the files and binaries that are installed by Oracle GoldenGate, allow additional disk space to hold the Oracle GoldenGate trails. Trails can be created up to 2GB in size, with a default of 500MB. The space required depends upon the selected size of the trails, the amount of data being captured for replication, and how long the consumed trails are kept on the disk. The recommended minimum disk allocated for Trails may be computed as:

((transaction log size * 0.33) * number of log switches per day) * number of days to retain trails

Based on this equation, if the transaction logs are 1GB in size and there is an average of 10 log switches per day, it means that Oracle GoldenGate will capture 3.3GB data per day. To be able to retain trails for 7 days, the minimum amount of disk space needed to hold the trails is 23GB.

A trail is a set of self-aging files that contain the working data at rest and during processing. You may need more or less than this amount, because the space that is consumed by the trails depends on the volume of data that will be processed.

Disk Requirements for DB2 z/OS

On the DB2 host system
(Only applicable if you are installing stored procedures.) Assign a zFS (zSeries file systems) or hierarchical file system volume. To determine the size of the Oracle GoldenGate download file, examine the size of `zOSPrograms.zip` on the remote DB2 system after extracting the installation image.

Network

The following network resources must be available to support Oracle GoldenGate Classic Architecture.

- For optimal performance and reliability, especially in maintaining low latency on the target, use the fastest network possible and install redundancies at all points of failure.
- Configure the system to use both TCP and UDP services, including DNS. Oracle GoldenGate supports IPv4 and IPv6 and can operate in a system that supports one or both of these protocols.
- Configure the network with the host names or IP addresses of all systems that will be hosting Oracle GoldenGate processes and to which Oracle GoldenGate will be connecting.
- Oracle GoldenGate requires some unreserved and unrestricted TCP/IP network ports, the number of which depends on the number and types of processes in your configuration. See Configure Network Communications.
- Keep a record of the ports that you assigned to Oracle GoldenGate processes. You specify them with parameters when configuring deployments for the Microservices Architecture and for the Manager and pumps with the Classic Architecture.
- Configure your firewalls to accept connections through the Oracle GoldenGate ports.

Operating System Privileges

The following are the privileges in the operating system that are required to install Oracle GoldenGate and to run the processes:

- The person who installs Oracle GoldenGate must be granted read and write privileges on the Oracle GoldenGate software home directory.
- To install on Windows, the person who installs Oracle GoldenGate must log in as an Administrator.

- The Oracle GoldenGate Extract, Replicat, and Manager processes, and configuring deployments using the `oggca.sh` script must operate as an operating system user that has read, write, and delete privileges on files and subdirectories in the Oracle GoldenGate directory. In addition, the `oggca.sh` process requires privileges to control the other Oracle GoldenGate processes.
- Oracle recommends that you dedicate the Extract and Replicat operating system users to Oracle GoldenGate. Sensitive information might be available to anyone who runs an Oracle GoldenGate process, depending on how database authentication is configured.

Operating System Privileges for DB2 z/OS

The remote host requires privileges to use the `chmod +rw` command on the sub-directories in the Oracle GoldenGate product directory.

Table 2-1 Operating System Privileges

DB2 z/OS User Privilege	Extract	Stored Procedures	Replicat
`CONNECT` to the remote DB2 subsystem.	X	X	X

Operating System Privileges

The following are the privileges in the operating system that are required to install Oracle GoldenGate and to run the processes.

- To install on Windows, the person who installs Oracle GoldenGate must log in as Administrator.
- To install on UNIX, the person who installs Oracle GoldenGate must have read and write privileges on the Oracle GoldenGate installation directory.
- The Oracle GoldenGate Extract, Replicat, and Manager processes must operate as an operating system user that has privileges to read, write, and delete files and subdirectories in the Oracle GoldenGate directory. In addition, the Manager process requires privileges to control the other Oracle GoldenGate processes.
- The Extract process must operate as an operating system user that has read access to the transaction log files, both online and archived. If you install the Manager process as a Windows service during the installation steps in this documentation, you must install as Administrator for the correct permissions to be assigned. If you cannot install Manager as a service, assign read access to the Extract process manually, and then always run Manager and Extract as Administrator.
- Dedicate the Extract, Replicat, and Manager operating system users to Oracle GoldenGate. Sensitive information might be available to anyone who runs an Oracle GoldenGate process.

Operating System Privileges for Teradata

The Manager process requires an operating system user that has privileges to control Oracle GoldenGate processes and to read, write, and purge files and subdirectories in the Oracle GoldenGate directory. The Replicat processes require privileges to access the database.

Operating System Privileges

The operating system privileges for using Oracle GoldenGate for Oracle TimesTen are:

- You need read and write privileges on the Oracle GoldenGate installation directory.
- Oracle GoldenGate Replicat and Manager processes must operate as an operating system user that has privileges to read, write, and delete files and subdirectories in the Oracle GoldenGate directory. In addition, the Manager process requires privileges to control all other Oracle GoldenGate processes.
- Dedicate the Replicat and Manager operating system users to Oracle GoldenGate to avoid access to sensitive information to other users who run Oracle GoldenGate processes.

Other Operating System Requirements

The following additional features of the operating system must be available to support Oracle GoldenGate.

- To use Oracle GoldenGate user exits, install the C/C++ Compiler, which creates the programs in the required shared object or DLL.
- Gzip to decompress the Oracle GoldenGate installation files. Otherwise, you must unzip the installation on a PC by using a Windows-based product, and then FTP it to the AIX, DB2 for i, or DB2 z/OS platforms.
- For best results on DB2 platforms, apply high impact (HIPER) maintenance on a regular basis staying within one year of the current maintenance release. The HIPER process identifies defects that could affect data availability or integrity. IBM provides Program Temporary Fixes (PTF) to correct defects found in DB2 for i and DB2 z/OS.
- Oracle GoldenGate for SQL Server when installed on Linux requires the `libnsl` and `unixODBC` packages to be installed prior to launching GGSCI.
- Before installing Oracle GoldenGate on a Windows system, install the Microsoft Visual C ++ 2013 Redistributable Package and the Microsoft Visual C++ 2017 Redistributable Package. These packages install runtime components of Visual C++ Libraries that are required for Oracle GoldenGate processes.

  Download and install the x64 version of Visual C++ 2013 package from :

  https://support.microsoft.com/en-us/help/4032938/update-for-visual-c-2013-redistributable-package

  Download and install the x64 version of Visual C++ 2017 package from

  https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads
- For Oracle GoldenGate for Oracle to be installed on a remote hub server, download and install the Oracle Database 19c client for the operating system platform where Oracle GoldenGate will be installed and ensure that you install the Administrator version of the client.

Security and Other Considerations

Oracle GoldenGate fully supports virtual machine environments created with any virtualization software on any platform unless otherwise noted. When installing Oracle GoldenGate into a virtual machine environment, select a build that matches the database and the operating system of the virtual machine, not the host system.

> **Note:**
>
> Oracle customers with an active support contract and running supported versions of Oracle products (including Oracle GoldenGate) receive assistance from Oracle when running those products on VMware virtualized environments.
>
> If Oracle identifies the underlying issue is not caused by Oracle’s products or is being run in a computing environment not supported by Oracle, Oracle will refer customers to VMware for further assistance and Oracle will provide assistance to VMware as applicable in resolving the issue.
>
> This support policy does not affect Oracle or VMware licensing policies.

Windows Console Character Sets

The operating system and the command console must have the same character sets. Mismatches occur on Microsoft Windows systems, where the operating system is set to one character set, but the DOS command prompt uses a different, older DOS character set. Oracle GoldenGate uses the character set of the operating system to send information to GGSCI command output; therefore a non-matching console character set causes characters not to display correctly. You can set the character set of the console before opening a GGSCI session by using the following DOS command:

```
chcp codepagenumber
```

For example, `chcp 437`.

For a code page overview, see https://msdn.microsoft.com/en-us/library/windows/desktop/dd317752(v=vs.85).aspx and the list of code page identifiers https://msdn.microsoft.com/en-us/library/windows/desktop/dd317756(v=vs.85).aspx.

Prerequisites for Installing Oracle GoldenGate for DB2 z/OS

Learn about the requirements to install Oracle GoldenGate for DB2 z/OS.

Choosing an Installation Operating System

Oracle GoldenGate for DB2 for z/OS operates remotely on zLinux, AIX or Intel Linux systems. To capture data, a small component must be installed on the DB2 z/OS system that contains the DB2 instance that will allow Oracle GoldenGate to read the DB2 log data.

To install Oracle GoldenGate on a remote zLinux, AIX or Linux system, you have the following options for connecting to DB2 on the z/OS system:

- DB2 Connect v10.5 or greater
- IBM Data Server Driver for ODBC and CLI v10.5 or greater
- IBM Data Server Client v10.5 or greater
- IBM Data Server Runtime Client v10.5 or greater

Consider the following:

- Extract uses Open Database Connectivity (ODBC) to connect to the DB2 subsystem on the z/OS system. If one of the other drivers is not already installed, the IBM Data Server Driver for ODBC and CLI is the most lightweight driver and is recommended for most configurations, although the other drivers are suitable also.
- To capture DB2 log data, the log reader component must be installed in a Library (PDSE) on the z/OS system. Load Libraries (PDS) are not supported. The library must be authorized program facility (APF) helps your installation protect the system. APF-authorized programs can access system facility (APF) authorized. The log read component is called through SQL from the remote system and since it is APF authorized, an authorized Workload Manager (WLM) environment must also be used to run these programs since the default DB2 supplied WLM environment is not able to run authorized workload.
- No special requirements beyond what capture already has for Oracle GoldenGate delivery. Because this Oracle GoldenGate release is a fully-remote distribution, the former Oracle GoldenGate DB2 Remote product is no longer shipped separately. However, Windows is not supported in Oracle GoldenGate for DB2 z/OS in this release. If you still require delivery to z/OS from Windows, then Oracle GoldenGate DB2 Remote 12.2 is still available.
- UNIX System Services (USS) is no longer required (as in prior releases) except for a few installation procedures.
- Windows only: To apply data to a DB2 target from Windows, Oracle GoldenGate DB2 Remote v12.2 must be used. Capture is not support in this scenario.
- Install Oracle GoldenGate DB2 Remote on a remote system for remote delivery to the DB2 target system. In this configuration, Replicat connects to the target DB2 database by using the ODBC API that is supplied in DB2 Connect . This configuration requires DB2 LUW to be installed on the remote system.

> **Note:**
>
> All of the Oracle GoldenGate functionality that is supported for DB2 for z/OS is supported by DB2Connect. In addition, ASCII character data is converted to EBCDIC automatically by DB2 Connect.

- Although it is possible to install Oracle GoldenGate on zLinux, AIX, and Intel based Linux, the best performance is seen with a system that has the lowest network latency to the z/OS system that you use. Although it is possible to run over a wide area network, the performance suffers due to the increased network latency. Oracle recommends using a zLinux partition on the same physical hardware as the z/OS system that is running DB2 using Hipersockets or a VLAN between the partitions. Otherwise, systems connected with OSA adapters in the same machine room, would be the next best choice. Alternatively, the fastest Ethernet connection between the systems that is available would be acceptable.

Using the Remote Delivery to the DB2 z/OS using DB2Connect

1. For the intermediary system, select any platform that Oracle GoldenGate supports for the DB2 for LUW database. This is the system on which Oracle GoldenGate is installed.
2. Install and run DB2 for LUW on the selected remote system so that the Replicat process can use the supplied DB2 Connect driver.

3. Catalog the DB2 target node in the DB2 for LUW database on the remote system by using the following DB2 command:

   ```
   catalog tcpip node db2_node_name remote DNS_name server DB2_port-number
   ```

4. Add the target DB2 database to the DB2 for LUW catalog on the intermediary system by using the following DB2 command:

   ```
   catalog db database_name as database_alias at node db_node_name
   ```

See the IBM DB2 LUW documentation for more information about these commands.

Prerequisites for Installing Oracle GoldenGate for DB2 for i

Learn about the requirements for installing Oracle GoldenGate Classic Architecture for DB2 for i.

Oracle GoldenGate for DB2 for i runs directly on a DB2 for i source system to capture data from the transaction journals for replication to a target system. To apply data to a target DB2 for i, Oracle GoldenGate can run directly on the DB2 for i target system or on a remote Windows or Linux system. If installed on a remote system, Replicat delivers the data by means of an ODBC connection, and no Oracle GoldenGate software is installed on the DB2 for i target.

> **Note:**
>
> The DB2 for i platform uses one or more **journals** to keep a record of transaction change data. For consistency of terminology in the supporting administrative and reference Oracle GoldenGate documentation, the terms **log** or **transaction log** may be used interchangeably with the term journal where the use of the term journal is not explicitly required.

General Requirements

- Portable Application Solution Environment (PASE) must be installed on the system.
- Java 8 must be installed on the IBM i and the Linux host system where GoldenGate for IBM i will run.
- OpenSSH is recommended to be installed on the system. OpenSSH is part of the IBM Portable Utilities licensed program and allows SSH terminal access to the system in the same manner as other Linux system.
- A library/schema should be dedicated to each install for Oracle GoldenGate on the IBM i system.
- The IBM DB2 for i Program temporary fixes (PTFs) that are required by release for Oracle GoldenGate are detailed in the following tables:

IBM i7.3 Group PTF	Level	Name	Notes
SF99730	23103	Cumulative PTF	NA

IBM i7.3 Group PTF	Level	Name	Notes
SF99731	12	All PTF groups except cumulative PTF package	.

IBM i7.4 Group PTF	Level	Name	Notes
SF99740	23117	Cumulative PTF	NA
SF99741	8	All PTF groups except cumulative PTF package	NA

IBM i7.5 Group PTF	Level	Name	Notes
SF99750	23110	Cumulative PTF	NA
SF99751	4	All PTF groups except cumulative PTF package	NA

These required PTFs are the levels at which Oracle GoldenGate has been tested against for the 21*c* releases. To check the group PTF levels, you must use the `WRKPTFGRP` command from a 5250 terminal session and check for the specific PTFs with the commands shown in the preceding tables. The specific extra PTFs must be at least temporarily applied.

Prerequisite Setup the DB2 for i System

Follow these steps before you begin installing Oracle GoldenGate for a DB2 for i system.

> **Note:**
>
> The user profile running the install must have authority to the `RSTOBJ` command.

1. On the system where Oracle GoldenGate is to be installed, create a directory for Oracle GoldenGate.

```
- MKDIR DIR('/GoldenGate')
```

2. You can create a library for Oracle GoldenGate on the installation system, or you can create it through the installation script that you will run later in these steps.

```
- CRTLIB LIB(goldengate) TEXT('Oracle GoldenGate Product Library') ASP(1)
```

3. Unzip the downloaded file on your system.
4. FTP the resulting tar file from that system to the folder that you created on the DB2 for i installation system.

```
ftp IBMi_IP_address
.
```

```
User (system:(none)):userid
.
331 Enter password.
. Password: password .
230 userid logged on. .
ftp> bin
.
ftp> cd goldengate
.
ftp> put install_file
.
ftp> quit
```

5. (If you created a library) From a 5250 terminal session, change your current library to the Oracle GoldenGate library.

   ```
   CHGCURLIB Oracle_GoldenGate_ library
   ```

6. Run a QP2TERM terminal session.

   ```
   - CALL QP2TERM
   ```

7. Extract the installation objects from the tar file.

   ```
   tar -xf tar_file
   ```

8. In the Oracle GoldenGate directory, run the shell script `ggos400install`.

   ```
   ggos400install -l goldengate
   ```

   The default is to install the required objects into the current library (set in the preceding steps), but you can create a library by using the `-c` option. Additional options are available.

   > **Note:**
   >
   > There must be a separate Oracle GoldenGate library for each Oracle GoldenGate directory. The install script checks for this condition and will prevent installation to the same library that another installation is using. The reason for this is to prevent mismatches between the Oracle GoldenGate installation and the `OGGPRCJRN *SRVPGM` object.

   Syntax:

   ```
   ./ggos400install [-h] [-f] [-u userid] [[-a aspname] | [-n aspnum]] [-c|-l
   library name]
   ```

   Options:

   - `-h` shows this usage help.

- `-f` forces a change to a new installation library. This argument only affects an existing installation.
- `-u userid` specifies the userid that will own the installation.
- `-a aspname` specifies the name of the ASP where objects will be restored. If no `aspname` is provided, the system asp is assumed. This option cannot be used with `-n`.
- `-n aspnum` specifies the number of the user asp where the objects will be restored. This option cannot be used with `-a`.
- `-c library` specifies the name of the library where the objects will be restored. The library will be created.
- `-l library` specifies the name of the library where the objects will be restored. The library must exist. If a library is not specified for a new installation, the installer will attempt to use the current library of the user that is running the installer. If a library is not specified for an existing installation, the installer will attempt to use the library that is set in the `oggprcjrn.srvpgm` link.

> **Note:**
>
> If Oracle GoldenGate is reinstalled, you must run `ggos400install` again. On a reinstall, `ggos400install` will recognize the prior configuration, so no arguments are needed. If the `oggprcjrn.srvpgm` link is changed or removed, `ggos400install` must be run again with the Oracle GoldenGate installation library specified by the link.

9. Exit QP2TERM.

```
- F3
```

> **Note:**
>
> On an DB2 for i system, it is not necessary to create any working directories in the Oracle GoldenGate installation directory. The `ggos400install` script performs this task.

10. Install Oracle GoldenGate on the DB2 for i database server, see Installing for all Platforms.

Prerequisites for Installing Oracle GoldenGate for DB2 LUW

Learn about the requirements for installing Oracle GoldenGate Classic Architecture for DB2 LUW.

Choosing an Installation System for DB2 LUW

To install Oracle GoldenGate for DB2 LUW, you can use either of the following configurations:

- Install Oracle GoldenGate on the DB2 LUW database server.

- Install Oracle GoldenGate on another server, and configure Oracle GoldenGate to connect remotely to the database server through DB2 Connect. All of the Oracle GoldenGate functionality that is supported for DB2 LUW is supported in this configuration. To use this option, proceed to Choosing and Configuring a System for Remote Capture or Delivery.

To Use Remote Delivery to the DB2 LUW System Using DB2 Connect

1. For the intermediary system, select any supported for the DB2 for LUW database to be the system that Oracle GoldenGate is installed on.
2. Install and run DB2 for LUW on the selected remote system so that the Replicat process can use the supplied DB2 Connect driver.
3. Catalog the DB2 target node in the DB2 for LUW database on the remote system by using the following DB2 command:

   ```
   catalog tcpip node db2_node_name remote DNS_nameserver DB2_port-number
   ```

4. Add the target DB2 database to the DB2 for LUW catalog on the intermediary system by using the following DB2 command:

   ```
   catalog db database_name as database_alias at node db_node_name
   ```

   > **Note:**
   >
   > Refer to the IBM DB2 LUW documentation for more information about these commands.

5. Install Oracle GoldenGate. For CA, see Installing Oracle GoldenGate for Non-Oracle Databases. For MA, see Installing Microservices Architecture for Oracle GoldenGate.
6. Specify the DB2 target database name with the Replicat parameter.`TARGETDB` when you configure the Oracle GoldenGate processes.

Choosing and Configuring a System for Remote Capture or Delivery

In a remote installation, you install Oracle GoldenGate on a server that is remote from the source or target database server. This server can be any Linux, UNIX, or Windows platform that Oracle GoldenGate supports for the DB2 for LUW database. The Oracle GoldenGate build must match the version of DB2 LUW that is running on the installation server.

In this configuration, the location of the database is transparent to Extract and Replicat. Extract can read the DB2 logs on a source DB2 LUW database server, and Replicat can apply data to a target DB2 LUW server.

To Configure Remote Capture or Delivery:

1. Install and run DB2 for LUW on the remote server that has DB2 Connect.
2. Catalog the remote server in the DB2 source or target database by using the following DB2 command.

   ```
   catalog tcpip node db2_node_name remote remote_DNS_name
   ```

3. Catalog the DB2 target node in the DB2 for LUW database on the remote server by using the following DB2 command:

```
catalog tcpip node db2_node_name remote remote_DNS_name server
remote_port_number
```

4. Add the DB2 source or target database to the DB2 catalog on the remote server by using the following DB2 command:

```
catalog db database_name as database_alias at node db_node_name
```

> **Note:**
>
> Refer to the IBM DB2 LUW documentation for more information about these commands.

5. Download and install the Oracle GoldenGate build that is appropriate for the DB2 LUW database on the remote server.

Prerequisites for Installing Oracle GoldenGate for MySQL

Installing OpenSSL on Linux

OpenSSL 1.0 is included with the core operating system packages of OEL 7 and RHEL7 but is not included with OEL8/9 or RHEL 8/9, and therefore must be manually installed for these operating systems/versions.

To install the required OpenSSL 1.0 libraries (`libssl.so.10` and `libcrypto.so.10`), download and install the MySQL Connector/ODBC, version 8.0.17, using the following instructions:

1. Select version **`8.0.17`** from the **Product Version** drop-down menu, using the link, https://downloads.mysql.com/archives/c-odbc/.
2. Select **Linux-Generic** from the **Operating System** drop-down menu.
3. Click the **Download** link for the **`x86, 64-bit`** version of the package: ***Linux - Generic (glibc 2.12) (x86, 64-bit), Compressed TAR Archive***.
4. Save the file to a location of your choice, such as `/opt/app/`, and untar the file, as shown in the following example:

```
tar -xvzf mysql-connector-odbc-8.0.17-linux-glibc2.12-x86-64bit
```

5. The OpenSSL 1.0 libraries needed for Oracle GoldenGate are located in the `mysql-connector-odbc-8.0.17-linux-glibc2.12-x86-64bit/lib` folder. This path needs to be added to the `LD_LIBRARY_PATH` system variable, prior to creating a MySQL deployment, as shown in the following example.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/app/mysql-connector-
odbc-8.0.17-linux-glibc2.12-x86-64bit/lib
```

Installing OpenSSL on Windows

OpenSSL 1.0 needs to be installed on the Windows server for Oracle GoldenGate for MySQL.

To install the required OpenSSL 1.0 libraries (`libcrypto-1_1-x64.dll` and `libssl-1_1-x64.dll`), download and install the MySQL Connector/ODBC, version 8.0.33, using the following instructions:

1. Select version `8.0.33` from the **Product Version** drop-down menu, using the link, https://downloads.mysql.com/archives/c-odbc/.
2. Select **Microsoft Windows** from the **Operating System** drop-down menu.
3. Click the **Download** link for the `x86, 64-bit` version of the package: ***Windows (x86, 64-bit), ZIP Archive***.
4. Save the file to a location of your choice, such as `C:\mysql` and extract the file, as shown in the following example:

   ```
   tar -xvf mysql-connector-odbc-noinstall-8.0.33-winx64.zip
   ```

5. The OpenSSL 1.0 libraries needed for Oracle GoldenGate are located in the `mysql-connector-odbc-noinstall-8.0.33-winx64\lib` folder. This path needs to be added to the `PATH` system variable, prior to creating a MySQL deployment, as shown in the following example.

   ```
   set PATH=%PATH%;C:\mysql\ mysql-connector-odbc-noinstall-8.0.33-winx64\lib
   ```

Prerequisites for Installing Oracle GoldenGate for Oracle Database

Oracle GoldenGate uses a single, unified build for capturing from and applying to multiple major Oracle Database versions for supported operating systems by including the latest Oracle database client libraries as part of Oracle GoldenGate. As a result, there are no prerequisites for installing Oracle GoldenGate for Oracle database.

Prerequisites for Installing Oracle GoldenGate for PostgreSQL

PostgreSQL libpq Module

For Oracle GoldenGate installations beginning with release version 19.1.0.0.220419 and after, required PostgreSQL `libpq` libraries are now packaged with the Oracle GoldenGate installation and do not need to be installed separately.

For Oracle GoldenGate installations prior to release version 19.1.0.0.220419, PostgreSQL `libpq` libraries need to be manually installed where Oracle GoldenGate is to be installed. Perform the instructions below to install the correct `libpq` module when running Oracle GoldenGate release versions prior to 19.1.0.0.220419.

The steps to install the PostgreSQL `libpq` module are:

1. Follow the steps to install the required PostgreSQL package available at: https://www.postgresql.org/download/.
2. Select the **Linux** operating system family and **Red Hat/Rocky/CentOS Linux** distribution from the **Packages and Installers** drop-down list.
3. Select the PostgreSQL version based on the highest version of PostgreSQL that will be used with Oracle GoldenGate.
4. Select the platform where Oracle GoldenGate will be installed, such as **Red Hat Enterprise**, **CentOS**, **Scientific**, or **Oracle version 7**.
5. Select the architecture as **x86_64**from the **Architecture** drop-down list.

This will provide the PostgreSQL setup scripts needed to install the required package(s).

6. Install the repository RPM and the `libs` module. For example:

```
# Install the repository RPM:

sudo yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/
EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm

# Install PostgreSQL libs module:

sudo yum install -y postgresql12-libs
```

Database Software for Capture Requirements

To capture from a PostgreSQL database, Oracle GoldenGate requires the `test_decoding` database plug-in be installed for the database. This plug-in might not have been installed by default when the database was installed.

Ensure that the `postgresqlversion#-contrib` package is installed on the database server. For example:

```
sudo yum install postgresql12-contrib
```

Additional Programs and Settings

Configure the `LD_LIBARY_PATH` and `OGG_HOME` environment variables prior to installing Oracle GoldenGate.

- For Oracle GoldenGate installations prior to release version 19.1.0.0.220419, set the following environment variables before installing Oracle GoldenGate:
  1. `OGG_HOME` – The planned Oracle GoldenGate installation path.
  2. `LD_LIBARY_PATH` – Includes the `$OGG_HOME/lib` and PostgreSQL `libpq` directories.

     Example:

     ```
     export OGG_HOME=<path_to_install_GoldenGate>

     export LD_LIBRARY_PATH=$OGG_HOME/lib:/usr/pgsql-12/lib
     ```
- For Oracle GoldenGate installations of release version 19.1.0.0.220419 and after, set the following environment variables before installing Oracle GoldenGate:
  1. `OGG_HOME` – The planned Oracle GoldenGate installation path.
  2. `LD_LIBARY_PATH` – Includes the `$OGG_HOME/lib` directory.

     Example:

     ```
     export OGG_HOME=<path_to_install_GoldenGate>

     export LD_LIBRARY_PATH=$OGG_HOME/lib
     ```
- When installing Oracle GoldenGate on a remote server (one different from where the database is running), set the remote server's time and time zone to that of the source database server so that Oracle GoldenGate Extract can correctly position by time when creating the Extract with the `BEGIN` option, otherwise, position by a valid `LSN` value.

Prerequisites for Installing Oracle GoldenGate for SQL Server

Observe the following program and settings information for Oracle GoldenGate for SQL Server:

- Install either the Microsoft ODBC Driver 17 or Microsoft ODBC Driver 18 for the operating system where Oracle GoldenGate is to be installed:

  For Oracle GoldenGate on Windows, install the driver available at the following link:

  https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-2017

  For Oracle GoldenGate on Linux, install the driver available at this link, and follow the instructions for RHEL and Oracle Linux packages:

  https://learn.microsoft.com/en-us/sql/connect/odbc/linux-mac/installing-the-microsoft-odbc-driver-for-sql-server?view=sql-server-2017

  > **Note:**
  >
  > Support for Microsoft ODBC Driver 18 was added with Oracle GoldenGate release 19.1.0.0.221021. Versions prior to release 19.1.0.0.221021 do not support the Microsoft ODBC Driver 18 for SQL Server.

- Installation of the Oracle GoldenGate CDC cleanup tasks requires the Microsoft `sqlcmd` Utility. Download instructions for Windows and Linux systems can be found at:

  https://docs.microsoft.com/en-us/sql/tools/sqlcmd-utility?view=sql-server-ver15

- To install capture on a remote Linux or Windows server, set the remote server's time and time zone to that of the database server, or use LSN based positioning for the Extract.

Prerequisites for Installing Oracle GoldenGate for Sybase

Learn about the requirements to install Oracle GoldenGate for Sybase.

Oracle GoldenGate uses shared libraries. When you install Oracle GoldenGate on a UNIX system, the following must be done *before* you run GGSCI or any other Oracle GoldenGate process . If you will be running an Oracle GoldenGate program from outside the Oracle GoldenGate installation directory on a UNIX system:

- (Optional) Add the Oracle GoldenGate installation directory to the `PATH` environment variable.
- (Required) Add the Oracle GoldenGate installation directory to the shared-libraries environment variable.

When Oracle GoldenGate connects remotely to the database server through SQL*Net, the following are required:

- Replicat: The Oracle client library and the Oracle GoldenGate build must have the same Oracle version, bit type (64-bit or IA64), and operating system version.
- Extract: The Oracle client library and the Oracle GoldenGate build must have the same Oracle version, bit type (64-bit or IA64), and operating system version. In addition, both operating systems must be the same endian.

For example, given an Oracle GoldenGate installation directory of `/users/ogg`, the second command in the following example requires these variables to be set:

Table 2-2   Command Requiring Library Variable

Command	Requires GG libraries in environment variable?
`$ users/ogg > ./ggsci`	No
`$ users > ./ogg/ggsci`	Yes

Set the Variables in Korn Shell

```
PATH=installation_directory:$PATH
export PATH
shared_libraries_variable=absolute_path_of_installation_directory:$shared_librari
es_variable
export shared_libraries_variable
```

Set the Variables in Bourne Shell

```
export PATH=installation_directory:$PATH
export
shared_libraries_variable=absolute_path_of_installation_directory:$shared_librari
es_variable
```

Set the Variables in C Shell

```
setenv PATH installation_directory:$PATH
setenv shared_libraries_variable
absolute_path_of_installation_directory:$shared_libraries_variable
```

Where: *`shared_libraries_variable`* is one of the variables shown in Table 2-3:

Table 2-3   UNIX/Linux Library Path Variables per Platform

Platform[1]	Environment variable

`IBM AIX`
`IBM z/OS`
	`LIBPATH`

`HP-UX`	`SHLIB_PATH`

`Sun Solaris`
`LINUX`
	`LD_LIBRARY_PATH`[2]

[1] A specific platform may or may not be supported by Oracle GoldenGate for your database.

[2] In 64-bit environments with 32-bit Oracle databases, Oracle GoldenGate requires the `LD_LIBRARY_PATH` to include the 32-bit Oracle libraries.

Example

```
export LD_LIBRARY_PATH=/ggs/12.0:$LD_LIBRARY_PATH
```

> **Note:**
>
> To view the libraries that are required by an Oracle GoldenGate process, use the `ldd` *`goldengate_process`* shell command before starting the process. This command also shows an error message for any that are missing.

To install Oracle GoldenGate for Sybase, see Installing for all Platforms.

Prerequisites for Installing Oracle GoldenGate for Oracle TimesTen

Learn about the prerequisites for installing Oracle GoldenGate for Oracle TimesTen.

Supported Platforms and Database Versions

Oracle TimesTen supports installing Oracle GoldenGate on Linux.

For supported platform and database version information, review the certification matrix:

https://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html.

Oracle TimesTen Software Installation

The Oracle TimesTen Client needs to be installed on the server where Oracle GoldenGate is going to be installed. If Oracle GoldenGate is installed on the Oracle TimesTen database server, then the required components are already available. However, if you are installing Oracle GoldenGate on a hub server, then you must separately install the Oracle TimesTen Client.

In both cases you will need to configure the ODBC connection information.

For Linux platforms there is only one TimesTen software distribution that provides both server and client components. To download the Oracle TimesTen Software, visit:

https://www.oracle.com/database/technologies/timesten-downloads.html

Before beginning to install Oracle GoldenGate with Oracle TimesTen, you must also set the `LD_LIBRARY_PATH` variable:

1. Download the *TimesTen Scaleout and TimesTen Classic/Cache 18.x for Linux x86 (64-bit)* build.
2. Extract the Oracle TimesTen installation files to the designated location, based on the instructions provided in *Oracle TimesTen In-Memory Database Installation Guide*.
3. Set the `LD_LIBARY_PATH` system variable to include the TimesTen installation’s `lib` directory. This system variable must be set to install and run Oracle GoldenGate. Example:

   ```
   export LD_LIBRARY_PATH=/installpath/tt18.1.2.2.0/lib:$LD_LIBRARY_PATH
   ```

Client-only Instance Creation

For non-database server environments where you plan to install Oracle GoldenGate, after installing the Oracle TimesTen client libraries, follow the TimesTen document instructions to create a client-only instance of TimesTen.

1. Perform the following:

   ```
   [oracle@tt_installation_dir]$ ./tt18.1.2.1.0/bin/ttInstanceCreate -
   clientonly
   ```

2. Follow the instance installation prompts, taking note of where the TimesTen instance is installed. This information will be required when setting up a Replicat's ODBC connection to TimesTen.
3. Set the `TIMESTEN_HOME` system variable to the TimesTen instance path.

   Example:

   ```
   export TIMESTEN_HOME=/instancepath/tt181
   ```

Installing Oracle GoldenGate Classic Architecture

Learn about the steps to install Oracle GoldenGate Classic Architecture with various supported databases.

Installing Oracle GoldenGate Classic for Oracle Database

Installing Oracle GoldenGate installs all of the components that are required to run and manage the processing (excluding any components required from other vendors, such as drivers or libraries) and it installs the Oracle GoldenGate utilities.

Oracle GoldenGate for Oracle Database is installed from the Oracle Universal Installer (OUI). OUI is a graphic installation program that prompts you for the input required to install the Oracle GoldenGate binaries and working files. It also sets the correct database environment that Oracle GoldenGate will operate in.

You can use OUI on any of the Linux, UNIX, and Windows platforms supported by OUI and Oracle GoldenGate.

An instance of Oracle GoldenGate can be installed for only one major Oracle Database version in any given Oracle home. For example, if you have Oracle Database 11.2 and 12.1, you must have separate Oracle GoldenGate installations for each one. This does not apply to data patch levels within the same major release. You can install multiple instances of Oracle GoldenGate for the same or different database versions on the same host.

The installer registers the Oracle GoldenGate home directory with the central inventory that is associated with the selected database. The inventory stores information about all Oracle software products installed on a host, provided the product was installed using OUI.

1. Copy the Oracle GoldenGate installation file to the system and directory where you want to install Oracle GoldenGate, and then extract it.

> **Note:**
>
> The path *cannot* contain any spaces.

2. Install using one of these installation methods:
   - Performing an Interactive Installation with OUI.
   - Performing a Silent Installation with OUI.
3. From this directory, run the GGSCI program, `ggsci.exe`. For Linux and UNIX, open a command shell to run `ggsci.sh`.

   For Windows, it may be necessary to run `ggsci.exe` as an Administrator based on the systems User Account Control settings. Right-click the executable file then select **Run as administrator**.
4. In GGSCI, issue the following command to create the Oracle GoldenGate working directories.

   ```
   CREATE SUBDIRS
   ```
5. Exit GGSCI.

   ```
   EXIT
   ```

Performing an Interactive Installation with OUI

The interactive installation provides a graphical user interface that prompts for the required installation information. These instructions apply to new installations as well as upgrades. However, to perform an upgrade to Oracle GoldenGate, follow the instructions in the Upgrade section of this guide, which includes a prompt to run OUI at the appropriate time.

1. Expand the installation file.
2. From the expanded directory, run the `runInstaller` program on UNIX or Linux, or run `setup.exe` on Windows.
3. On the **Select Installation Option** page, select the Oracle GoldenGate version to install, and then click **Next** to continue.
4. On the **Specify Installation Details** page, specify the following:
   - For **Software Location**, specify the Oracle GoldenGate installation directory. It can be a new or existing directory that is empty and has the amount of disk space shown on the screen or in the existing Oracle GoldenGate installation location (if you are upgrading an existing Oracle GoldenGate installation). The default location is under the installing user's home directory, but Oracle recommends changing it to a local directory that is not mounted and has no quotas. The specified directory cannot be a registered home in the Oracle central inventory. If installing in a cluster, install Oracle GoldenGate on local storage on each node in the cluster to provide high availability options for upgrading and software patching.

     > **Note:**
     >
     > The software location path cannot contain any whitespace.

- (Optional) Select **Start Manager** to perform configuration functions, such as creating the Oracle GoldenGate subdirectories in the installation location, setting library paths, and starting Manager on the specified port number. To proceed, a database must exist on the system. When **Start Manager** is selected, the **Database Location and Manager Port** fields are displayed.
  - For **Database Location**, the database version in the specified location must be **Oracle Database 12c** if you are installing Oracle GoldenGate for Oracle Database 12c or **Oracle Database 11g** if you are installing Oracle GoldenGate for Oracle Database11g. The database must have a registered home in the Oracle central inventory. The installer registers the Oracle GoldenGate home directory with the central inventory.
  - For **Manager Port**, accept the default port number or enter a different unreserved, unrestricted port number for the Manager process to use for interprocess communication. The default port is the first available one starting with 7809. If you are installing multiple instances of Oracle GoldenGate on the same system, each must use a different port number.
- Click **Next** to continue. If this is an upgrade to an existing Oracle GoldenGate installation, OUI prompts that the selected software location has files or directories. Click **Yes**.

5. The **Create Inventory** page is displayed if this is the first Oracle product to be installed from OUI on a host that does not have a central inventory.
   - For **Inventory Directory**, specify a directory for the central inventory. It can be a new directory or an existing directory that is empty and has the amount of disk space shown on the screen. The directory cannot be on a shared drive.
   - Select an operating system group in which the members have write permission to the inventory directory. This group is used to add inventory information to the Oracle GoldenGate subfolder.
6. On the **Summary** page, confirm that there is enough space for the installation and that the installation selections are correct. Optionally, click **Save Response File** to save the installation information to a response file. You can run the installer from the command line with this file as input to duplicate the results of a successful installation on other systems. You can edit this file or create a new one from a template. See Performing a Silent Installation with OUI.
7. Click **Install** to begin the installation or **Back** to go back and change any input specifications. When upgrading an existing Oracle GoldenGate installation, OUI notifies you that the software location has files or directories. Click **Yes** to continue. You are notified when the installation is finished.
8. If you created a central inventory directory, you are prompted to run the `INVENTORY_LOCATION/orainstRoot.sh` script. This script must be executed as the root operating system user. This script establishes the inventory data and creates subdirectories for Oracle GoldenGate.

Performing a Silent Installation with OUI

These instructions apply to new installations, as well as upgrades.

You can perform a silent installation from the command console if the system has no X-Windows interface or to perform an automated installation. Silent installations can ensure that multiple users in your organization use the same installation options when they install your Oracle products.

You perform a silent installation by running a response file. You can create a response file by selecting the **Save Response File** option during an interactive OUI session or by editing a template. To run a response file, issue the following command.

```
runInstaller -silent -nowait -responseFile absolute_path_to_response_file
```

The response files and the template are stored in the response subdirectory of the Oracle GoldenGate installation directory. The Oracle GoldenGate response file contains a standard set of Oracle configuration parameters in addition to parameters that are specific to Oracle GoldenGate. These parameters correspond to the fields in the interactive session.

> **Note:**
>
> If you are upgrading an existing Oracle GoldenGate installation with the silent option, you might get the following warning:
>
> ```
> WARNING:OUI-10030:You have specified a non-empty directory to install this
> product. It is recommended to specify either an empty or a non-existent
> directory.
> You may, however, choose to ignore this message if the directory contains
> Operating System generated files or subdirectories like lost+found. Do you
> want to proceed with installation in this Oracle Home?
> ```
>
> Press **ENTER** to continue.

Installing Oracle GoldenGate for Non-Oracle Databases

Learn how to install Oracle GoldenGate on Linux, UNIX, and Windows environments for non-Oracle databases.

Consult the section for your database to meet any prerequisites and learn about any installation considerations.

Installing for all Platforms

1. Copy the Oracle GoldenGate installation file to the system and directory where you want to install Oracle GoldenGate, and then unzip it.

   > **Note:**
   >
   > The installation path *cannot* contain any spaces.

2. From this directory, run GGSCI. For Linux and UNIX, open a command shell to run `ggsci.sh`.

   For Windows, it may be necessary to run `ggsci.exe` as an Administrator based on the systems User Account Control settings. Right-click the executable file then select **Run as administrator**.

3. In GGSCI, issue the following command to create the Oracle GoldenGate working directories.

   ```
   CREATE SUBDIRS
   ```

4. Exit GGSCI.

   ```
   EXIT
   ```

Specifying a Custom Manager Name for Windows

If you plan to install the Manager process as a Windows service and either of the following is true, then you must specify a custom name for the Manager service:

- You are installing the Manager as a Windows service and want to use a service name other then the default, which is `GGSMGR`.
- You want to have multiple Manager processes running as Windows services on this system. Each Manager service on a system must have a unique name.

To specify a custom Manager service name:

1. From the Oracle GoldenGate installation directory, run `ggsci.exe` from the Oracle GoldenGate directory.
2. Issue the following command:

   ```
   EDIT PARAMS ./GLOBALS
   ```

   > **Note:**
   >
   > The `./` portion of this command must be used, because the `GLOBALS` file must reside at the root of the Oracle GoldenGate installation file.

3. In the file, add the following line, where *`name`* is a unique, one-word name for the Manager service.

   ```
   MGRSERVNAME name
   ```

4. Save the file. The file is saved automatically with the name `GLOBALS`, but without a file extension. Do not move this file because it is used during installation of the Windows service and during data processing.

Installing Manager as a Windows Service

By default, Manager is not installed as a service and can be run by a local or domain account. However, when run this way, Manager will stop when the user logs out. When you install Manager as a service, you can operate it independently of user connections, and you can configure it to start manually or at system startup.

Installing Manager as a service is required on a Windows Cluster, but optional otherwise.

To install Manager as a Windows service:

1. Click **Start**, then **Run**, and then type `cmd` in the Run dialog box.
2. Go to the directory that contains the Manager program that you are installing as a service, then run the `INSTALL` utility with the following syntax:

   ```
   install option [...]
   ```

   Where *`option`* is one of the following:

Table 2-4 INSTALL Utility Options

Option	Description
`ADDEVENTS`	Adds Oracle GoldenGate events to the Windows Event Manager.
`ADDSERVICE`	Adds Manager as a service with the name that is specified with the `MGRSERVNAME` parameter in the `GLOBALS` file, if one exists, or the `GGSMGR` default. The `ADDSERVICE` configures the service to run as the Local System account, the standard for most Windows applications because the service can be run independently of user logins and password changes. To run Manager as a specific account, use the `USER` and `PASSWORD` options. **Note:** A user account can be changed by selecting the **Properties** action from the Services applet of the Windows Control Panel. The service is installed to start at system boot time (see `AUTOSTART`). To start it after installation, either reboot the system or start the service manually from the Services applet in the Control Panel.
`AUTOSTART`	Sets the service that is created with `ADDSERVICE` to start at system boot time. This is the default unless `MANUALSTART` is used.
`MANUALSTART`	Sets the service that is created with `ADDSERVICE` to start manually through GGSCI, a script, or the Services applet in the Control Panel. The default is `AUTOSTART`.
`USER` *`name`*	Specifies a domain user account that executes Manager. For the *`name`*, include the domain name, a backward slash, and the user name, for example `HEADQT\GGSMGR` . By default, the Manager service is installed to use the Local System account.
`PASSWORD` *`password`*	Specifies the password for the user that is specified with `USER`.

3. If Windows User Account Control (UAC) is enabled, you are prompted to allow or deny the program access to the computer. Select **Allow** to enable the `INSTALL` utility to run.

   The `INSTALL` utility installs the Manager service with a local system account running with administrator privileges. No further UAC prompts will be encountered when running Manager if installed as a service.

> **Note:**
>
> If Manager is not installed as a service, Oracle GoldenGate users will receive a UAC prompt to confirm the elevation of privileges for Manager when it is started from the GGSCI command prompt. Running other Oracle GoldenGate programs also returns a prompt.

Patching for Classic Architecture

Patches for Oracle GoldenGate for Oracle Database can be found on My Oracle Support when available, and are located under the Patches and Updates section of MOS.

Cumulative and one-off patches for Oracle GoldenGate can be applied on top of a base release or previously patched release, or they may be a one-off patch that should be applied to a specific Oracle GoldenGate version. The instructions in the subsequent topic apply to both types of patches.

Learn to prepare and install patches using OPatch for Classic Architecture.

Downloading Patches for Oracle GoldenGate

Download the appropriate patches for the Oracle GoldenGate build for each system that will be part of the Oracle GoldenGate configuration.

1. Using a browser, navigate to `https://support.oracle.com`.
2. Log in with your Oracle ID and password.
3. Select the **Patches and Upgrades** tab.
4. On the Search tab, click **Product or Family**.
5. In the `Product` field, type **Oracle GoldenGate**.
6. From the **Release** drop-down list, select the release version that you want to download.
7. Make certain that **Platform** is displayed as the default in the next field, and then select the platform from the drop-down list.
8. Leave the last field blank.
9. Click **Search**.
10. In the Patch Advanced Search Results list, select the patch that best meets your search criteria, making certain that the Oracle GoldenGate patch that you select corresponds to the database that will be used.

    When you select the build, a dialog box pops up under the build description, and then you are advanced to the download page.
11. Click the Patch file name link for each patch that you want to download. The File Download dialog box appears.
12. Select either **Open with** or **Save File**:

To...	Select...
Extract the patch immediately	**Open with**, then select the desired file extraction utility and extract the files to a location on your file system.
Save the patch for later extraction	Select **Save file**, then save to a directory on your file system.

> **Note:**
>
> Before installing the software, see Oracle GoldenGate Release Notes for any new features, parameter changes, upgrade requirements, known issues, or bug fixes that affect your current configuration.

Patching Oracle GoldenGate Classic Architechture for Oracle Database Using OPatch

This topic lists the prerequisites and the steps to be performed for patching Oracle GoldenGate Classic Architecture for Oracle database only, using OPatch.

Perform the following prerequisites before installing the patch:

1. Download and install the most recent release of OPatch, and keep a note of the installation directory where you installed the latest release of OPatch.

   Details from where to download OPatch is available at: How To Download And Install The Latest OPatch(6880880) Version (Doc ID 274526.1)
2. Download the Oracle GoldenGate patch and maintain a location for storing the contents of the patch ZIP file. This location or the absolute path is referred to as *`patch_top_dir`* in the subsequent steps.
3. Navigate to the *`patch_top_dir`* directory and run the following command to extract the contents of the patch ZIP file to the location you created previously.

   ```
   $ cd patch_top_dir
   ```

   ```
   $ unzip patch_number_version_platform.zip
   ```
4. Navigate to the unzipped patch directory:

   ```
   $ cd patch_top_dir/patch_number_dir
   ```
5. Set the `ORACLE_HOME` environment variable to the Oracle GoldenGate installation directory:

   ```
   $ export ORACLE_HOME=GoldenGate_Installation_Path
   ```
6. Set the `PATH` environment variable to include the locations of the `$ORACLE_HOME` and `OPatch` directories.

   ```
   $ export PATH=$PATH:$ORACLE_HOME:/OPatch
   ```
7. Run the following command to verify the Oracle inventory, which OPatch accesses to install the patches:

   ```
   $ opatch lsinventory
   ```

   If the command displays any errors, contact Oracle Support to resolve the issue.
8. Run the OPatch prerequisites check and verify that it passes.

   ```
   $ opatch prereq CheckConflictAgainstOHWithDetail -ph ./
   ```

   If any errors are displayed, identify the error type. OPatch categorizes conflicts in the following types:

   - Conflicts with a patch already applied to the `ORACLE_HOME`: In this case, stop the patch installation and contact Oracle Support Services.
   - Conflicts with a patch already applied to the `ORACLE_HOME` that is a subset of the patch you are trying to apply: In this case, continue with the patch installation because the new patch contains all the fixes from the existing patch in the `ORACLE_HOME`. The subset patch will automatically be rolled back prior to the installation of the new patch.

9. Before patching Oracle GoldenGate, ensure that you shut down all processes such as Extracts and Replicats, and stop all other services such as the Oracle GoldenGate Monitor JAgent and Performance Metrics Service.
    a. Use the Oracle GoldenGate Software Command Interface (`ggsci`) in the GoldenGate Software Home to stop all processes.

    ```
    $ ./ggsci
    ```

    b. Stop the Extract and Replicat processes and the Distribution Paths.

    ```
    GGSCI> STOP ER *
    ```

    c. If monitoring is enabled, stop the GoldenGate Monitor JAgent and Performance Metrics Service.

    ```
    GGSCI> STOP PMSRVR
    GGSCI> STOP JAGENT
    ```

    d. Stop the Manager process.

    ```
    GGSCI> STOP MGR!
    ```

    e. Re-check to verify that all processes have stopped.

    ```
    GGSCI> INFO ALL
    ```

    f. Exit the Oracle GoldenGate Software Command Interface.

    ```
    GGSCI> EXIT
    ```

Perform the following steps to install the patch:

10. Install the patch by running the following command:

    ```
    $ opatch apply
    ```

    When the `OPatch` command starts, it validates the patch and ensures that there are no conflicts with the software already installed in `ORACLE_HOME` of the Oracle GoldenGate release.

11. After the patch installation completes, run the following command to verify that the Oracle inventory contains the installed patch:

    ```
    $ opatch lsinventory
    ```

12. Start the Manager, followed by the other services such as GoldenGate Monitor JAgent and Performance Metrics Service, and the Oracle GoldenGate processes.
    a. Use Oracle GoldenGate Software Command Interface (`ggsci`) in the GoldenGate Software Home to start all processes and services.

    ```
    $ ./ggsci
    ```

b. Start the Manager process.

```
GGSCI> START MGR
```

c. If monitoring was enabled, start the GoldenGate Monitor JAgent and Performance Metrics Service.

```
GGSCI> START PMSRVR
GGSCI> START JAGENT
```

d. Start the Extracts and the Replicats.

```
GGSCI> START ER *
```

e. Check the status and verify that all processes and services are running.

```
GGSCI> INFO ALL
```

Patching Oracle GoldenGate Classic Architecture for Non-Oracle Databases

The following steps guide you to install patches for any of the non-Oracle databases released for Oracle GoldenGate.

1. (Source and target systems) Back up the current Oracle GoldenGate installation directory on the source and target systems, and any working directories that you have installed on a shared drive in a cluster (if applicable).
2. (Source and target systems, as applicable) Expand the patch version 19c (19.1.0) of Oracle GoldenGate into a new directory on each system (not the current Oracle GoldenGate directory). Do not create the sub-directories, just complete the steps to the point where the installation files are expanded.
3. (Source system) Stop user activity on objects in the Oracle GoldenGate configuration.
4. (Source system) In GGSCI on the source system, issue the `SEND EXTRACT` command with the `LOGEND` option until it shows there is no more data in transaction log to process.

   ```
   GGSCI> SEND EXTRACT group LOGEND
   ```

5. (Source system) In GGSCI, stop Extract and data pumps:

   ```
    GGSCI> STOP EXTRACT group
   ```

6. (Target systems) In GGSCI on each target system, issue the `SEND REPLICAT` command with the `STATUS` option until it shows a status of "At EOF" to indicate that it finished processing all of the data in the trail. This must be done on all target systems until all Replicat processes return `At EOF`.

   ```
   GGSCI> SEND REPLICAT group STATUS
   ```

7. (Target systems) In GGSCI, stop all Replicat processes:

   ```
   GGSCI> STOP REPLICAT group
   ```

8. (Source and target systems) In GGSCI, stop Manager on the source and target systems.

   ```
   GGSCI> STOP MANAGER
   ```

9. (Source for MySQL with DDL replication enabled) Ensure that there are no new DDL operations during the patching process, then stop the metadata server by executing the following:

   ```
   ./ddl_install.sh stop user-id  password  port-number
   ```

10. (Source and target systems) Move the expanded Oracle GoldenGate files from the new directory to your existing Oracle GoldenGate directory on the source and target systems.
11. (DB2 for i) Run `ggos400install` without arguments. No arguments are necessary for an upgrade, however, if you change the library, the old library is left on the system until you remove it. For more information about `ggos400install`, see Prerequisites for Installing Oracle GoldenGate for DB2 for i.
12. > **Note:**
    >
    > (Only for the Oracle GoldenGate for SQL Server Extract) Before performing this step, review the steps for Patching Oracle GoldenGate for SQL Server - Extract Requirements .

    In GGSCI, start the Oracle GoldenGate processes on the source and target systems in the following order:

    ```
    GGSCI> START MANAGER
    GGSCI> START EXTRACT group
    GGSCI> START EXTRACT pump
    GGSCI> START REPLICAT group
    ```

13. (Source for MySQL with DDL replication enabled) Restart the `metadata_server` by executing the following:

    ```
    ./ddl_install.sh start user-id  password  port-number
    ```

Also see:

- For MySQL 5.7, see instructions for patching in Patching Oracle GoldenGate MySQL 5.7 with DDL Replication Enabled
- For SQL Server, see Patching Oracle GoldenGate for SQL Server - Extract Requirements

Patching Oracle GoldenGate for SQL Server - Extract Requirements

You must follow the existing patching procedures in Patching Oracle GoldenGate for Non-Oracle Databases. In addition, you must re-run `ADD TRANDATA` for each table that is already enabled for `TRANDATA` using these steps:

1. Stop all Oracle GoldenGate processes.
2. Follow normal patch procedures for binary replacement but do not start any Oracle GoldenGate processes.

3. Manually stop the SQL Server CDC Capture job for the database. If the job is processing a large transaction, it may take some time before it actually stops.
4. Ensure that the Extract is stopped.
5. Using GGSCI, run `ADD TRANDATA` again for every table that you previously enabled it for, including the heartbeat tables and any Replicat checkpoint table used as a `FILTERTABLE` object for active/active configurations.

> **Note:**
>
> Do not run the `DELETE TRANDATA` command.

6. Manually restart the SQL Server CDC Capture job.
7. Manually restart the Oracle GoldenGate processes such as Extract, Replicat, and Manager.

Patching Oracle GoldenGate MySQL 5.7 with DDL Replication Enabled

To patch Oracle GoldenGate MySQL 5.7 with DDL replication enabled:

1. Stop the metadata server using the following DDL install script `stop` option.

   ```
   ./ddl_install.sh stop user-id  password  port-number
   ```

2. Replace the `metadata_server` executable in the installation directory.
3. Start the metadata server running currently using ddl install script start option:

   ```
   ./ddl_install.sh start user-id  password  port-number
   ```

> **Note:**
>
> The DDL operations issued in between starting and stopping the `metadata_server` would be lost.

Uninstalling the Patch for Oracle and Non-Oracle Databases Using OPatch

To uninstall the patch, follow these steps:

1. Install the latest OPatch version, set the required environment variables, and stop the Oracle GoldenGate processes and services. The patch installation steps are documented in the previous topic.
2. Navigate to the *`patch_top_dir/patch_number`* directory:

   ```
   $ cd patch_top_dir/patch_number
   ```

3. Uninstall the patch by running the following command:

   ```
   $ opatch rollback -id patch_number
   ```

4. Start the services and processes from the Oracle GoldenGate home.

Uninstalling Oracle GoldenGate Classic Architecture for Oracle Database

Learn about uninstallling Oracle GoldenGate Classic Architecture processes and files from the host in Linux, UNIX, and Windows environments.

It is assumed that you no longer need the data in the Oracle GoldenGate trails, and that you no longer need to preserve the current Oracle GoldenGate environment. To preserve your current environment and data, make a backup of the Oracle GoldenGate directory and all subdirectories before starting this procedure.

Stopping Processes

This procedure stops the Extract and Replication processes. Leave Manager running until directed to stop it.

On all Systems:

1. Run the command shell.
2. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
3. Change directories to the Oracle GoldenGate installation directory.
4. Run `ggsci`.
5. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

6. Stop the Manager process.

   ```
   STOP MANAGER
   ```

Removing the DDL Environment

(Valid when the DDL trigger is being used to support DDL replication.) This procedure removes all of the Oracle GoldenGate DDL objects from the DDL schema on a source system.

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
2. Run `ggsci` from your Oracle GoldenGate directory.
3. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

4. Log in to SQL*Plus as a user that has `SYSDBA` privileges.
5. Disconnect all sessions that ever issued DDL, including those of Oracle GoldenGate processes, SQL*Plus, business applications, and any other software that uses Oracle. Otherwise the database might generate an ORA-04021 error.
6. Run the `ddl_disable` script to disable the DDL trigger.
7. Run the `ddl_remove` script to remove the Oracle GoldenGate DDL trigger, the DDL history and marker tables, and other associated objects. This script produces a

`ddl_remove_spool.txt` file that logs the script output and a `ddl_remove_set.txt` file that logs environment settings in case they are needed for debugging.

8. Run the `marker_remove` script to remove the Oracle GoldenGate marker support system. This script produces a `marker_remove_spool.txt` file that logs the script output and a `marker_remove_set.txt` file that logs environment settings in case they are needed for debugging.

Removing Database Objects

Follow these instructions to remove supplemental logging and any Oracle GoldenGate CDC Cleanup objects (for SQL Server) from the source database in the Oracle GoldenGate Extract configuration, and to remove the checkpoint table in the Replicat configuration. Specific steps and commands may not apply to your configuration.

On a Source System:

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
2. Run `ggsci` from your Oracle GoldenGate directory.
3. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

4. Stop the Manager process.

   ```
   STOP MANAGER
   ```

5. In GGSCI, log into the database with the `DBLOGIN` (or the `MININGDBLOGIN` command if you need to remove a database logmining server from a downstream mining database). `[MINING]DBLOGIN` requires privileges granted in the `dbms_goldengate_auth.grant_admin_privilege` procedure.

   ```
   [MINING]DBLOGIN USERIDALIAS alias
   ```

6. In GGSCI, run any or all of the following commands, depending on your configuration.
   - Disable schema-level supplemental logging (wildcards are not allowed):

     ```
     DELETE SCHEMATRANDATA schema [NOSCHEDULINGCOLS | ALLCOLS]
     ```

   - Disable table-level supplemental logging.

     ```
     DELETE TRANDATA [container.]schema.table  [NOSCHEDULINGCOLS | ALLCOLS]
     ```

   - (Bidirectional configuration) Remove the Oracle trace table.

     ```
     DELETE TRACETABLE [container.]schema.table
     ```

   - (Classic capture configuration) Disable log retention. `DBLOGIN` requires privileges shown in Log Retention Options.

     ```
     UNREGISTER EXTRACT group LOGRETENTION
     ```

- (Integrated capture configuration) Remove the database logmining server from an Oracle mining database.

  ```
  DELETE EXTRACT group
  UNREGISTER EXTRACT group DATABASE
  ```

7. Run the following Oracle procedure to remove the privileges from the Oracle GoldenGate administration users for both classic and integrated processes.

   ```
   dbms_goldengate_auth.revoke_admin_privilege('ggadm')
   ```

On a Target System:

1. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

2. Log into the database.

   ```
   DBLOGIN USERIDALIAS alias
   ```

3. (Integrated Replicat) Delete the Replicat group, which also deletes the inbound server from the target database.

   ```
   DELETE REPLICAT group
   ```

4. (Nonintegrated Replicat) Remove the Replicat checkpoint table by running the `DELETE CHECKPOINTTABLE` command.

   ```
   DELETE CHECKPOINTTABLE [container.]schema.table
   ```

Uninstalling Oracle GoldenGate Using Oracle Universal Installer

Follow these instructions to uninstall Oracle GoldenGate through an interactive session of Oracle Universal Installer (OUI).

> **WARNING:**
>
> Before removing Oracle GoldenGate through OUI, follow the instructions in Removing the DDL Environment (if using trigger-based DDL capture) and Removing Database Objects. These procedures require the use of Oracle GoldenGate commands and scripts, which are removed by the OUI uninstaller.

The following items are removed in this process.

- The Oracle GoldenGate home directory in the Oracle central inventory.
- The Oracle GoldenGate installation directory.
- The Oracle GoldenGate Manager service, if installed on Windows.
- The Oracle GoldenGate Windows Registry entries

To remove Oracle GoldenGate from the system:

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
2. Run `ggsci` from your Oracle GoldenGate directory.
3. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

4. Stop the Manager process.

   ```
   STOP MANAGER
   ```

5. Run the following script from the Oracle GoldenGate installation directory.

   UNIX and Linux:

   ```
   OGG_HOME/deinstall/deinstall.sh
   ```

   Windows:

   ```
   OGG_HOME/deinstall/deinstall.bat
   ```

Uninstalling Oracle GoldenGate Manually

Follow these instructions to remove the Oracle GoldenGate environment from the system manually through the operating system.

Manually Removing Oracle GoldenGate Windows Components

This procedure:

- Removes Oracle GoldenGate as a Windows cluster resource from a source or target Windows system
- Stops Oracle GoldenGate events from being reported to the Windows Event Manager
- Removes the Manager service

Perform these steps on source and target systems:

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and to delete files and directories from the operating system.
2. (Cluster) Working from the node in the cluster that owns the cluster group that contains the Manager resource, run `ggsci` and make certain that all Extract and Replicat processes are stopped. Stop any that are running.

   ```
   STATUS ER *
   ```

   ```
   STOP ER *
   ```

3. (Cluster) Use the Cluster Administrator tool to take the Manager resource offline.
4. (Cluster) Right click the resource and select **Delete** to remove it.

5. Click **Start** then **Run**, and then type `cmd` in the Run dialog box to open the command console.
6. Change directories to the Oracle GoldenGate installation directory.
7. Run the `INSTALL` utility with the following syntax.

   ```
   INSTALL DELETEEVENTS DELETESERVICE
   ```

8. (Cluster) Move the cluster group to the next node in the cluster, and repeat from Step 5.

Manually Removing the Oracle GoldenGate Files

These steps apply when the Oracle GoldenGate installation isn't done using the Oracle GoldenGate installer.

Perform these steps on all systems to remove the Oracle GoldenGate installation directory:

> **Note:**
>
> If Oracle GoldenGate has been installed using the installer, then you must uninstall Oracle GoldenGate with the uninstall script as well. Otherwise, you will have orphaned fragments within `ora inventory`.

1. In GGSCI, verify that all processes are stopped. Stop any that are running.

   ```
   STOP ER *
   STATUS ER *
   STOP MANAGER
   STATUS MANAGER
   ```

2. Exit GGSCI.

   ```
   EXIT
   ```

3. Remove the Oracle GoldenGate installation directory.

Uninstalling Oracle GoldenGate Classic Architecture for Non-Oracle Databases

Learn how to uninstall Oracle GoldenGate for non-Oracle databases.

Stopping Processes

This procedure stops the Extract and Replication processes. Leave Manager running until directed to stop it.

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
2. Run GGSCI from the Oracle GoldenGate directory.
3. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

Removing Oracle GoldenGate Database Objects

Use the following instructions to remove database objects and stopping processes for your configuration. Some steps and commands may not apply to your configuration, however other instructions are applicable to all databases (until specified).

For SQL Server, use these steps to remove supplemental logging and any Oracle GoldenGate CDC Cleanup objects from the source database in the Oracle GoldenGate capture configuration, and to remove the Replicat checkpoint table in the apply configuration.

On a Source System:

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and delete files and directories from the operating system.
2. Run `ggsci` from the Oracle GoldenGate directory.
3. Stop all Oracle GoldenGate processes if not already done.

   ```
   STOP EXTRACT *
   ```

4. Stop the Manager process.

   ```
   STOP MANAGER
   ```

5. Issue the following command to log into the source database, see SOURCEDB.

   ```
   DBLOGIN SOURCEDB {data_source | database@host:port} USERIDALIAS alias
   ```

6. Remove any heartbeat table entries by running the `DELETE HEARTBEATTABLE` command.
7. For a SQL Server Extract configuration, remove the Oracle GoldenGate CDC cleanup job and objects if they were created.
   a. Open a command prompt and change to the Oracle GoldenGate installation folder.
   b. Run the `ogg_cdc_cleanup_setup.sh/bat` file as follows:

      ```
      ogg_cdc_cleanup_setup.sh/bat dropJob userid password database_name
      servername\instancename schema
      ```

      The *`userid password`* must be a valid SQL Server login and password for a `sysadmin` user. *`database_name servername\instancename`* are the source database name and instance name. If only server name is listed, then the default instance will be used to connect to the database server. *`schema`* is the schema name listed in the `GLOBALS` file, with the `GGSCHEMA` parameter.

      For example:

      ```
      ogg_cdc_cleanup_setup.bat dropJob ggsuser ggspword db1 server1\inst1 ogg
      ```

8. Remove supplemental logging from tables that were enabled with it. See `DELETE TRANDATA`. Remove supplemental logging for any filter tables used for bi-directional replication as well. You can use a wildcard to specify multiple table names.

   ```
   DELETE TRANDATA owner.table
   ```

9. For PostgreSQL, the registered replication slot must be deleted after removing the Extract, otherwise the database logs will continue to grow.

   ```
   DELETE EXTRACT extname
   UNREGISTER EXTRACT extname
   ```

On a Target System:

1. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

2. Issue the following command to log into the target database. See SOURCEDB.

   ```
   DBLOGIN SOURCEDB {data_source | database@host:port} USERIDALIAS alias
   ```

3. Remove the Replicat checkpoint tables and heartbeat by running the `DELETE CHECKPOINTTABLE` and `DELETE HEARTBEATTABLE` commands.

   ```
   DELETE CHECKPOINTTABLE schema.table


   DELETE HEARTBEATTABLE
   ```

Uninstalling Oracle GoldenGate from a Source DB2 for i System

1. Ensure that all Oracle GoldenGate processes are stopped, and any database objects are removed, based on instructions provided in Removing Database Objects.
2. Delete the Oracle GoldenGate library. Specify `I` (ignore) for any prompts about unsaved journal receivers.

   ```
   clrlib library
   dltlib library
   ```

Uninstalling Oracle GoldenGate from a Linux System

Follow these instructions to remove Oracle GoldenGate from a Linux system.

1. Run the command shell of the operating system.
2. Ensure all Oracle GoldenGate processes are stopped, and any database objects have been removed, based on the instructions in Removing Oracle GoldenGate Database Objects.
3. Remove the Oracle GoldenGate files by removing the installation directory.

Uninstalling Oracle GoldenGate from a Windows System

Follow these instructions to remove Oracle GoldenGate from a Windows system.

1. Log on to the operating system as the system administrator or as a user with permission to issue Oracle GoldenGate commands and to delete files and directories from the operating system.

2. Ensure all Oracle GoldenGate processes are stopped, and any database objects have been removed based on instructions in Removing Database Objects.
3. (Windows Cluster) Use the Cluster Administrator tool to take the Manager resource offline.
4. (Windows Cluster) Right click the resource and select **Delete** to remove it.
5. Click **Start**, **Run**, and then type `cmd` in the **Run** dialog box to open the command console.
6. Change directories to the Oracle GoldenGate installation directory.
7. Remove the Manager service and events using the `INSTALL` utility with the following syntax:

   ```
   INSTALL DELETEEVENTS DELETESERVICE
   ```

8. (Windows Cluster) Move the cluster group to the next node in the cluster and repeat the process from step 6.
9. Remove the Oracle GoldenGate files by removing the installation directory.

Removing Oracle GoldenGate from a Windows Cluster

1. Working from the node in the cluster that owns the Windows Cluster group that contains the Manager resource, run GGSCI and then stop any Extract and Replicat processes that are still running.
2. Use the Windows Cluster Administrator tool to take the Manager resource offline.
3. Right click the resource and select **Delete** to remove it.
4. Click **Start**, then **Run**, and type `cmd` in the Run dialog box to open the command console.
5. Change directories to the Oracle GoldenGate installation directory.
6. Run the INSTALL utility using the following syntax.

   ```
   install deleteevents deleteservice
   ```

   This command stops Oracle GoldenGate events from being reported to the Windows Event Manager and removes the Manager service.
7. Move the Windows Cluster group to the next node in the cluster.

Removing Oracle GoldenGate from a Remote Windows System

On all systems:

1. (Suggested) Log on to the operating system as the system administrator or as a user with permission to issue Oracle GoldenGate commands and to delete files and directories from the operating system.
2. From the Oracle GoldenGate installation folder, run GGSCI.
3. Stop all Oracle GoldenGate processes.

   ```
   STOP ER *
   ```

4. Stop the Manager process.

   ```
   STOP MANAGER
   ```

5. > **Note:**
   >
   > Skip Steps 5 through 8 if you already performed them when removing Oracle GoldenGate from a Windows Cluster.

6. Click **Start**, then **Run**, and type `cmd` in the Run dialog box to open the command console.
7. Change directories to the Oracle GoldenGate installation directory.
8. Run the INSTALL utility using the following syntax.

   ```
   install deleteevents deleteservice
   ```

   This command stops Oracle GoldenGate events from being reported to the Windows Event Manager and removes the Manager service.
9. Log into the database with the `DBLOGIN` command.

   ```
   DBLOGIN SOURCEDB database, USERID db_user [, PASSWORD pw [encryption options]]
   ```

   > **Note:**
   >
   > Only `BLOWFISH` encryption is supported for DB2 for i systems.

10. Remove the Replicat checkpoint table by running the `DELETE CHECKPOINTTABLE` command.

    ```
    DELETE CHECKPOINTTABLE owner.table
    ```

11. Remove the Oracle GoldenGate files by removing the installation directory.

Removing Oracle GoldenGate Windows Components

(Valid for Windows installations) This procedure does the following:

- Removes Oracle GoldenGate as a Windows cluster resource from a source or target Windows system.
- Stops Oracle GoldenGate events from being reported to the Windows Event Manager.
- Removes the Manager service.

Perform these steps on source and target systems.

1. Log on as the system administrator or as a user with permission to issue Oracle GoldenGate commands and to delete files and directories from the operating system.
2. Run GGSCI and make certain that all Extract and Replicat processes are stopped. Stop any that are running.

   ```
   STATUS ER *
   STOP ER *
   ```

3. (Cluster) Use the Cluster Administrator tool to take the Manager resource offline.
4. (Cluster) Right click the resource and select **Delete** to remove it.
5. Click **Start** then **Run**, and then type `cmd` in the Run dialog box to open the command console.

6. Change directories to the Oracle GoldenGate installation directory.
7. Run the `INSTALL` utility with the following syntax.

   ```
   install deleteevents deleteservice
   ```

8. (Cluster) Move the cluster group to the next node in the cluster, and repeat from step 5.

3

Prepare

Learn about the tasks for preparing databases for Oracle GoldenGate and prerequisites for connecting Oracle GoldenGate to databases before beginning the configuration of Extract and Replicat processes.

Prepare Your Database for Oracle GoldenGate Classic Architecture

Learn about the tasks for preparing databases for Oracle GoldenGate and prerequisites for connecting Oracle GoldenGate to databases before beginning the configuration of Extract and Replicat processes.

Db2 LUW

With Oracle GoldenGate for Db2 LUW, you can perform initial loads and capture transactional data from supported Db2 LUW database versions and replicate the data to a Db2 LUW database or other supported Oracle GoldenGate targets, such as an Oracle Database.

Oracle GoldenGate for Db2 LUW supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

Database User for Oracle GoldenGate Processes for DB2 LUW

- Create a database user that is dedicated to Oracle GoldenGate. It can be the same user for all of the Oracle GoldenGate processes that must connect to a database:
  - Extract (source database)
  - Replicat (target database)
  - `DEFGEN` (source or target database)
- To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, the Oracle GoldenGate database user. It is recommended that you store the login credentials in an Oracle GoldenGate credential store. The credential store makes use of local secure storage for the login names and passwords, and permits you to specify only an alias in the Oracle GoldenGate parameter files.
- Assign system administrator (`SYSADM`) or database administrator (`DBADM`) authority to the database user under which Extract runs. To give the Extract user `DBADM` authority, a user with `SYSADM` authority can issue the following grant statement.

  ```
  GRANT DBADM ON DATABASE TO USER user
  ```

  This authority can also be granted from the `User and Group Objects` folder in the DB2 Control Center. The database tab for the user that is assigned to an Oracle GoldenGate process should have the Database Administrative Authority box checked.

> **Note:**
>
> If the Extract user does not have the required authority, Extract will log the following errors and stop.

```
[SC=-1224:SQL1224N A database agent could not be started to
service a request, or was terminated as a result of a database
system shutdown or a force command.
SQL STATE 55032: The CONNECT statement is invalid, because the
database manager was stopped after this application was started]
```

- Grant at least the following privileges to the database user under which Replicat runs:
  - Local `CONNECT` to the target database
  - `SELECT` on the system catalog views
  - `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on the target tables

Database Configuration for DB2 LUW

- The Oracle GoldenGate Extract process calls the `DB2READLOG` function in the Administrative API to read the transaction log files of a DB2 LUW source database. In addition to `DB2READLOG`, Extract uses a small number of other API routines to check the source database configuration on startup.
- The Oracle GoldenGate Replicat process uses the DB2 CLI interface on a DB2 LUW target database. For instructions on installing this interface, see the DB2 documentation.
- The database can reside on a different server from the one where Oracle GoldenGate is installed, so long as the database is defined locally. For example, the following enables you to use database `mydb` locally with data that is on `abc123`:

  ```
  catalog tcpip node abc123 remote abc123.us.mycompany.com server 00000
  catalog db mydb as abc123 at node abc123 AUTHENTICATION server
  ```

- The DB2 Universal Database has an internal trace facility called `db2trc`, which acquires Interprocess Communication resources (IPC) (both semaphore and shared memory). Even though a DB2 trace is not turned on, it may issue `semget()` calls to the operating system. These calls fail since no IPC resources are acquired so you must issue the following command on the DB2 client:

  ```
  db2trc alloc
  ```

- For best performance for DB2 clients with a local database, Oracle recommends that you create a local node catalog instead of TCP/IP when connecting Oracle GoldenGate to a database that resides on the same machine. This is because local node uses IPC, which is much faster than a TCP/IP node that uses a socket API to access the local database.
- To connect to DB2 LUW remotely from another system, you must use the following driver packages from IBM:
  - IBM Data Server Runtime Client
  - IBM Data Server Driver Package (DS Driver)
  - IBM Data Server Client

  The IBM Data Server Driver for ODBC and CLI (CLI Driver) is not supported for DB2 LUW.

Setting the Session Character Set

To support the conversion of character sets between the source and target databases, make certain that the session character set is the same as the database character set. You can set the session character set with the `DB2CODEPAGE` environment variable.

Preparing Tables for Processing

The following table attributes must be addressed in an Oracle GoldenGate environment.

Triggers and Cascade Constraints Considerations

Triggers

Disable triggers on the target tables, or alter them to ignore changes made by the Oracle GoldenGate database user. Oracle GoldenGate replicates DML that results from a trigger. If the same trigger gets activated on the target table, then it becomes redundant because of the replicated version, and the database returns an error.

Cascade Constraints Considerations

Cascading updates and deletes captured by Oracle GoldenGate are not logged in binary log, so they are not captured. This is valid for both MySQL and MariaDB. For example, when you run the delete statement in the parent table with a parent child relationship between tables, the cascading deletes (if there are any) happens for child table, but they are not logged in binary log. Only the delete or update record for the parent table is logged in the binary log and captured by Oracle GoldenGate.

See https://mariadb.com/kb/en/replication-and-foreign-keys/ and https://dev.mysql.com/doc/refman/8.0/en/innodb-and-mysql-replication.html for details.

To properly handle replication of cascading operations, it is recommended to disable cascade deletes and updates on the source and code your application to explicitly delete or update the child records prior to modifying the parent record. Alternatively, you must ensure that the target parent table has the same cascade constraints configured as the source parent table, but this could lead to an out-of-sync condition between source and target, especially in cases of bi-directional replication.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Using `KEYCOLS` to Specify a Custom Key

If a table does not have one of the preceding types of row identifiers, or if you prefer those identifiers not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds.

Preventing Key Changes

Do not add columns to a key after Oracle GoldenGate starts extracting data from the table. This rule applies to a primary key, a unique key, a `KEYCOLS` key, or an all-column key. DB2 LUW does not supply a before image for columns that are added to a table. If any columns in a key are updated on the source, Oracle GoldenGate needs a before image to compare with the current values in the target table when it replicates the update.

Enabling Change Capture

Configure DB2 to log data changes in the expanded format that is supplied by the `DATA CAPTURE CHANGES` feature of the `CREATE TABLE` and `ALTER TABLE` commands. This format provides Oracle GoldenGate with the entire before and after images of rows that are changed by update statements. You can use GGSCI to issue the `ALTER TABLE` command as follows.

To Enable Change Capture from GGSCI:

1. From the Oracle GoldenGate directory, run GGSCI.
2. Log on to DB2 from GGSCI as a user that has `ALTER TABLE` privileges. Specify the data source name with `SOURCEDB` and specify the user login with `USERID` and `PASSWORD`.

   ```
   DBLOGIN SOURCEDB dsn, USERID user[, PASSWORD password]
   ```

3. Issue the following command. where *`owner.table`* is the fully qualified name of the table. You can use a wildcard to specify multiple table names. Only the asterisk (*) wildcard is supported for DB2 LUW.

   ```
   ADD TRANDATA owner.table
   ```

   `ADD TRANDATA` issues the following command, which includes logging the before image of `LONGVAR` columns:

   ```
   ALTER TABLE name DATA CAPTURE CHANGES INCLUDE LONGVAR COLUMNS;
   ```

Example 3-1 To Exclude `LONGVAR` Logging:

To omit the `INCLUDE LONGVAR COLUMNS` clause from the `ALTER TABLE` command, use `ADD TRANDATA` with the `EXCLUDELONG` option.

```
ADD TRANDATA owner.table, EXCLUDELONG
```

> **Note:**
>
> If `LONGVAR` columns are excluded from logging, the Oracle GoldenGate features that require before images, such as the `GETUPDATEBEFORES`, `NOCOMPRESSUPDATES`, and `NOCOMPRESSDELETES` parameters, might return errors if tables contain those columns. For a workaround, see the `REQUIRELONGDATACAPTURECHANGES` | `NOREQUIRELONGDATACAPTURECHANGES` options of the `TRANLOGOPTIONS` parameter.

Maintaining Materialized Query Tables

To maintain parity between source and target materialized query tables (MQT), you replicate the base tables, but not the MQTs. The target database maintains the MQTs based on the changes that Replicat applies to the base tables.

The following are the rules for configuring these tables:

- Include the base tables in your `TABLE` and `MAP` statements.
- Do not include MQTs in the `TABLE` and `MAP` statements.

- Wildcards can be used in `TABLE` and `MAP` statements, even though they might resolve MQT names along with regular table names. Oracle GoldenGate automatically excludes MQTs from wildcarded table lists. However, any MQT that is explicitly listed in an Extract `TABLE` statement by name will cause Extract to abend.

Creating a Temporal Table

A temporal table is a table that maintains the history of its data and the time period when its data are valid. Temporal tables are used in Oracle GoldenGate to keep track of all the old rows that are deleted or updated in the table. Temporal tables are also used to maintain the business validity of its rows and data. For example, Oracle GoldenGate keeps track of the time period during which a row is valid. There are three types of temporal tables, system-period, application-period, and bitemporal table.

Support for Temporal Tables

- Replication between system-period temporal tables and application-period temporal tables is not supported.
- Replication from a non-temporal table to a temporal table is not supported.
- Replication of temporal tables with the `INSERTALLRECORDS` parameter is not supported.
- Bidirectional replication is supported only with the default replication.
- CDR in bidirectional replication is not supported.
- CDR in application-period temporal tables is supported.

Replicating with Temporal Tables

You can choose one of the following methods to replicate a system-period or a bitemporal temporal table as follows:

- You can replicate a temporal table to another temporal table only; this is the default behavior. Oracle GoldenGate will not replicate the `SYSTEM_TIME` period and transaction id columns because these are automatically generated columns at the apply side. The database manager populates the columns in the target temporal table using the system clock time and with the default values. You can preserve the original values these columns then use any of the following:
  - Add extra timestamp columns in the target temporal table and map the columns accordingly. The extra columns are automatically added to the associated history table.
  - Use a non-temporal table at the apply side and map the columns appropriately. In this scenario, you will not be able to maintain the history table.
  - In a non-oracle configuration where the source is DB2 LUW and the target is a different database, you can either ignore the automatically generated columns or use an appropriate column conversion function to convert the columns value in the format that target database supports and map them to target columns accordingly.

  Or
- You can replicate a temporal table, with the associated history table, to a temporal and history table respectively then you must specify the replicate parameter, `DBOPTIONS SUPPRESSTEMPORALUPDATES`. You must specify both the temporal table and history table to be captured in the Extract parameter file. Oracle GoldenGate replicates the `SYSTEM_TIME` period and transactions id columns value. You must ensure that the database instance has the execute permission to run the stored procedure at the apply side.

Oracle GoldenGate cannot detect and resolve conflicts while using default replication as `SYSTEM_TIME` period and `transactionstart id` columns remains auto generated. These columns cannot be specified in `set` and `where` clause. If you use the `SUPPRESSTEMPORALUPDATES` parameter, then Oracle GoldenGate supports CDR.

Converting

You can convert an already existing table into a temporal table, which changes the structure of the table. This section describes how the structure of the tables changes. The following sample existing table is converted into all three temporal tables types in the examples in this section:.

```
Table policy_info
(
Policy_id char[4] not null primary key,
Coverage int not null
              )
And the tables contains the following initial rows
                   POLICY_ID       COVERAGE
                   -------------       -----------
                     ABC                12000
                     DEF                 13000
                     ERT                 14000
```

Example 1 Converting an existing table into System-period temporal table.
You convert the sample existing table into a system-period temporal table by adding `SYSTEM_PERIOD`, `transaction id` columns, and `SYSTEM_TIME` period as in the following:

```
ALTER TABLE policy_info
   ADD COLUMN sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE policy_info
   ADD COLUMN sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END;
ALTER TABLE policy_info
   ADD COLUMN ts_id TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE policy_info ADD PERIOD SYSTEM_TIME(sys_start, sys_end);
```

Then you create a history table for the new temporal table using one of the following two methods:

-
  ```
      CREATE TABLE hist_policy_info
  (
   policy_id      CHAR(4) NOT NULL,
  coverage      INT NOT NULL,
  sys_start     TIMESTAMP(12) NOT NULL ,
  sys_end       TIMESTAMP(12) NOT NULL,
  ts_id             TIMESTAMP(12) NOT NULL
              );
   ALTER TABLE hist_policy_info ADD RESTRICT ON DROP;
  ```
- `CREATE TABLE hist_policy_info LIKE policy_info with RESTRICT ON DROP;`

  The `RESTRICT ON DROP` clause will not allow the history table to get dropped while dropping system-period temporal table. Otherwise the history table gets implicitly dropped while dropping its associated temporal table. You can create a history table without `RESTRICT ON DROP.` A history table cannot be explicitly dropped.

  You should not use the `GENERATED ALWAYS` clause while creating a history table. The primary key of the system-period temporal table also does not apply here as there could be many updates for a particular row in the base table, which triggers many inserts into the history table for the same set of primary keys. Apart from these, the structure of a history table should be exactly same as its associated system-period temporal table. The history table must have the same number and order of columns as system-period temporal table. History table columns cannot explicitly be added, dropped, or changed. You must associate a system-period temporal table with its history table with the following statement:

```
 ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info.
```

The `GENERATED ALWAYS` columns of the table are the ones that are always populated by the database manager so you do not have any control over these columns. The database manager populates these columns based on the system time.

The extra added `SYSTEM_PERIOD` and `transaction id` columns will have default values for already existing rows as in the following:

```
POLICY_ID                                   COVERAGE
SYS_START
SYS_END                                                TS_ID
--------- ----------- -------------------------------
--------------------------------
-------------------------------------------------------------------------------
ABC             12000 0001-01-01-00.00.00.000000000000
9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
DEF             13000 0001-01-01-00.00.00.000000000000
9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
ERT             14000 0001-01-01-00.00.00.000000000000
9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
```

The associated history table is populated with the before images once you start updating the temporal table.

Example 2 Converting an existing table into application-period temporal table.
You can convert the sample existing table into application-period temporal table by adding time columns and a `BUSINESS_TIME` period as in the following:

```
ALTER TABLE policy_info ADD COLUMN bus_start DATE NOT NULL DEFAULT '10/10/2001'"
ALTER TABLE policy_info ADD COLUMN bus_end DATE NOT NULL DEFAULT '10/10/2002'
ALTER TABLE policy_info ADD PERIOD BUSINESS_TIME(bus_start, bus_end)
```

While adding time columns, you need to make sure that while entering business validity time values of the existing time columns, the `bus_start` column always has value lesser than `bus_end` because these columns specify the business validity of the rows.
The new application-period temporal table will look similar to:

```
POLICY_ID   COVERAGE    BUS_START  BUS_END
--------- ----------- ---------- -------------------------------
ERT             14000              10/10/2001  10/10/2002
DEF             13000             10/10/2001   10/10/2002
ABC             12000             10/10/2001   10/10/2002
```

Example 3 Converting an existing table into bitemporal table.
You can convert the sample existing table into bitemporal table by adding S`YSTEM_PERIOD`, time columns along with the `SYSTEM_TIME` and `BUSINESS_TIME` period as in the following:

```
ALTER TABLE policy_info
   ADD COLUMN sys_start TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN;
ALTER TABLE policy_info
   ADD COLUMN sys_end TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END;
```

```
ALTER TABLE policy_info
   ADD COLUMN ts_id TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS TRANSACTION START ID;
ALTER TABLE policy_info ADD PERIOD SYSTEM_TIME(sys_start, sys_end);

ALTER TABLE policy_info ADD COLUMN bus_start DATE NOT NULL DEFAULT '10/10/2001'"
ALTER TABLE policy_info ADD COLUMN bus_end DATE NOT NULL DEFAULT '10/10/2002'
ALTER TABLE policy_info ADD PERIOD BUSINESS_TIME(bus_start, bus_end)
```

While adding the time columns, you must make sure that while entering business validity time values of already existing time columns, the `bus_start` column always has value lesser than `bus_end` because these columns specify the business validity of the rows.
Then you create a history table for the new temporal table using one of the following two methods:

-
  ```
      CREATE TABLE hist_policy_info
  (
   policy_id     CHAR(4) NOT NULL,
  coverage     INT NOT NULL,
  sys_start    TIMESTAMP(12) NOT NULL ,
  sys_end      TIMESTAMP(12) NOT NULL,
  ts_id            TIMESTAMP(12) NOT NULL
              );
   ALTER TABLE hist_policy_info ADD RESTRICT ON DROP;
  CREATE TABLE hist_policy_info LIKE policy_info with RESTRICT ON DROP;
  ```
- The `RESTRICT ON DROP` clause will not allow the history table to get dropped while dropping system-period temporal table. Otherwise the history table gets implicitly dropped while dropping its associated temporal table. You can create a history table without `RESTRICT ON DROP.` A history table cannot be explicitly dropped.

  You should not use the `GENERATED ALWAYS` clause while creating a history table. The primary key of the system-period temporal table also does not apply here as there could be many updates for a particular row in the base table, which triggers many inserts into the history table for the same set of primary keys. Apart from these, the structure of a history table should be exactly same as its associated system-period temporal table. The history table must have the same number and order of columns as system-period temporal table. History table columns cannot explicitly be added, dropped, or changed. You must associate a system-period temporal table with its history table with the following statement:

  ```
   ALTER TABLE policy_info ADD VERSIONING USE HISTORY TABLE hist_policy_info.
  ```

  The `GENERATED ALWAYS` columns of the table are the ones that are always populated by the database manager so you do not have any control over these columns. The database manager populates these columns based on the system time.

  The extra added `SYSTEM_PERIOD` and `transaction id` columns will have default values for already existing rows as in the following:

  ```
  POLICY_ID                                   COVERAGE
  SYS_START
  SYS_END                                                TS_ID
  --------- ----------- --------------------------------
  --------------------------------
  -------------------------------------------------------------------------------
  ABC             12000 0001-01-01-00.00.00.000000000000
  9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
  DEF             13000 0001-01-01-00.00.00.000000000000
  ```

```
9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
ERT             14000 0001-01-01-00.00.00.000000000000
9999-12-30-00.00.00.000000000000 0001-01-01-00.00.00.000000000000
```

The associated history table is populated with the before images once you start updating the temporal table.

The extra added `SYSTEM_TIME` period, transaction id and time columns will have default values for already existing rows as in the following:

```
POLICY_ID COVERAGE    SYS_START                        SYS_END
TS_ID                            BUS_START         BUS_END
--------- ----------- -------------------------------- --------------------------------
-------------------------------- ---------- -------------------------------------
ABC   12000 0001-01-01-00.00.00.000000000000 9999-12-30-00.00.00.000000000000
0001-01-01-00.00.00.000000000000 10/10/2001 10/10/2002
DEF             13000 0001-01-01-00.00.00.000000000000 9999-12-30-00.00.00.000000000000
0001-01-01-00.00.00.000000000000 10/10/2001 10/10/2002
ERT             14000 0001-01-01-00.00.00.000000000000 9999-12-30-00.00.00.000000000000
0001-01-01-00.00.00.000000000000 10/10/2001 10/10/2002
```

The history table is populated with the before images once user starts updating the temporal table.

Example 4 Replication in Non-Oracle Environment.
In a non-oracle configuration in which you do not have temporal tables at the apply side, you can only replicate the system-period and bitemporal tables though *not* the associated history tables. While performing replication in this situation, you must take care of the `SYSTEM_PERIOD` and transaction id columns value. These columns will have some values that the target database might not support. You should first use the map conversion functions to convert these values into the format that the target database supports, and then map the columns accordingly.
For example, MySQL has a `DATETIME` range from `1000-01-01 00:00:00.000000` to `9999-12-31 23:59:59.999999`. You cannot replicate a timestamp value of `0001-01-01-00.00.00.000000000000` to MySQL. To replicate such values, you must convert this value into the MySQL `DATETIME` value `1000-01-01 00:00:00.000000`, and then map the columns. If you have the following row in the `policy_info` system-period table:

```
POLICY_ID                                   COVERAGE
SYS_START
SYS_END                                                TS_ID
--------- ----------- -------------------------------- --------------------------------
-------------------------------------------------------------------------------
ABC             12000 0001-01-01-00.00.00.000000000000 9999-12-30-00.00.00.000000000000
0001-01-01-00.00.00.000000000000
```

To replicate the row into MySQL, you would use the `colmap()` function:

```
map source_schema.policy_info, target target_schema.policy_info colmap
(policy_id=policy_id, coverage=coverage, sys_start= @IF( ( @NUMSTR( @STREXT(sys_
start,1,4))) > 1000, sys_start, '1000-01-01 00.00.00.000000'), sys_end=sys_end,
 ts_id= @IF( ( @NUMSTR( @STREXT(ts_id,1,4))) > 1000, ts_id, '1000-01-01
 00.00.00.000000'));
```

Creating a Checkpoint Table

The checkpoint table is a required component of Replicat.

Replicat maintains its recovery checkpoints in the checkpoint table, which is stored in the target database. Checkpoints are written to the checkpoint table within the Replicat transaction. Because a checkpoint either succeeds or fails with the transaction, Replicat ensures that a transaction is only applied once, even if there is a failure of the process or the database.

To configure a checkpoint table, see Creating a Checkpoint Table in *Administering Oracle GoldenGate*.

Configuring the Replicat Parameter File

These steps configure the Replicat process. This process applies replicated data to a DB2 LUW target database.

1. In GGSCI on the target system, create the Replicat parameter file.

   ```
   EDIT PARAMS name
   ```

   Where: `name` is the name of the Replicat group.

2. Enter the Replicat parameters in the order shown, starting a new line for each parameter statement.

   Basic parameters for the Replicat group:

   ```
   REPLICAT financer
   TARGETDB mytarget, USERIDALIAS myalias
   ASSUMETARGETDEFS
   MAP hr.*, TARGET hr2.*;
   ```

Parameter	Description
`REPLICAT group`	`group` is the name of the Replicat group.
`TARGETDB database, USERIDALIAS alias`	Specifies the real name of the target DB2 LUW database (not an alias), plus the alias of the database login credential of the user that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Database User for Oracle GoldenGate Processes.
`ASSUMETARGETDEFS`	Specifies how to interpret data definitions. `ASSUMETARGETDEFS` assumes the source and target tables have identical definitions. (This procedure assume identical definitions.) Use the alternative `SOURCEDEFS` if the source and target tables have different definitions, and create a source data-definitions file with the `DEFGEN` utility.
`MAP schema.object, TARGET schema.object;`	Specifies the relationship between a source table or multiple objects, and the corresponding target object or objects. • `MAP` specifies the source portion of the `MAP` statement and is a required keyword. Specify the source objects in this clause. • `TARGET` specifies the target portion of the `MAP` statement and is a required keyword. Specify the target objects to which you are mapping the source objects. • `schema` is the schema name or a wildcarded set of schemas. • `object` is the name of a table or a wildcarded set of objects. Terminate this parameter statement with a semi-colon. Note that only the asterisk (*) wildcard is supported for DB2 LUW. The question mark (?) wildcard is not supported for this database. To exclude objects from a wildcard specification, use the `MAPEXCLUDE` parameter.

3. Enter any optional Replicat parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command in GGSCI.
4. Save and close the file.

Configuring the Transaction Logs for Oracle GoldenGate

To capture DML operations, Oracle GoldenGatereads the DB2 LUW online logs by default. However, it reads the archived logs if an online log is not available. To ensure the continuity and integrity of Oracle GoldenGateprocessing, configure the logs as follows.

Retaining the Transaction Logs

Configure the database to retain the transaction logs for roll forward recovery by enabling one of the following parameter sets, depending on the database version.

- DB2 LUW 9.5 and later:

  Set the `LOGARCHMETH` parameters as follows:

  – Set `LOGARCHMETH1` to `LOGRETAIN`.

  – Set `LOGARCHMETH2` to `OFF`.

  Alternatively, you can use any other `LOGARCHMETH` options, as long as forward recovery is enabled. For example, the following is valid:

  – Set `LOGARCHMETH1` to `DISK`.

  – Set `LOGARCHMETH2` to `TSM`.

To determine the log retention parameters:

1. Connect to the database.

   ```
   db2 connect to database user username using password
   ```

2. Get the database name.

   ```
   db2 list db directory
   ```

3. Get the database configuration for the database.

   ```
   db2 get db cfg for database
   ```

   The fields to view are:

   ```
   Log retain for recovery status = RECOVERY
   User exit for logging status = YES
   ```

To set the log retention parameters:

1. Issue one of the following commands.

   To enable `USEREXIT`:

   ```
   db2 update db cfg for database using USEREXIT ON
   ```

   If not using `USEREXIT`, use this command:

   ```
   db2 update db cfg for database using LOGRETAIN RECOVERY
   ```

   To set `LOGARCHMETH`:

```
db2 update db cfg for database using LOGARCHMETH1 LOGRETAIN
db2 update db cfg for database using LOGARCHMETH2 OFF
```

2. Make a full backup of the database by issuing the following command.

   ```
   db2 backup db database to device
   ```

3. Place the backup in a directory to which DB2 LUW has access rights. If you get the following message, contact your systems administrator:

   ```
   SQL2061N An attempt to access media "device" is denied.
   ```

Specifying the Archive Path

Set the DB2 LUW `OVERFLOWLOGPATH` parameter to the archive log directory. The node attaches automatically to the path variable that you specify.

```
db2 connect to database
db2 update db cfg using overflowlogpath "path"
```

Exclude the node itself from the path. For example, if the full path to the archive log directory is `/sdb2logarch/oltpods1/archive/OLTPODS1/NODE0000`, then the `OVERFLOWLOGPATH` value should be specified as `/sdb2logarch/oltpods1/archive/OLTPODS1`.

Understanding What's Supported for DB2 LUW

This chapter contains information on database and table features supported by Oracle GoldenGate for DB2 LUW.

Supported DB2 LUW Data Types

Oracle GoldenGate supports all DB2 LUW data types, except those listed in Non-Supported DB2 LUW Data Types.

Limitations of Support

Oracle GoldenGate has the following limitations for supporting DB2 LUW data types:

- Oracle GoldenGate supports multi-byte character data types and multi-byte data stored in character columns. Multi-byte data is only supported in a like-to-like configuration. Transformation, filtering, and other types of manipulation are not supported for multi-byte character data.
- `BLOB` and `CLOB` columns must have a `LOGGED` clause in their definitions.
- Due to limitations in the IBM DB2READLOG interface, Oracle GoldenGate does not support coordination of transactions across nodes in a DB2 Database Partitioning Feature (DPF) environment. In DPF, a transaction may span multiple nodes, depending upon how the data is partitioned.

  However, if you need to capture from it, you can do it with certain limitations. Check the Oracle Support note Does Oracle GoldenGate Support DB2 LUW Data Partitioning Feature (DPF)? (DocID 2763006.1)

  .
- `GRAPHIC` and `VARGRAPHIC` columns must be in a database, where the character set is UTF16. Any other character set causes the Oracle GoldenGate to abend.
- The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review

the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.

- Extract fully supports the capture and apply of `TIMESTAMP(0)` through `TIMESTAMP(12)` when the output trail format is 19.1 or higher. Otherwise Extract truncates data from `TIMESTAMP(10)` through `TIMESTAMP(12)` to nanoseconds (maximum of nine digits of fractional time) and issues a warning to the report file.
- Oracle GoldenGate supports timestamp data from 0001/01/03:00:00:00 to 9999/12/31:23:59:59. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the timezone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.
- Oracle GoldenGate does not support the filtering, column mapping, or manipulation of large objects that are larger than 4K. You can use the full Oracle GoldenGate functionality for objects that are 4K or smaller.
- Replication of XML columns between source and target databases with the *same* character set is supported. If the source and target database character sets are different, then XML replication may fail with a database error because some characters may not be recognized (or valid) in the target database character set.

Non-Supported DB2 LUW Data Types

The non-supported DB2 LUW data types are:

- `XMLType`
- User-defined types
- Negative dates

Supported Objects and Operations for DB2 LUW

Object and operations that are supported for DB2 LUW are:

- Oracle GoldenGate Extract supports cross-endian capture where the database and Oracle GoldenGate are running on different byte order servers. The byte order is detected automatically for DB2 LUW version 10.5 or higher. If the DB2 database auto-detection on the DB2 LUW 10.5 database is not required then you can override it by specifying the `TRANLOGOPTIONS MIXEDENDIAN [ON|OFF]` parameter. For DB2 LUW version 10.1, this parameter must be used in the Extract parameter file for cross-endian capture. See `TRANLOGOPTIONS` in *Parameters and Functions Reference for Oracle GoldenGate*.
- DB2 pureScale environment is supported.
- Oracle GoldenGate supports the maximum number of columns and column size per table that is supported by the database.
- `TRUNCATE TABLE`.
- Multi-Dimensional Clustered Tables (MDC).
- Materialized Query Tables. Oracle GoldenGate does not replicate the MQT itself, but only the base tables. The target database automatically maintains the content of the MQT based on the changes that are applied to the base tables by Replicat.
- Tables with `ROW COMPRESSION`. In DB2 LUW version 10.1 and later, `COMPRESS YES STATIC` is supported and `COMPRESS YES ADAPTIVE` are supported.
- Extended row size feature is enabled by default. It is supported with a workaround using `FETCHCOLS`. For any column values that are `VARCHAR` or `VARGRAPHIC` data types and are

stored out of row in the database, you must fetch these extended rows by specifying these columns using the `FETCHCOLS` option in the `TABLE` parameter in the extract parameter file. With this option set, when the column values are out of row then Oracle GoldenGate will fetch its value. If the value is out of and `FETCHCOLS` is *not* specified then Extract will abend to prevent any data loss. If you do not want to use this feature, set the `extended_row_size` parameter to `DISABLE`.

Extended row size feature is enabled, by default. It is supported with a workaround using `FETCHCOLS` for DB2 LUW 10.1. For any column values that are `VARCHAR` or `VARGRAPHIC` data types and are stored out of row in the database, you must fetch these extended rows by specifying these columns using the `FETCHCOLS` option in the `TABLE` parameter in the Extract parameter file. With this option set, when the column values are out of row, then Oracle GoldenGate fetches its value. If the value is out of and `FETCHCOLS` is not specified then Extract abends to prevent any data loss. If you do not want to use this feature, set the `extended_row_size parameter` to `DISABLE`. For DB2 LUW 10.5 and higher out of row values are captured seamlessly by Extract. `FETCHCOLS` is no more needed to capture out of row columns from these database versions.

- Temporal tables with DB2 LUW 10.1 FixPack 2 and greater are supported. This is the default for Replicat.
- Supported options with `SHOWTRANS`

  ```
  SHOWTRANS [transaction_ID]  [COUNT n]
  [DURATION duration unit]
  [TABULAR][FILE file_name] |
  ```

- Options with `SKIPTRANS` and `FORCETRANS`.

  ```
  SKIPTRANS transaction_ID
  [FORCE] FORCETRANS transaction_ID  [FORCE]
  ```

- Limitations on Automatic Heartbeat Table support are as follows:
  - `[THREAD n] [DETAIL]` is not supported.
  - Oracle GoldenGate heartbeat parameters frequency and purge frequency are accepted in seconds and days. However, the DB2 LUW task scheduler accepts its schedule only in cron format so the Oracle GoldenGate input value to cron format may result in some loss of accuracy. For example:

    ```
    ADD HEARTBEATTABLE, FREQUENCY 150, PURGE_FREQUENCY 20
    ```

    This example sets the `FREQUENCY` to 150 seconds, which is converted to the closest minute value of 2 minutes, so the heartbeat table is updated every 120 seconds instead of every 150 seconds. Setting `PURGE_FREQUENCY` to 20 means that the history table is purged at midnight on every 20th day.
  - The following are steps are necessary for the heartbeat scheduled tasks to run:
    1. Set the `DB2_ATS_ENABLE` registry variable to `db2set DB2_ATS_ENABLE=YES`.
    2. Create the `SYSTOOLSPACE` tablespace if it does not already exist:

       ```
       CREATE TABLESPACE SYSTOOLSPACE IN IBMCATGROUP MANAGED BY AUTOMATIC STORAGE
       EXTENTSIZE 4
       ```

    3. Ensure instance owner has Database administration authority (DBADM):

       ```
       GRANT DBADM ON DATABASE TO instance_owner_name
       ```

Non-Supported Objects and Operations for DB2 LUW

Objects and operations for DB2 LUW that are not supported by Oracle GoldenGate are:

- Schema, table or column names that have trailing spaces
- Multiple instances of a database
- Datalinks
- Extraction or replication of DDL (data definition language) operations
- Generated columns (`GENERATE ALWAYS` clause)

> **Note:**

- DB2 Data Partitioning Feature (DPF) is not supported. DPF doesn't provide a way to read the log records in a coordinated fashion across all the nodes in a partition. So, there is no way to ensure that even if all nodes are being replicated with separate Extracts, it would be possible to ensure that all records from all transactions are applied in the correct temporal order. This may occur due to a number of factors including caching in the database and the underlying operating system not allowing the Extracts to have visibility into whether there are any cached and not yet visible entries that may affect the ordering of the record operations across all partitions. Due to this uncertainty, it is not possible to ensure that transactions that span partitions, or primary key updates can be replicated in a consistent manner.

System Schemas

The following schemas or objects are not be automatically replicated by Oracle GoldenGate unless they are explicitly specified without a wildcard.

- `"SYSIBM"`
- `"SYSCAT"`
- `"SYSSTAT"`
- `"SYSPROC"`
- `"SYSFUN"`
- `"SYSIBMADMIN"`
- `"SYSTOOLS"`
- `"SYSPUBLIC"`

Supported Object Names

For a list of characters that are supported in object names, see Supported Database Object Names in *Administering Oracle GoldenGate*.

Db2 for i

With Oracle GoldenGate for Db2 for i, you can perform initial loads and capture transactional data from supported Db2 for i versions and replicate the data to a Db2 for i database or other supported Oracle GoldenGate targets, such as an Oracle Database.

Oracle GoldenGate for Db2 for i supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

Preparing the System for Oracle GoldenGate

This chapter contains guidelines for preparing the DB2 for i system to support Oracle GoldenGate.

Preparing the Journals for Data Capture by Extract

All tables for which you want data to be captured must be journaled, either explicitly or by default by means of a `QSQJRN` journal in the same library. To preserve data integrity, data journal entries are sent to the Extract process in time order as they appear on the system. This section provides guidelines for configuring the journals to support capture by the Extract process.

Allocating Journals to an Extract Group

One Extract process can process a single journal. If using more journals than that, use additional Extract processes to handle the extra journals. You can also use additional Extract processes to improve capture performance if necessary.

> **Note:**
>
> To ensure transaction integrity, all objects that correspond to any given transaction must be read by the same Extract group. For more information about using multiple Extract processes, see Tuning the Performance of Oracle GoldenGate in *Administering Oracle GoldenGate*.

Setting Journal Parameters

To support the capture of data by the Extract process, the following are the minimal journaling parameter settings that are required.

- Manage Receivers `(MNGRCV): *SYSTEM`
- Delete Receivers `(DLTRCV): *NO`
- Receiver Size Option `(RCVSIZOPT): *MAXOPT2` (`*MAXOPT3` recommended to avoid the necessity to perform an `ALTER EXTRACT` with the `ETROLLOVER` option when the journal sequence numbers run out if `*MAXOPT2` is used.)
- Journal State `(JRNSTATE): *ACTIVE`
- Minimize Entry Specific Data `(MINENTDTA): *NONE`
- Fixed Length Data `(FIXLENDTA): *USR`

In the following example, the command to set these attributes for a journal `JRN1` in library `LIB1` would be:

```
CHGJRN JRN(LIB1/JRN1) MNGRCV(*SYSTEM) DLTRCV(*NO) RCVSIZOPT(*MAXOPT3)
JRNSTATE(*ACTIVE) MINENTDTA(*NONE) FIXLENDTA(*USR)
```

Note:

To check the attributes of a journal, use the command `WRKJRNA JRN(LIB1/JRN1) DETAIL(*CURATR)`.

When the journaling is set to the recommended parameter settings, you are assured that the entries in the journals contain all of the information necessary for Oracle GoldenGate processing to occur. These settings also ensure that the system does not delete the journal receivers automatically, but retains them in case Extract needs to process older data.

Deleting Old Journal Receivers

Although the `DLTRCV` parameter is set to `NO` in the recommended configuration for Extract, you can delete old journal receivers manually once Extract is finished capturing from them.

If using another application that is using the journals that Oracle GoldenGate will be reading, consideration must be given regarding any automatic journal receiver cleanup that may be in place. Oracle GoldenGate must be able to read the journal receivers before they are detached or removed.

To Delete Journal Receivers

1. Run GGSCI.
2. In GGSCI, issue the following command to view the journal positions in which Extract has current processing points, along with their journal receivers.

   ```
   INFO EXTRACT group
   ```

3. Use the following DB2 for i command to delete any journal receivers that were generated prior to the ones that are shown in the `INFO EXTRACT` command.

   ```
   DLTJRNRCV JRNRCV(library/journal_receiver)
   ```

   Where:

   `library` and *`journal_receiver`* are the actual names of the library and journal receiver to be deleted. See the DB2 for i Information Center for more information about this command.

Using Remote Journal

Learn about remote journal preparation and adding a remote journal.
Remote Journal support in the IBM DB2 for i operating system provides the ability for a system to replicate, in its entirety, a sequence of journal entries from one DB2 for i system to another. Once setup, this replication is handled automatically and transparently by the operating system. The entries that are replicated are placed in a journal on the target system that is available to be read by an application in the same way as on the source system.

You must have an understanding of how to setup and use remote journaling on an DB2 for i system to use this feature with Oracle GoldenGate. There are no special software requirements for either Oracle GoldenGate or the DB2 for i systems to use remote journaling.

Preparing to Use Remote Journals

Before establishing the remote journal environment, complete the following steps:

1. Determine the extent of your remote journal network or environment.

2. *Library redirection* is the ability to allow the remote journal and associated journal receivers to reside in different libraries on the target system from the corresponding source journal and its associated journal receivers.

   Determine what library redirection, if any, you will be using for the remote journals and associated journal receivers.

3. Ensure that all selected libraries exist on the target systems. You must consider whether or not library redirection will be used when adding the remote journal.
4. Create the appropriate local journal if it does not already exist.
5. Configure and activate the communications protocol you have chosen to use.
6. After you have configured the communications protocol, it must be active while you are using the remote journal function.

   For example, if you are using the OptiConnect for IBM i bus transport method, then the OptiConnect for IBM i subsystem, QSOC, must be active. QSOC must be active for both the source system and the target system, and the appropriate controllers and devices must be varied on. If you are using a SNA communications transport, vary on the appropriate line, controller, and devices and ensure subsystem QCMN is active on both systems. Start of change If you are using TCP/IP or Sockets IPv6, you must start TCP/IP by using the Start TCP/IP (`STRTCP`) command, including the distributed data management (DDM) servers. If you are using data port, you must configure a cluster, make sure that the cluster is active, and start the internet Daemon (`inetd`) server using the Start TCP/IP Server (`STRTCPSVR`) command.End of change

7. If one does not already exist, create the appropriate relational database (RDB) directory entry that will be used to define the communications protocol for the remote journal environment. When TCP communications are being used to connect to an independent disk pool, the RDB entry to the independent disk pool must have the Relational database value set to the target system's local RDB entry and the relational database alias value set to the independent disk pool's name.
8. Now you should be able to see the remote database connection by issuing the `WRKRDBDIRE` command.

```
Work with Relational Database Directory Entries

Position to . . . . . .

Type options, press Enter.
  1=Add 2=Change 4=Remove 5=Display details 6=Print details

                            Remote
Option Entry Location Text

        SYS1 system1
        SYS2 system2
        MYSYSTEM *LOCAL Entry added by system








                                                                        Bottom
F3=Exit F5=Refresh F6=Print list F12=Cancel F22=Display entire field
(C) COPYRIGHT IBM CORP. 1980, 2007.
```

Adding a Remote Journal

Adding a remote journal creates a remote journal on a target system or independent disk pool and associates that remote journal with the journal on the source system. This occurs if this is the first time the remote journal is being established for a journal. The journal on the source system can be either a local or remote journal.

If a remote journal environment has previously been established, adding a remote journal reassociates the remote journal on the target system with the journal on the source system.

You can establish and associate a remote journal on a target system with a journal on the source system by one of the following methods:

- System i Navigator.
- Add the Remote Journal (`QjoAddRemoteJournal`) API on the source system.
- Add the Remote Journal (`ADDRMTJRN`) command on the source system.

What Happens During Add Remote Journal Processing?

The processing that takes place as part of adding a remote journal includes the following:

- A check is performed on the target system to verify that the user profile adding the remote journal exists. A user profile with the same name as the user profile which is adding a remote journal must exist on the target system. If the profile does not exist on the target system, then an exception is signaled, and the processing ends.
- A check is performed to verify that the target system has a library by the same name as the library for the journal on the source system. If the library does not exist on the target system, then an exception is signaled, and the processing ends.
- A check is performed on the target system to determine if a journal by the same qualified name as the journal on the source system already exists. If a journal already exists, it can be used for the remainder of the add remote journal processing if it meets the following criteria:
  1. It is a remote journal.
  2. It was previously associated with this same source journal or part of the same remote journal network.
  3. The type of the remote journal matches the specified remote journal type.
- If a journal was found, but does not meet the preceding criteria, then an exception is signaled, and the processing ends. Otherwise, the remote journal is used for the rest of the add remote journal processing.
- If no journal is found on the specified target system, then a remote journal is created on the target system. The new remote journal has the same configuration, authority, and audit characteristics of the source journal. The journal that is created has a journal type of `*REMOTE`.

When adding the remote journal, you must specify the type of remote journal to add. The remote journal type influences the library redirection rules and other operational characteristics for the journal.

Guidelines For Adding a Remote Journal

You should observe the following guidelines for adding a remote journal:

- You can only associate a remote journal with a single source journal.

  Note: The same remote journal can then have additional remote journals that are associated with it that are located on other target systems. This is the cascade configuration that is shown in Network configurations for remote journals.

- The remote journal will only have its attached receiver populated with journal entries that are replicated from the corresponding journal receiver on the source system. No journal entries can be directly deposited to a remote journal.
- A maximum of 255 remote journals can be associated with a single journal on a source system. This can be any combination of asynchronously maintained or synchronously maintained remote journals.

To Add a Remote Journal

The following is an example using the physical file QGPL/TESTPF setup to have remote journaling enabled to a second system.

1. Create the physical file.:

```
> CRTPF FILE(QGPL/TESTPF) RCDLEN(10)
  File TESTPF created in library QGPL.
  Member TESTPF added to file TESTPF in QGPL.
```

2. Create the local journal receiver and journals, and enable the journaling of the physical file created:

```
> crtjrnrcv jrnrcv(qgpl/jrcvrmt)
  Journal receiver JRCVRMT created in library QGPL

> crtjrn jrn(qgpl/jrnrmt) jrnrcv(qgpl/jrcvrmt) fixlendta(*job *usr *pgm *sysseq)
  Journal JRNRMT created in library QGPL

strjrnpf file(qgpl/testpf) jrn(qgpl/testpf)
  1 of 1 files have started journaling
```

3. Add the remote journal:

```
> addrmtjrn rdb(sys2) srcjrn(qgpl/JRNRMT) rmtjrntype(*TYPE2)
  Remote journal JRNRMT in QGPL was added
```

4. Activate the remote journaling:

```
> chgrmtjrn rbd(sys2) srcjrn(qgpl/jrnrmt) jrnstate(*active)
  Remote journal JRNRMT in library QGPL was activated
```

Specifying Object Names

Oracle GoldenGate commands and parameters support input in the form of SQL names, native names in the format of `library_name/file_name(member_name)`, or a mix of the two. If a native file system name does not include the member name, all members are implicitly selected by the Oracle GoldenGate process. For a SQL name only the first member is used.

To support case sensitivity of double quoted object names, specify those names within double quotes in the Oracle GoldenGate parameter files. This is true of both SQL and native file system names.

When specifying a native table name in a `MAP` statement on a platform other than DB2 for i, the name must be enclosed within double quotes so that Oracle GoldenGate correctly interprets it as a separator character.

For consistency of terminology in other administrative and reference Oracle GoldenGate documentation, the SQL terms "schema" and "table" are used to reference the containers for the DB2 for i data, as shown here.

Table 3-1 Native-SQL object name relationships

Native	SQL	Notes
Library (maximum length 10)	Schema (maximum length 128)	The operating system creates a corresponding native name for a SQL-created schema.
File (maximum length 10)	Table (maximum length 128)	The operating system creates a corresponding native name for a SQL-created table.
Member	Not Applicable	Contains the actual data. Only the first member of a `FILE` object can be accessed through SQL. To access data in other members the native system name must be used.

Adjusting the System Clock

It is recommended that you set the system clock to UTC (Universal Time Coordinate) time and use the timezone offset in the DB2 for i system values to represent the correct local time. If this setup is done correctly, local daylight savings time adjustments can occur automatically with no disruption to replication.

Configuring Database Connections

This section applies only if you installed Replicat on a Windows or Linux system to operate remotely from the DB2 for i target. In this configuration, Replicat must connect to the target system over a database connection that is specified with ODBC. The following steps show how to install and configure ODBC to connect to the DB2 for i target system.

Configuring ODBC on Linux

Configuring ODBC on Windows

Configuring ODBC on Linux

To configure ODBC, you can use the graphical user interface to run the ODBC configuration utility that is supplied with your Linux distribution, or you can edit the `odbc.ini` file with the settings described in these steps. (These steps show the ODBC Administrator tool launched from the `ODBCConfig` graphical interface utility for Linux.)

1. Download and install the 32-bit or 64-bit iSeries Access ODBC driver on the remote Linux system according to the vendor documentation. The iSeries ODBC driver is supplied as a free component of iSeries Access.
2. Issue one of the following commands, depending on the driver that you want to use.

   32-bit driver:

   ```
   rpm -ivh iSeriesAccess-7.1.0-1.0.i386.rpm
   ```

   64-bit driver:

   ```
   rpm -ivh iSeriesAccess-7.1.0-1.0.x86_64.rpm
   ```

3. You can create a user DSN (a connection that is available only to the user that created it) or a system DSN (a connection that is available to all users on the system). To create a user DSN, log on to the system as the user that you will be using for the Replicat process.
4. Run the ODBC configuration utility.
5. On the initial page of the ODBC configuration tool, select the **User DSN** tab to create a user DSN or the **System DSN** tab to create a system DSN. (These steps create a user DSN; creating a system DSN is similar.)
6. On the tab you selected, click **Add**.
7. Select the appropriate iSeries Access ODBC driver, click **OK**. If the correct driver is not shown in the **Select the DRIVER** list, click the **Add** button and then complete the fields.

   Steps to complete the Driver Properties fields when the driver is not found:

   - Set **Name** to the name of the driver.
   - Set **Driver** to the path where the driver is installed.
   - Set **Setup** to the `libcwbodbci.so` file that is in the driver installation directory.
   - Leave the other settings to their defaults.
   - Click the check mark above the **Name** field to save your settings.
8. In the **Name** field of the Data Source Properties dialog, supply a one-word name for the data source. In the **System** field, enter the fully qualified name of the target DB2 for i system, for example: `sysname.company.com`. Leave the **UserID** and **Password** fields blank, to allow Replicat to supply credentials when it connects to the database. Leave the remaining fields set to their defaults, and then click the check mark above the **Name** field to save your settings.
9. You are returned to the ODBC Data Source Administrator dialog. Click **OK** to exit the ODBC configuration utility.
10. To support `GRAPHIC`, `VARGRAPHIC` and `DBCLOB` types, edit the `.odbc.ini` file and add the following line.

    ```
    GRAPHIC = 1
    ```

    > **Note:**
    >
    > If you created a user Data Source Name, this file is located in the home directory of the user that created it. If you created a system DSN, this file is in `/etc/odbc.ini` or `/usr/local/etc/odbc.ini`.

11. From the Oracle GoldenGate directory on the target, run GGSCI and issue the `DBLOGIN` command to log into the target database.

    ```
    DBLOGIN SOURCEDB database, USERID db_user [, PASSWORD pw [encryption options]]
    ```

    Where:

    - `SOURCEDB` *`database`* specifies the new Data Source Name.
    - `USERID` *`db_user`*`,` `PASSWORD` *`pw`* are the Replicat database user profile and password.
    - *`encryption options`* is optional password encryption.

> **Note:**
>
> Only `BLOWFISH` encryption is supported for DB2 for i systems.

Configuring ODBC on Windows

On Windows, the ODBC Administration tool is in the Administrative Tools folder as **Data Sources (ODBC)**.

1. Download and install the 32-bit or 64-bit iSeries Access ODBC driver from the DB2 for iSeries Access package on the remote Windows system according to the vendor documentation. The iSeries ODBC driver is supplied as a free component of iSeries Access.
2. You can create a user DSN (a connection that is available only to the user that created it) or a system DSN (a connection that is available to all users on the system). To create a user DSN, log on to the system as the user that you will be using for the Replicat process.
3. From the Windows Control Panel, select **Administrative Tools**, then **Data Sources (ODBC)**.
4. On the first page of the ODBC configuration tool, select the **User DSN** tab to create a user DSN or the **System DSN** tab to create a system DSN. (These steps create a user DSN; creating a system DSN is similar.)
5. On the tab that you selected, click **Add**.
6. Select the appropriate iSeries Access ODBC Driver from the list of drivers, and then click **Finish**.
7. On the General tab of the DB2 for i Access for Windows ODBC Setup dialog, provide a name (without any spaces) in the **Data Source Name** field, add an optional description in the **Description** field, and then select the system name from the **System** selection list.
8. On the Server tab, set **Naming Convention** to `SQL Naming Convention (*SQL)`. Leave the other fields set to their defaults.
9. On the Data Types tab, select the **Report as Supported** check box under Double Byte Character Set (DBCS) graphic data types.
10. On the Conversions tab, clear the **Convert binary data (CCSID 65535) to text** check box.
11. Click **Apply**, then **OK**. You are returned to the ODBC Data Source Administrator dialog.
12. Confirm that the new Data Source Name appears under **User Data Sources**.
13. Click **OK** to exit the ODBC configuration utility.
14. From the Oracle GoldenGate directory on the target, run GGSCI and issue the `DBLOGIN` command to log into the target database. See *Parameters and Functions Reference for Oracle GoldenGate* for detailed syntax.

    ```
    DBLOGIN SOURCEDB database, USERID db_user [, PASSWORD pw [encryption_options]]
    ```

    Where:

    - `SOURCEDB` *`database`* specifies the new data source name.
    - `USERID` *`db_user`*`,` `PASSWORD` *`pw`* are the Replicat database user profile and password.
    - *`encryption_options`* is optional password encryption.

Note:

Only `BLOWFISH` encryption is supported for DB2 for i systems.

Configuring Oracle GoldenGate for DB2 for i

This chapter contains instructions for configuring Oracle GoldenGate to capture source DB2 for i data and apply it to a supported target database.

Creating a GLOBALS File

The GLOBALS parameter file contains parameters that affect all processes within an Oracle GoldenGate instance.

GGSCHEMA is a mandatory parameter for Oracle GoldenGate 21c (21.3.0) onwards and defines the schema, which Oracle GoldenGate uses on the remote system for necessary Oracle GoldenGate database objects.

The GLOBALS parameter `NAMECCSID` is specific to DB2 for i and may be required, if the SQL catalog contains object names that are referenced by a different CCSID than the system CCSID. The SQL catalog is created in the system CCSID and does not indicate this difference when queried. Oracle GoldenGate makes queries to the catalog and could retrieve the name incorrectly unless `NAMECCSID` is used to supply the correct CCSID value. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Preparing Tables for Processing

The following table attributes must be addressed in an Oracle GoldenGate environment.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Using `KEYCOLS` to Specify a Custom Key

If a table does not have one of the preceding types of row identifiers, or if you prefer those identifiers not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds.

Preventing Key Changes

If you must add columns to the key that Extract is using as the row identifier for a table (primary key, unique key, `KEYCOLS` key, or all-column key) after Oracle GoldenGate has started processing journal data, follow these steps to make the change.

1. Stop Extract.

```
STOP EXTRACT group
```

2. Issue the following command until it returns `EOF`, indicating that it has processed all of the existing journal data.

   ```
   INFO EXTRACT group
   ```

3. Make the change to the key.
4. Start Extract.

   ```
   START EXTRACT group
   ```

Disabling Constraints on the Target

Triggers and cascade constraints must be disabled on the target tables or configured to ignore changes made by Replicat. Constraints must be disabled because Oracle GoldenGate replicates DML that results from a trigger or a cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

Enabling Change Capture

To capture changes to a table in a journal, you can run the `STRJRNPF` command on the OS/400 command line or run the `ADD TRANDATA` command from GGSCI. The `ADD TRANDATA` command calls `STRJRNPF` and is the recommended method to start journaling for tables, because it ensures that the required journal image attribute of Record Images `(IMAGES): *BOTH` is set on the `STRJRNPF` command.

To Run `ADD TRANDATA`

1. Run GGSCI on the source system.
2. Issue the `DBLOGIN` command.

   ```
   DBLOGIN SOURCEDB database USERID user, PASSWORD password
   [encryption_options]
   ```

   Where: `SOURCEDB` specifies the default DB 2 for i database, `USERID` specifies the Extract user profile, and `PASSWORD` specifies that profile's password.

3. Issue the `ADD TRANDATA` command.

   ```
   ADD TRANDATA table_specification
   ```

   Where: *`table_specification`* is one of the following:

- `*schema.table* [JOURNAL *library/journal*]`
- `*library/file* [JOURNAL  *library/journal]*`

Specifying a Default Journal

To specify a default journal for multiple tables or files in the `ADD TRANDATA` command, instead of specifying the `JOURNAL` keyword, use the following GGSCI command before issuing `ADD TRANDATA`.

```
DEFAULTJOURNAL library/journal
```

Any `ADD TRANDATA` command used without a journal assumes the journal from `DEFAULTJOURNAL`.

To display the current setting of `DEFAULTJOURNAL`, you can issue the command with no arguments.

Removing a Default Journal Specification

To remove the use of a default journal, use the following GGSCI command:

```
DEFAULTJOURNAL CLEAR
```

Maintaining Materialized Query Tables

To maintain parity between source and target materialized query tables (MQT), you replicate the base tables, but not the MQTs. The target database maintains the MQTs based on the changes that Replicat applies to the base tables.

The following are the rules for configuring these tables:

- Include the base tables in your `TABLE` and `MAP` statements.
- Do not include MQTs in the `TABLE` and `MAP` statements.
- Wildcards can be used in `TABLE` and `MAP` statements, even though they might resolve MQT names along with regular table names. Oracle GoldenGate automatically excludes MQTs from wildcarded table lists. However, any MQT that is explicitly listed in an Extract `TABLE` statement by name will cause Extract to abend.

Specifying the Oracle GoldenGate Library

Before starting Oracle GoldenGate, specify the name of the Oracle GoldenGate for DB2 for i library when running the `ggos400install` script. This creates a link to the `OGGPRCJRN *SRVPGM` (service program) object that was restored to that library. If the link to the `oggprcjrn` service program is deleted you could just re-run the `ggos400install` shell script and specify the same library, or use the command "`ln -s /qsys.lib/`*`OGG_library`*`.lib/oggprcjrn.srvpgm oggprcjrn.srvpgm`". If this link is incorrect or missing, Extract will abend.

Understanding What's Supported for DB2 for i

This chapter contains information on database and table features supported by Oracle GoldenGate for DB2 for i.

Oracle GoldenGate on DB2 for i supports the filtering, mapping, and transformation of data unless otherwise noted in this documentation.

Oracle GoldenGate for DB2 for i runs remotely from a Linux system on a DB2 for i source system to capture data from the transaction journals for replication to a target system. To apply data to a target DB2 for i database, Oracle GoldenGate can run remotely from a Linux system. Oracle GoldenGate communicates with the IBM i system by means of an ODBC connection, and no Oracle GoldenGate software is installed on the DB2 for i target.

> **Note:**
>
> The DB2 for i platform uses one or more **journals** to keep a record of transaction change data. For consistency of terminology in the supporting administrative and reference Oracle GoldenGate documentation, the terms "log" or "transaction log" may be used interchangeably with the term "journal" where the use of the term "journal" is not explicitly required.

Supported DB2 for i Data Types

Oracle GoldenGate supports all DB2 for i data types, except those listed in Non-Supported DB2 for i Data Types.

Limitations of support

Extract fully supports the capture and apply of `TIMESTAMP(0)` through `TIMESTAMP(12)` when the output trail format is 19.1 or higher. Otherwise Extract truncates data from `TIMESTAMP(10)` through `TIMESTAMP(12)` to nanoseconds (maximum of nine digits of fractional time) and issues a warning to the report file.

Oracle GoldenGate supports timestamp data from 0001/01/03:00:00:00.000000 to 9999/12/31:23:59:59.999999. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the time zone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.

Non-Supported DB2 for i Data Types

Oracle GoldenGate does not support the following DB2 for i data types:

- `XML`
- `DATALINK`
- User-defined types

Supported Objects and Operations for DB2 for i

Oracle GoldenGate supports the following DB2 for i objects and operations.

- Extraction and replication of DML operations .
- Tables with the maximum row length supported by the database.
- Tables that contain up to the maximum number of columns that is supported by the database, up to the maximum supported column size.
- `TRUNCATE` operations are supported and are represented by `DELETE FROM` with no `WHERE` clause SQL statements and `Clear Physical File Member` (`CLRPFM`).

- Base tables underlying Materialized Query Tables, but not the MQTs themselves. The target database automatically maintains the content of the MQT based on the changes that are applied to the base tables by Replicat.
- Both Library (Native) names including members, and SQL names are allowed.
- Partitioned tables
- Supported options with `SHOWTRANS`:

```
SHOWTRANS [transaction_ID]  [COUNT n]
[DURATION duration unit] [TABULAR]
[FILE file_name] |
```

- Options for `SKIPTRANS` and `FORCETRANS`:

```
SKIPTRANS transaction_ID  [FORCE]
FORCETRANS transaction_ID  [FORCE]
```

- Limitations on Automatic Heartbeat Table support are as follows:
  - The `ADD HEARTBEATTABLE` command creates a new file called `ogghbfreq` in the Oracle GoldenGate installation directory. Do not delete this file because the `pase` heartbeat program reads the frequency values from it.
  - There is an extra executable in the Oracle GoldenGate build folder named `ogghb`.
  - An extra process named `ogghb` starts running on the IBM i system when the `ADD HEARTBEATTABLE` command runs until you disable the heartbeat with the `DELETE HEARTBEATTABLE` command. This process automatically restarts even if it is killed. To remove this process from the system, use the `DELETE HEARTBEATTABLE` command.
  - When using the `ALTER HEARTBEATTABLE` command to change the heartbeat frequency with the `PURGE_FREQUENCY` or `RETENTION_TIME` options, it takes approximately 60 + older 'frequency') seconds to be implemented.
  - There is an initial delay of 30 seconds between `ADD HEARTBEATTABLE` and the first record is updated in the heartbeat seed table.
  - `[THREAD n]` and `[DETAIL]` is not supported.

Non-Supported Objects and Operations for DB2 for i

Oracle GoldenGate does not support the following objects or operations for DB2 for i.

- DDL operations
- Schema, table or column names that have trailing spaces
- Multiple instances of a database

Oracle GoldenGate Parameters Not Supported for DB2 for i

This section lists some of the Oracle GoldenGate configuration parameters that are not supported for the DB2 for i platform. For full descriptions of Oracle GoldenGate parameters and the databases they support, see Oracle GoldenGate Parameters.

`BATCHSQL`

`BR`

`ASCIITOEBCDIC` and `EBCDICTOASCII`

`BINARYCHARS`

```
LOBMEMORY
TRAILCHARSETEBCDIC
```

Supported Object Naming Conventions

Oracle GoldenGate supports SQL naming conventions and also supports native file system names in the format of `library/file(member)`.

For native (system) names, Oracle GoldenGate supports the normal DB2 for i naming rules for wildcarding, which allows `*ALL` or a partial name with a trailing asterisk (*) wildcard. For example:

- `library/*all(*all)`
- `library/a*(a*)`
- `library/abcde*`

The member name is optional and may be left off. In that case, data for all of the members will be extracted, but only the library and file names will be captured and included in the records that are written to the trail. The result is that the data will appear to have come from only one member on the source, and you should be aware that this could cause integrity conflicts on the target if there are duplicate keys across members. To include the member name in the trail records, include the member explicitly or though a wildcarded member specification.

For SQL names, only the first member in the underlying native file is extracted in accordance with the normal operation of SQL on an DB2 for i system. For SQL names, Oracle GoldenGate supports the wildcarding of both table names and schema names. For instructions on wildcarding SQL names, see Specifying Object Names in Oracle GoldenGate Input in *Administering Oracle GoldenGate*.

System Schemas for DB2 for i

The following schemas or objects are not automatically replicated by Oracle GoldenGate unless they are explicitly specified without a wildcard..

- `"Q*"`
- `"SYSIBM"`
- `"SYSIBMADM"`
- `"SYSPROC"`
- `"SYSTOOLS"`
- `"#LIBRARY"`
- `"#RPGLIB"`

Supported Character Sets

The default behavior of a DB2 for i Extract is to convert all character data to Unicode. The overhead of the performance of the conversion to UTF-8 for the text data has been substantially reduced. However, if you want to send data in its native character set you can use the parameter `DBOPTIONS USEDATABASEENCODING` to override the default behavior.

Db2 z/OS

With Oracle GoldenGate for Db2 z/OS, you can perform initial loads and capture transactional data from supported Db2 z/OS versions and replicate the data to a Db2 z/OS database or other supported Oracle GoldenGate targets, such as an Oracle Database.

Oracle GoldenGate for Db2 z/OS is installed and runs remotely on Linux, zLinux, or AIX.

Oracle GoldenGate for DB2 z/OS supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

System Services

Activate UNIX System Services (USS) only if required to install the executables for the Extract support modules.

Oracle GoldenGate supports Sysplex data sharing.

Database User for Oracle GoldenGate Processes

Oracle GoldenGate requires a database user account. Create this account and assign privileges according to the following guidelines.

Assign the DB2 privileges listed in the following table to the user. These are in addition to any permissions that DB2 ODBC requires. All Extract privileges apply to initial-load and log-based Extract processes, except where noted.

The following authorities can be provided by granting either `SYSCTRL` or `DBADM` plus `SQLADM` authority to the user running the Oracle GoldenGate processes.

Table 3-2 Privileges Needed by Oracle GoldenGate for Db2 z/OS

User privilege	Extract	Replicat
`MONITOR2` (does not apply to initial-load Extract)	X	

`SELECT ON` the following `SYSIBM` tables:
`SYSTABLES`
`SYSCOLUMNS`
`SYSTABLEPART`
`SYSKEYS`
`SYSINDEXES`
`SYSCOLAUTH`
`SYSDATABASE`
`SYSFOREIGNKEYS`
`SYSPARMS`
`SYSRELS`
`SYSROUTINES`
`SYSSYNONYMS`
`SYSTABAUTH`
`SYSAUXRELS`
	X	X

Table 3-2 (Cont.) Privileges Needed by Oracle GoldenGate for Db2 z/OS

User privilege	Extract	Replicat
`SELECT` on source tables[1]	*X*	
`INSERT`, `UPDATE`, `DELETE` on target tables		X
`CREATE TABLE`[2]		X
`EXECUTE` on ODBC plan (default is DSNACLI)	X	
Privileges required by `SQLEXEC` procedures or queries that you will be using.[3]	X	X

[1] `SELECT` on source tables required only if tables contain LOB columns, or for an initial-load Extract, if used.

[2] Required if using `ADD CHECKPOINTTABLE` in GGSCI to use the database checkpoint feature.

[3] `SQLEXEC` enables stored procedures and queries to be executed by an Oracle GoldenGate process.

Ensuring ODBC Connection Compatibility

To ensure that you configure the DB2 ODBC initialization file correctly, follow the guidelines in the *DB2 UDB for z/OS ODBC Guide and Reference* manual. One important consideration is the coding of the open and close square brackets (the [character and the] character). The square bracket characters are "variant" characters that are encoded differently in different coded character set identifiers (CCSID), but must be of the IBM-1047 CCSID in the ODBC initialization file. DB2 ODBC does not recognize brackets of any other CCSID. Note the following:

- The first (or open) bracket must use the hexadecimal characters `X'AD'` (`0xAD`).
- The second (or close) bracket must use the hexadecimal characters `X'BD'` (`0xBD`).

To set the correct code for square brackets, use any of the following methods.

- Use the `hex` command in OEDIT and change the hex code for each character appropriately.
- Use the `iconv` utility to convert the ODBC initialization file. For example, to convert from CCSID IBM-037 to IBM-1047, use the following command:

  ```
  iconv -f IBM-037 -t IBM-1047 ODBC.ini > ODBC-1047.ini

  mv ODBC-1047.ini ODBC.ini
  ```
- Change your terminal emulator or terminal configuration to use CCSID IBM-1047 when you create or alter the file.

Configure a Database Connection

This section contains instructions for setting up the Extract and Replicat connections to a DB2 z/OS database.

Database Configuration for DB2 z/OS

No special DB2 z/OS database settings are required for Oracle GoldenGate.

Setting Initialization Parameters

The following DB2 for z/OS initialization parameters apply to Oracle GoldenGate and must be set correctly before starting Oracle GoldenGate processes.

- `MVSDEFAULTSSID`: set to the DB2 subsystem.
- `LOCATION`: set to the DB2 location name as stored in the DB2 Boot Strap Dataset.
- `MVSATTACHTYPE`: set to `RRSAF` (Recoverable Resource Manager Services Attachment Facility) or `CAF` (Call Attachment Facility). IBM recommends using `RRSAF`.
- `MULTICONTEXT:` set to 1 if using `RRSAF`.
- `PLANNAME`: set to the DB2 plan. The default plan name is `DSNACLI`.

Do not use the `CURRENTAPPENSCH` initialization parameter (keyword).

> **Note:**
>
> When using the `CAF` attachment type, you must use the Oracle GoldenGate `DBOPTIONS` parameter with the `NOCATALOGCONNECT` option in the parameter file of any Extract or Replicat process that connects to DB2. This parameter disables the usual attempt by Oracle GoldenGate to obtain a second thread for the DB2 catalog. Otherwise, you will receive error messages, such as: `ODBC operation failed: Couldn't connect to` *`data source`* `for catalog queries.`

Specifying the Path to the Initialization File

Specify the ODBC initialization file by setting the `DSNAOINI` environment variable in the z/OS UNIX profile, as in the following example:

```
export DSNAOINI="/etc/odbc810.ini"
```

Specifying the Number of Connection Threads

Every Oracle GoldenGate process makes a database connection. Depending on the number of processes that you will be using and the number of other DB2 connections that you expect, you might need to adjust the following DB2 system parameters on the DSNTIPE DB2 Thread Management Panel:

- `MAX USERS` (macro `DSN6SYSP CTHREAD`)
- `MAX TSO CONNECT` (macro `DSN6SYSP IDFORE`)
- `MAX BATCH CONNECT` (macro `DSN6SYSP IDBACK`)

If using RRSAF, allow:

- Two DB2 threads per process for each of the following:
  - Extract
  - Replicat
  - The GGSCI command `DBLOGIN` (logs into the database)
  - `DEFGEN` utility (generates data definitions for column mapping)

- One extra DB2 thread for Extract for IFI calls.
- One extra DB2 thread for each `SQLEXEC` parameter statement that will be issued by each Extract and Replicat process. For more information about `SQLEXEC`, see the *Parameters and Functions Reference for Oracle GoldenGate*.

If using CAF, there can be only one thread per Oracle GoldenGate process.

Database Configuration

Learn how to configure the DB2 z/OS environment to support Oracle GoldenGate Classic Architecture.

Monitoring Processes

These sections provide information about monitoring Oracle GoldenGate with z/OS system facilities.

Interpreting Statistics for Update Operations

The actual number of DML operations that are executed on the DB2 database might not match the number of extracted DML operations that are reported by Oracle GoldenGate. DB2 does not log update statements if they do not physically change a row, so Oracle GoldenGate cannot detect them or include them in statistics.

Supporting Globalization Functions

Oracle GoldenGate provides globalization support and you should take into consideration when using this support.

Replicating From a Source that Contains Both ASCII and EBCDIC

When replicating to or from a DB2 source system to a target that has a different character set, some consideration must be given to the encoding of the character data on the DB2 source if it contains a mix of ASCII and EBCDIC data. Character set conversion by any given Replicat requires source data to be in a single character set.

The source character set is specified in the trail header. Thus, the Oracle GoldenGate trail can contain either ASCII or EBCDIC data, but not both. Unicode tables are processed without any special configuration and are exempt from the one-character set requirement.

With respect to a source that contains both character encoding types, you have the following options:

- You can use one Extract for all of your tables, and have it write the character data to the trail as either ASCII or as EBCDIC.
- You can use different Extracts: one Extract to write the ASCII character data to a trail, and another Extract to write the EBCDIC character data to a different trail. You then associate each trail with its own data pump process and Replicat process, so that the two data streams are processed separately.

To output the correct character set in either of those scenarios, use the `TRAILCHARSETASCII` and `TRAILCHARSETEBCDIC` parameters. The default is `TRAILCHARSETEBCDIC`. Without these parameters, ASCII and EBCDIC data are written to the trail as-is. When using these parameters, note the following:

- If used on a single-byte DB2 subsystem, these parameters cause Extract to convert all of the character data to either the ASCII or EBCDIC single-byte CCSID of the subsystem to which Extract is connected, depending on which parameter is used (except for Unicode, which is processed as-is).

- If used on a multi-byte DB2 subsystem, these parameters cause Extract to capture only ASCII or EBCDIC tables (and Unicode). Character data is written in either the ASCII or EBCDIC mixed CCSID (depending on the parameter used) of the DB2 z/OS subsystem to which Extract is connected.

Specifying Multi-Byte Characters in Object Names

If the name of a schema, table, column, or stored procedure in a parameter file contains a multi-byte character, the name must be double-quoted. For more information about specifying object names, see .

Installing Extract Components on Db2 z/OS

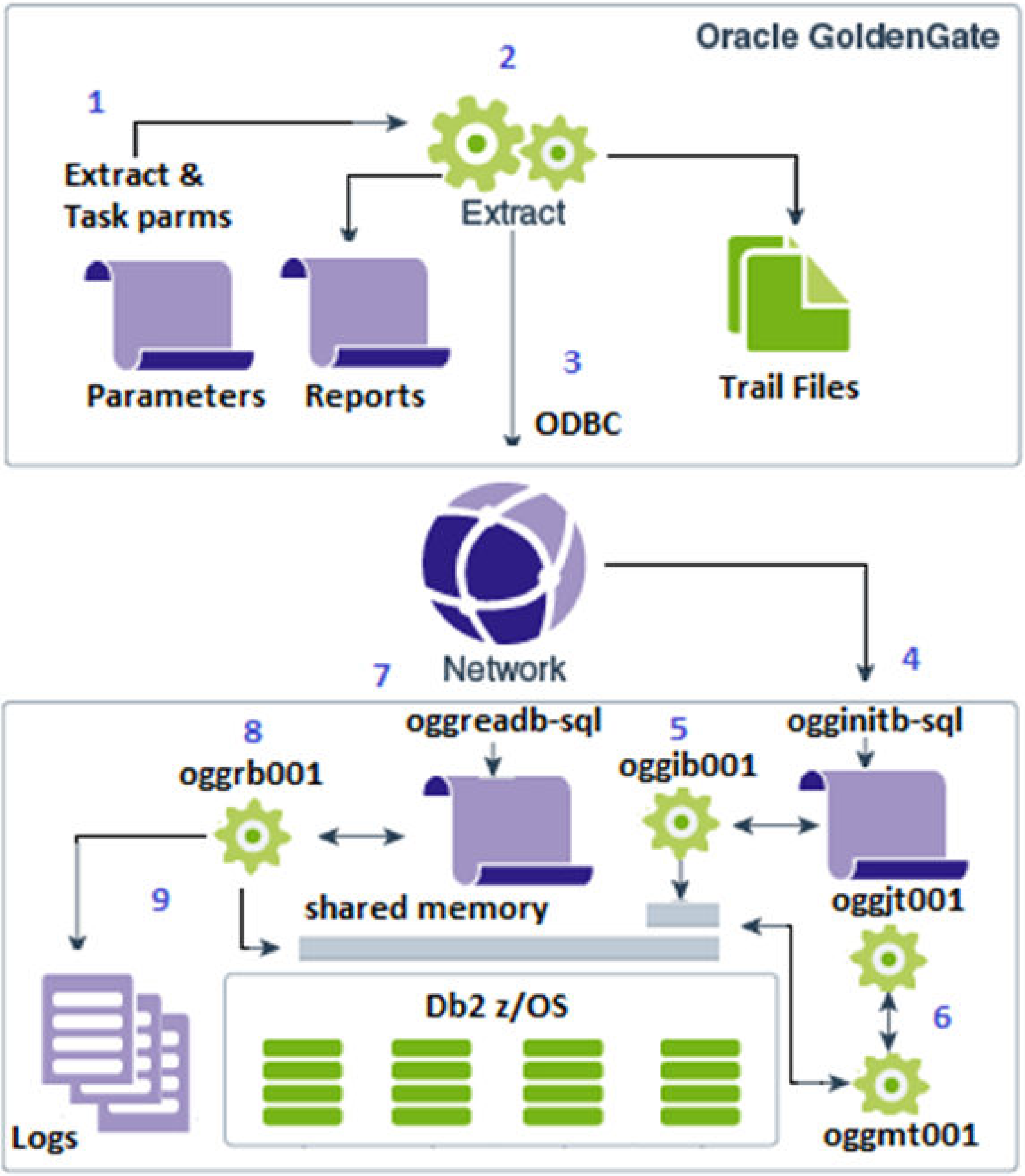
The Oracle GoldenGate Db2 z/OS Extract uses SQL objects to access and read the Db2 log. These Oracle GoldenGate Db2 z/OS objects require a minimum hardware platform of z10, a minimum operating system release of 1.13, and a minimum Db2 release of 11. The components consist of executable load modules, SQL stored procedures and functions, and external programs called via the stored procedures. these components are:

1. External programs (authorized) includes the following programs:
    a. `oggib001` – Initialization and utility program
    b. `oggrb001` – Log read program functionality
    c. `oggmt001` – Stand-alone program that monitors ECSA and 64-bit memory
    d. `oggjt001` – Setup program for the `oggmt001` startup JCL run from `oggib001` program
    e. `oggfr001` – Utility for use by a DBA under guidance from Oracle Support
2. SQL stored procedure and function includes `demo_db2_setupb_os390.sql` with the `OGGINITB` and `OGGREADB SQL`.
3. JCL procedure, `oggtask.jcl`

> **Note:**
>
> These external names, SQL and JCL names are the default names, which you can edit and update. This process is discussed in the subsequent sections.

The Replication Process for Db2 z/OS Extract figure illustrates the replication process for the Db2 z/OS Extract and its mainframe components.

Figure 3-1 Replication Process for Db2 z/OS Extract



The process starts and runs as shown using the numbers 1 through 9 in the figure, which is given below:

1. Extract reads the parameters, including the JCL parameters, from the parameter file created during installation.
2. Extract reports the startup information and prepares to write the trail files.
3. ODBC is used to gather information from the Db2 database and start replication.
4. The `OGGINITB` SQL stored procedure starts to prepare shared memory and to gather other data needed for replication.

5. The `OGGIB001` external program called by the SQL stored procedure starts the memory monitor task using the `OGGJT001` job setup program.
6. The `OGGMT001` memory monitor task starts monitoring the ECSA and 64-bit shared memory.
7. The `OGGREADB` SQL Function calls the external program `OGGRB001`.
8. The `OGGRB001` external program repeatedly calls the Db2 log read program to create a result set that returns 1 to many log record buffers to the Extract.
9. When a log record result set is complete, `OGGRB001` ends after sending the result set to the Extract.

Extract repeats steps 7 to 9 until shut down or abnormal termination. If the memory task fails to start, `OGGI001` program returns a flag indicating there was a JCL error or setup issue and Extract manages its own memory. If the memory task starts properly, the memory task tests constantly changing fields in the 48-byte ECSA shared memory. These fields stop changing if the Extract terminates for any reason. At that point, the memory manager waits in case the Extract or network is slow and releases the memory before shutting down after a configured time limit.

To install the components needed for Oracle GoldenGate for Db2 z/OS for Extract:

1. Ensure that a library (PDSE) exists on the Db2 z/OS system and an entry for it is made in the authorized library list. This library is the location where the Oracle GoldenGate external program objects will reside.
2. Ensure that an APF-authorized WLM environment exists that references the PDSE from the preceding step. Oracle recommends that NUMTCB value for the WLM environment be 10-40 for stored procedures. The NUMTCB value depends on the maximum number of Extracts that are running concurrently against the database and on how much throughput each Extract requires. If you want flexibility in setting NUMTCB, you specify it in the startup JCL for the WLM, but not in the creation panel.
3. You can set up security for the WLM application environments and for creating stored procedures by completing the following:
    a. (Optional) Specify which WLM-established address spaces can run stored procedures. If you do not complete this step, then any WLM-established address space can run stored procedures.
    b. Grant access to users to create procedures in specific WLM address spaces.
    c. Grant access to users to create procedures in specific schemas. Use the `GRANT` statement with the `CREATIN` option for the appropriate schema.
    d. Grant access to users to create packages for procedures in specific collections. Use the `GRANT` statement with the `CREATE` option for the appropriate collection.
    e. Grant access to refresh the WLM environments to the appropriate people.
    f. Add additional RACF authority to the appropriate people, allowing the WLM procedures to start the memory manager job.
4. Ensure the ID used to run the WLM startup JCL procedure has permission to use RRSAF. Each time one of the Db2 WLM address spaces is started, it uses RRSAF to attach to Db2. See the Db2 11 for z/OS Installation and Migration Guide
5. In the Linux or UNIX installation of Oracle GoldenGate for Db2 z/OS, there is a ZIP file called `zOSPrograms.zip`. Unzip `zOSPrograms.zip` to `zOSPrograms.tar` and copy `zOSPrograms.tar` in binary mode to your Db2 z/OS system into an HFS directory.
6. On your Db2 z/OS system in USS or OMVS, change directories to the directory containing `zOSPrograms.tar`.

7. Restore the objects with the command: `tar -xovf zOSPrograms.tar`.

> **Note:**
>
> In this command, the copy target is double-quote forward-slash single-quote authorized PDSE name single-quote double quote. The `-X` is an uppercase capital X *not* a lowercase x.

8. Copy the objects to the authorized PDSE. Use the `cp -X ogg[irmj][abt][0-9]* “//’authorized_PDSE_name’”` where `authorized_PDSE_name` is the name of the APF authorized PDSE, which is intended for the Oracle GoldenGate objects. Using this command installs the objects with the default names.
9. Installing the scripts with different names allows you to conform with system protocols, or it allows you to run multiple versions of Oracle GoldenGate. To install the scripts with different names, it is recommended to create a shell script that renames the programs before copying them to the PDSE. An example of the shell script is given in the following code snippet.

   ```
   #!/bin/bash
   # Copy new programs renaming them to version 21.12 names.
   cp oggib001 oggi2112
   cp oggrb001 oggr2112
   cp oggmt001 oggm2112
   cp oggjt001 oggj2112
   cp -X oggi2112 “//’SYS4.WLMDSNA.AUTHLOAD’”
   cp -X oggr2112 “//’SYS4.WLMDSNA.AUTHLOAD’”
   cp -X oggm2112 “//’SYS4.WLMDSNA.AUTHLOAD’”
   cp -X oggj2112 “//’SYS4.WLMDSNA.AUTHLOAD’”
   ```

   You can run the script using `chmod +x` command. You can copy and reuse this script for new versions.
10. You must create the SQL procedures using your SQL tool of choice so that Oracle GoldenGate can call the Extract objects. The Oracle GoldenGate stored procedures should have permission granted to only those users that use them for replication.

    An example SQL script in the Oracle GoldenGate install directory contains the SQL statements to set up the stored procedures on the Db2 z/OS instance. The `demo_db2_setupb_os390.sql` script is for Db2 v11.1 and higher and can run from any SQL tool on any platform that can connect to your Db2 z/OS instance. This script must run on the Db2 instance that you use with your Extract. The script provided in the remote installation directory is in ASCII format. The same script is restored through `zOSPrograms.tar` on the Db2 z/OS system in EBCDIC and is suitable for use through native Db2 z/OS tools such as `SPUFI`.

    Edit the following line before running the scripts:

    - Modify the `WLM ENVIRONMENT` line to use the correct name for the WLM environment that you will use for Oracle GoldenGate.

Note:

The `oggifi0001` schema name is configurable using the `TRANLOGOPTIONS REMOTESCHEMA` *`schemaname`* parameter. The procedure names are not configurable. Each of the external names in the script and the PDSE can be renamed as long as the script names and the PDSE object names match. Changing these names is part of the procedure that allows migration to new versions or if specific naming procedures must be adhered to on Db2 z/OS. The following table contains a check list of components that you may wish to edit and/or update:

Table 3-3 List of Editable Components

Component	From	Rename	Where
`oggib001`	tar file		authorized PDSE
`oggrb001`	tar file		authorized PDSE
`oggmt001`	tar file		authorized PDSE & proc library
`oggjt001`	tar file		authorized PDSE & Extract parm
`oggpr001`	tar file		procedure library & Extract parm
proclib	MVS		add Extract parm if needed
step libraries	MVS		WLM and `oggpr001` procedure library
remoteschema			`demo_db2_setupb_os390.sql` and Extract parm
WLM name	MVS		`demo_db2_setupb_os390.sql`
external program			`demo_db2_setupb_os390.sql`

Note:

Remember to perform all these steps after every new patch installation.

Preparing Tables for Processing

You must perform the following tasks to prepare your tables for use in an Oracle GoldenGate environment.

Disabling Triggers and Cascade Constraints

Disable triggers, cascade delete constraints, and cascade update constraints on the target tables, or alter them to ignore changes made by the Oracle GoldenGate database user. Oracle GoldenGate replicates DML that results from a trigger or cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

- A delete is issued for `emp_src`.
- It cascades a delete to `salary_src`.
- Oracle GoldenGate sends both deletes to the target.
- The parent delete arrives first and is applied to `emp_targ`.
- The parent delete cascades a delete to `salary_targ`.
- The cascaded delete from `salary_src` is applied to `salary_targ`.
- The row cannot be located because it was already deleted in step 5.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key

2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Using `KEYCOLS` to Specify a Custom Key

If a table does not have one of the preceding types of row identifiers, or if you prefer those identifiers not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Handling `ROWID` Columns

Any attempt to insert into a target table that includes a column with a data type of `ROWID GENERATED ALWAYS` (the default) will fail with the following ODBC error:

```
ODBC error: SQLSTATE 428C9 native database error -798. {DB2 FOR OS/390}{ODBC
DRIVER}{DSN08015} DSNT408I SQLCODE = -798, ERROR: YOU CANNOT INSERT A VALUE
INTO A COLUMN THAT IS DEFINED WITH THE OPTION GENERATED ALWAYS. COLUMN NAME
ROWIDCOL.
```

You can do one of the following to prepare tables with `ROWID` columns to be processed by Oracle GoldenGate:

- Ensure that any `ROWID` columns in target tables are defined as `GENERATED BY DEFAULT`.
- If it is not possible to change the table definition, you can work around it with the following procedure.

To Work Around `ROWID GENERATE ALWAYS`:

1. For the source table, create an Extract `TABLE` statement, and use a `COLSEXCEPT` clause in that statement that excludes the `ROWID` column. For example:

   ```
   TABLE tab1, COLSEXCEPT (rowidcol);
   ```

   The `COLSEXCEPT` clause excludes the `ROWID` column from being captured and replicated to the target table.

2. For the target table, ensure that Replicat does not attempt to use the `ROWID` column as the key. This can be done in one of the following ways:
    - Specify a primary key in the target table definition.
    - If a key cannot be created, create a Replicat `MAP` parameter for the table, and use a `KEYCOLS` clause in that statement that contains any unique columns except for the `ROWID` column. Replicat will use those columns as a key. For example:

      ```
      MAP tab1, TARGET tab1, KEYCOLS (num, ckey);
      ```

Preparing the DB2 for z/OS Transaction Logs for Oracle GoldenGate

Learn how to configure the DB2 transaction logging to support data capture by Oracle GoldenGate Extract.

Preparing the DB2 z/OS Transaction Logs for Oracle GoldenGate

Learn to configure the DB2 transaction logging to support data capture by Oracle GoldenGate Extract.

Oracle GoldenGate can extract DB2 transaction data from the active and archived logs. Follow these guidelines to configure the logs so that Extract can capture data.

Enabling Change Capture

Follow these steps to configure DB2 to log data changes in the expanded format that is supplied by the `DATA CAPTURE CHANGES` feature of the `CREATE TABLE` and `ALTER TABLE` commands. This format provides Oracle GoldenGate with the entire before and after images of rows that are changed with update statements.

1. From the Oracle GoldenGate directory, run GGSCI.
2. Log on to DB2 from GGSCI as a user that has `ALTER TABLE` privileges.

   ```
   DBLOGIN SOURCEDB DSN, USERID user[, PASSWORD password][, encryption_options]
   ```

3. Issue the following command. where *`table`* is the fully qualified name of the table. You can use a wildcard to specify multiple table names but not owner names.

   ```
   ADD TRANDATA table
   ```

   By default, `ADD TRANDATA` issues the following command:

   ```
   ALTER TABLE name DATA CAPTURE CHANGES;
   ```

Enabling Access to Log Records

Activate DB2 Monitor Trace Class 1 ("`TRACE(MONITOR) CLASS(1)` ") so that DB2 allows Extract to read the active log. The default destination of `OPX` is sufficient, because Oracle GoldenGate does not use a destination.

To Start the Trace Manually

1. Log on to DB2 as a DB2 user who has the `TRACE` privilege or at least `SYSOPR` authority.
2. Issue the following command:

   ```
   start trace(monitor) class(1) scope(group)
   ```

To Start the Trace Automatically When DB2 is Started

Do either of the following:

- Set `MONITOR TRACE` to "`YES`" on the `DSNTIPN` installation tracing panel.
- Set '`DSN6SYSP MON=YES` ' in the `DSNTIJUZ` installation job, as described in the *DB2 UDB Installation Guide*.

> **Note:**
>
> The primary authorization ID, or one of the secondary authorization IDs, of the ODBC plan executor also must have the `MONITOR2` privilege.

Sizing and Retaining the Logs

When tables are defined with `DATA CAPTURE CHANGES`, more data is logged than when they are defined with `DATA CAPTURE NONE` . If any of the following is true, you might need to increase the number and size of the active and archived logs.

- Your applications generate large amounts of DB2 data.
- Your applications have infrequent commits.
- You expect to stop Extract for long periods of time.
- Your network is unreliable or slow.

To control log retention, use the `DSN6LOGP MAXARCH` system parameter in the `DSNTIJUZ` installation job.

Retain enough log data so that Extract can start again from its checkpoints after you stop it or after an unplanned outage. Extract must have access to the log that contains the start of the oldest uncommitted unit of work, and all logs thereafter.

If data that Extract needs during processing was not retained, either in online or archived logs, one of the following corrective actions might be required:

- Alter Extract to capture from a later point in time for which log data is available (and accept possible data loss on the target).
- Resynchronize the source and target tables, and then start the Oracle GoldenGate environment over again.

> **Note:**
>
> The IBM documentation makes recommendations for improving the performance of log reads. In particular, you can use large log output buffers, large active logs, and make archives to disk.

Using Archive Logs on Tape

Oracle GoldenGate can read DB2 archive logs on tape, but it will degrade performance. For example, DB2 reserves taped archives for a single recovery task. Therefore, Extract would not be able to read an archive tape that is being used to recover a table until the recovery is finished. You could use DFHSM or an equivalent tools to move the archive logs in a seamless manner between online DASD storage and tape, but Extract will have to wait until the transfer is finished. Delays in Extract processing increase the latency between source and target data.

Controlling Log Flushes

When reading the transaction log, Extract does not process a transaction until it captures the commit record. If the commit record is on a data block that is not full, it cannot be captured until more log activity is generated to complete the block. The API that is used by Extract to read the logs only retrieves full physical data blocks.

A delay in receiving blocks that contain commits can cause latency between the source and target data. If the applications are not generating enough log records to fill a block, Extract generates its own log records by issuing `SAVEPOINT` and `COMMIT` statements, until the block fills up one way or the other and is released.

In a data sharing group, each API call causes DB2 to flush the data blocks of all active members, eliminating the need for Extract to perform flushes.

To prevent Extract from performing flushes, use the Extract parameter `TRANLOGOPTIONS` with the `NOFLUSH` option.

Understanding What's Supported for DB2 for z/OS

This chapter contains information on database and table features supported byOracle GoldenGate for DB2 z/OS.

Topics:

Supported DB2 for z/OS Data Types

This section lists the DB2 for z/OS data types that Oracle GoldenGate supports and any limitations of this support.

- Oracle GoldenGate does not perform character set conversion for columns that could contain multi-byte data. This includes `GRAPHIC`, `VARGRAPHIC` and `DBCLOB` data types, as well as `CHAR`, `VARCHAR`, and `CLOB` for tables defined with `ENCODING_SCHEME` of '`M`' (multiple CCSID set or multiple encoding schemes) or '`U`' (Unicode). Such data is only supported if the source and target systems are the same CCSID.
- Oracle GoldenGate supports ASCII, EBCDIC, and Unicode data format. Oracle GoldenGate converts between ASCII and EBCDIC data automatically. Unicode is not converted.
- Oracle GoldenGate supports most DB2 data types except those listed in Non-Supported DB2 for z/OS Data Types.

Limitations of Support

- The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- Oracle GoldenGate does not support the filtering, column mapping, or manipulation of large objects greater than 4K in size. You can use the full Oracle GoldenGate functionality for objects that are 4K or smaller.
- Oracle GoldenGate supports the default `TIMESTAMP` and the `TIMESTAMP` with `TIMEZONE` to up to 9 digit fractional value, but no further.

Non-Supported DB2 for z/OS Data Types

This section lists DB2 for z/OS data types that Oracle GoldenGate does not support. Data that is not supported may affect the integrity of the target data in relation to the source data.

- `XML`
- User-defined types
- Negative dates

Supported Objects and Operations for DB2 z/OS

This section lists the database objects and types of operations that Oracle GoldenGate supports.

- Extraction and replication of DML operations on DB2 for z/OS tables that contain rows of up to 512KB in length. This size exceeds the maximum row size of DB2.
- `INSERT` operations from the IBM `LOAD` utility are supported for change capture if the utility is run with `LOG YES` and `SHRLEVEL CHANGE`, and the source tables that are being loaded have `DATA CAPTURE CHANGES` enabled (required by Oracle GoldenGate) and are specified in the Oracle GoldenGate Extract configuration. Oracle GoldenGate also supports initial loads with the `LOAD` utility to instantiate target tables during initial synchronization.
- Oracle GoldenGate supports the maximum number of columns per table, which is supported by the database.
- Oracle GoldenGate supports the maximum column size that is supported by the database.
- Extraction and replication of data that is stored using DB2 data compression (`CREATE TABLESPACE COMPRESS YES`).
- Capture from temporal history tables is supported.
- `TRUNCATE TABLE` is supported, but because this command issues row deletes to perform the truncate, they are shown in Oracle GoldenGate statistics as such, and not as a truncate operation. To replicate a `TRUNCATE` , the Replicat process uses a `DELETE` operation without a `WHERE` clause.
- `TRUNCATES` are always captured from a DB2 for z/OS source, but can be ignored by Replicat if the `IGNORETRUNCATES` parameter is used in the Replicat parameter file.
- `UNICODE` columns in `EBCDIC` tables are supported.
- Supported options with `SHOWTRANS`

  ```
  SHOWTRANS [transaction_ID]  [COUNT n]
  [DURATION duration unit]
  [FILE file_name] |
  ```

  `transaction_ID` and `count` cannot be specified together.

  `transaction_ID` and `duration` cannot be specified together.
- Options supported with `SKIPTRANS` and `FORCETRANS`:

  ```
  SKIPTRANS transaction_ID
  [FORCE] FORCETRANS transaction_ID  [FORCE]
  ```

Non-Supported Objects and Operations for DB2 for z/OS

The following objects and operations are not supported by Oracle GoldenGate on DB2 for z/OS:

- Extraction or replication of DDL operations
- Clone tables
- Data manipulation, including compression, that is performed within user-supplied DB2 exit routines, such as:
  - Date and time routines
  - Edit routines (`CREATE TABLE EDITPROC`)
  - Validation routines (`CREATE TABLE VALIDPROC`)
- Replicating with `BATCHSQL` is not fully functional for DB2 for z/OS. Non-insert operations are not supported so any update or delete operations will cause Replicat to drop temporarily out of `BATCHSQL` mode. The transactions will stop and errors will occur.

MySQL

With Oracle GoldenGate for MySQL, you can perform initial loads and capture transactional data and table changes from supported MySQL versions and replicate the data and table changes to a MySQL database or replicate the data to other supported Oracle GoldenGate targets, such as an Oracle Database.

Oracle GoldenGate for MySQL supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

This part describes tasks for configuring and running Oracle GoldenGate for MySQL and supported variants, such as MariaDB, Amazon RDS for MySQL, and Amazon Aurora MySQL.

Supported Databases

Oracle GoldenGate for MySQL supports capture and delivery for MySQL, Oracle MySQL Database Service, Amazon Aurora MySQL, Amazon RDS for MariaDB, Amazon RDS for MySQL, Azure Database for MySQL, and MariaDB.

For supported database versions, review the Certification Matrix.

Limitations of Support

Following are the limitations of support for Oracle GoldenGate for MySQL:

- MySQL databases enabled with binary log transaction compression are not supported with Oracle GoldenGate Extract.
- MySQL databases enabled with binary log encryption are not supported with Oracle GoldenGate Extract.

Database Storage Engine

Requirements for the database storage engine are as follows:

- Oracle GoldenGate supports the InnoDB storage engine for a source MySQL database.

- All the components of Oracle GoldenGate for MySQL, including Extract, Replicat, and GGSCI connect to the database using the MySQL native API.
- Oracle GoldenGate supports capture and apply from and to the InnoDB engine. Apply to MyISAM engine works, but there might be data integrity issues as MyISAM engine in non-transactional.

Database User for Oracle GoldenGate Processes for MySQL

Requirements for the database user for Oracle GoldenGate processes are as follows:

- Create a database user that is dedicated to Oracle GoldenGate. It can be the same user for all the Oracle GoldenGate processes that must connect to a database.
- To support DDL replication, the MySQL user must have privileges to install the database plug-ins. The required permissions for the plug-in is only required with MySQL 5.7. The `INSERT` privilege is required on the `mysql.plugin` system table.
- To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, the Oracle GoldenGate database user.
- Keep a record of the database users. They must be specified in the Oracle GoldenGate parameter files with the `USERID` parameter.
- The Oracle GoldenGate user requires read access to the `INFORMATION_SCHEMA` database.
- The Oracle GoldenGate user requires the following user privileges.

Privilege	**Source Extract**	**Target Replicat**	**Purpose**
`SELECT`	Yes	Yes	Connect to the database and select object definitions
`REPLICATION SLAVE`	Yes	NA	Connect and receive updates from the replication master's binary log.

`CREATE`
`CREATE VIEW`
`EVENT`
`INSERT`
`UPDATE`
`DELETE`
	Yes	Yes	Source and target database heartbeat and checkpoint table creation, and data record generation and purging

Privilege	Source Extract	Target Replicat	Purpose
`DROP`	Yes	Yes	Dropping a Replicat checkpoint table or deleting a heartbeat table implementation
`EXECUTE`	Yes	Yes	To execute stored procedures.
`INSERT`, `UPDATE`, `DELETE` on target tables	NA	Yes	Apply replicated DML to target objects
DDL privileges on target objects (if using DDL support)	NA	Yes	Issue replicated DDL on target objects

- To capture binary log events, an Administrator must provide the following privileges to the Extract user:
  - Read and Execute permissions for the directory where the MySQL configuration file (`my.cnf`) is located
  - Read permission for the MySQL configuration file (`my.cnf`).
  - Read and Execute permissions for the directory where the binary logs are located.
  - Read and Execute permission for the `tmp` directory. The `tmp` directory is `/tmp`. The MySQL database connection requires access to the `/tmp/mysql.sock` file for versions prior to MySQL 8.0.

Database Configuration

Learn about supported MySQL databases, required settings for configuring MySQL for Oracle GoldenGate.

Ensuring Data Availability

Retain enough binary log data so that if you stop Extract or there is an unplanned outage, Extract can start again from its checkpoints. Extract must have access to the binary log that contains the start of the oldest uncommitted unit of work, and all binary logs thereafter. The recommended retention period is at least 24 hours worth of transaction data, including both active and archived information. You might need to do some testing to determine the best retention time given your data volume and business requirements.

If data that Extract needs during processing was not retained, either in active or backup logs, one of the following corrective actions might be required:

- Alter Extract to capture from a later point in time for which binary log data is available (and accept possible data loss on the target).
- Resynchronize the source and target tables, and then start the Oracle GoldenGate environment over again.

To determine where the Extract checkpoints are, use the `INFO EXTRACT` command. For more information, see INFO EXTRACT in *Command Line Interface Reference for Oracle GoldenGate*.

Setting Logging Parameters

To capture from the MySQL transaction logs, the Oracle GoldenGate Extract process must be able to find the index file, which contains the paths of all binary log files.

Extract expects that all of the table columns are in the binary log. As a result, only `binlog_row_image` set as `full` is supported and this is the default. Other values of `binlog_row_image` are not supported.

> **Note:**
>
> Oracle recommends that the binary log is retained for at least 24 hours.

In MySQL 5.7, the `server_id` option must be specified along with `log-bin`, otherwise the server will not start. For MySQL 8.0, the `server_id` is enabled by default.

Extract checks the following parameter settings to get this index file path:

1. Extract `TRANLOGOPTIONS` parameter with the `ALTLOGDEST` option. If this parameter specifies a location for the log index file, Extract accepts this location over any default that is specified in the MySQL Server configuration file. When `ALTLOGDEST` is used, the binary log index file must also be stored in the specified directory. This parameter should be used if the MySQL configuration file does not specify the full index file path name, specifies an incorrect location, or if there are multiple installations of MySQL on the same machine. From Oracle GoldenGate 21c onward, `ALTLOGDEST` parameter is optional for local Extract, however, for remote Extract this parameter is mandatory. When `ALTLOGDEST` is not specified, the binary log index and binary log filepaths will be fetched from the database directly.Please note: The paths thus fetched are also subject to same accessibilitychecks as in the existing process.

   To specify the index file path using `TRANLOGOPTIONS` with `ALTLOGDEST` use a command similar to the following:

   ```
   TRANLOGOPTIONS ALTLOGDEST "/mnt/rdbms/mysql/data/logs/binlog.index"
   ```

   To capture from a remote server or in case of remote capture, you only need to specify the `REMOTE` option instead of the index file path on the remote server. For remote capture specify the following in the Extract parameter file:

   ```
   TRANLOGOPTIONS ALTLOGDEST REMOTE
   ```

2. The MySQL Server configuration file: The configuration file stores default startup options for the MySQL server and clients. On Windows, the name of the configuration file is `my.ini`. On other platforms, it is `my.cnf`. In the absence of `TRANLOGOPTIONS` with `ALTLOGDEST`, Extract gets information about the location of the log files from the configuration file. However, even with `ALTLOGDEST`, these Extract parameters must be set correctly:

   - `binlog-ignore-db=oggddl`: This prevents DDL logging history table entries in the `binlog` and is set in the `my.cnf` or `my.ini` file.

- `log-bin`: This parameter is used to enable binary logging. This parameter also specifies the location of the binary log index file and is a required parameter for Oracle GoldenGate, even if `ALTLOGDEST` is used. If log-bin is not specified, binary logging will be disabled and Extract returns an error.
- `log-bin-index`: This parameter specifies the location of the binary log index. If it is not used, Extract assumes that the index file is in the same location as the log files. If this parameter is used and specifies a different directory from the one that contains the binary logs, the binary logs must not be moved once Extract is started.
- `max_binlog_size`: This parameter specifies the size, in bytes, of the binary log file.

  > **Note:**
  >
  > The server creates a new binary log file automatically when the size of the current log reaches the `max_binlog_size` value, unless it must finish recording a transaction before rolling over to a new file.

- `binlog_format`: This parameter sets the format of the logs. It must be set to the value of `ROW`, which directs the database to log DML statements in binary format. Extract silently ignores the `binlog` events that are not written in the `ROW` format instead of abending when it detects a `binlog_format` other than `ROW`.

  > **Note:**
  >
  > MySQL binary logging does not allow logging to be enabled or disabled for specific tables. It applies globally to all tables in the database.

- `mysql.rds_set_configuration`: When capturing from MySQL Amazon RDS instance, you need to call the `mysql.rds_set_configuraton` stored procedure on MySQL command line, to retain the binary logs for a specific duration. By default, the default value of `binlog_retention_hours` for MySQL Amazon RDS is set to NULL, which implies that the binary logs are not retained.

  The following example shows the command to preserve the binary log for 24 hours:

  ```
  mysql > call mysql.rds_set_configuration('binlog retention hours', 24);
  ```

To locate the configuration file, Extract checks the `MYSQL_HOME` environment variable: If `MYSQL_HOME` is set, Extract uses the configuration file in the specified directory. If `MYSQL_HOME` is not set, Extract queries the `information_schema.global_variables` table to determine the MySQL installation directory. If a configuration file exists in that directory, Extract uses it.

3. For MariaDB version 10.2 and later, Oracle GoldenGate works in the same way as for MySQL but a new variable needs to be configured in the `my.cnf` or `my.ini` file. The variable that needs to be added is `"binlog-annotate-row-events=OFF".` Restart MariaDB after configuring this variable and then start the Extract process.

Database Connection

Oracle GoldenGate gets the name of the database it is supposed to connect to from the `SOURCEDB` parameter. To configure the connection for the `SOURCEDB` parameter, use the following format:

```
SOURCEDB dbname@hostname:port, USERID mysqluser, PASSWORD welcome
```

The `dbname` is the name of the MySQL instance, `hostname` is the name or IP address of the MySQL database server, `port` is the port number of the MySQL instance. If using an unqualified host name, that name must be properly configured in the DNS database. Otherwise, use the fully qualified host name, for example `myhost.company.com`.

Setting the Session Character Set

The `GGSCI`, Extract and Replicat processes use a session character set when connecting to the database. For MySQL, the session character set is taken from the `SESSIONCHARSET` option of `SOURCEDB` and `TARGETDB`. Make certain you specify a session character set in one of these ways when you configure Oracle GoldenGate.

Changing the Log-Bin Location

Modifying the binary log location by using the `log-bin` variable in the MySQL configuration file might result in two different path entries inside the index file, which could result in errors. To avoid any potential errors, change the log-bin location by doing the following:

1. Stop any new DML operations.
2. Let the extract finish processing all of the existing binary logs. You can verify this by noting when the checkpoint position reaches the offset of the last log.
3. After Extract finishes processing the data, stop the Extract group and, if necessary, back up the binary logs.
4. Stop the MySQL database.
5. Modify the `log-bin` path for the new location.
6. Start the MySQL database.
7. To clean the old log name entries from index file, use `flush master` or `reset master` (based on your MySQL version).
8. Start Extract.

Configuring MySQL for Remote Capture

Oracle GoldenGate remote capture for MySQL, Amazon RDS for MySQL, Amazon Aurora MySQL, Azure Database for MySQL are used to capture transaction log data from a database located remotely to the Oracle GoldenGate installation.

Database Server Configuration

For remote capture to work, configure the MySQL server as follows:

1. Grant access permissions to the Oracle GoldenGate remote capture user.

Run the following statements against the remote database to create the user and grant the permissions needed for remote capture.

```
mysql > CREATE USER 'username'@'host' IDENTIFIED BY 'Password';
mysql > GRANT ALL PRIVILEGES ON *.* TO 'username'@'host' WITH GRANT
OPTION;
mysql > FLUSH PRIVILEGES;
```

2. The `server_id` value of the remote MySQL server should be greater than 0. This value can be verified by issuing the following statement on the MySQL remote server:

   ```
   mysql > show variables like ‘server_id’;
   ```

   If the `server_id` value is 0, modify the `my.cnf` configuration file to set to a value greater than 0.

Oracle GoldenGate Configuration

Oracle GoldenGate configuration has the following steps:

1. Provide the remote database's connection information in the Extract's parameter file.

   ```
   SOURCEDB remotedb@mysqlserver.company.com:port, USERID username, PASSWORD
   password
   ```

2. Add the following parameter to the Extract's parameter file, after the connection information.

   ```
   TRANLOGOPTIONS ALTLOGDEST REMOTE
   ```

Limitations of Oracle GoldenGate Remote Capture for MySQL

Co-existence of Oracle GoldenGate for MySQL remote capture with the MySQL’s native replication slave is supported with following conditions and limitations:

- The native replication slave should be assigned a different server_id than the currently running slaves. The slave server_id values can be seen using the following MySQL command on the master server.

  ```
  mysql> show slave hosts;
  ```

  - If the Oracle GoldenGate capture abends with error "`A slave with the same server_uuid or server_id as this slave has connected to the master`", then change the capture's name and restart the capture.
  - If the native replication slave dies with the error "`A slave with the same server_uuid or server_id as this slave has connected to the master`", then change the native replication slave’s `server_id` and restart it.
- Remote capture is supported for Oracle GoldenGate on running on Linux and can support databases running on Linux or Windows.

Capturing using a MySQL Replication Slave

You can configure a MySQL replication slave to capture the master's binary log events from the slave.

Typically, the transactions applied by the slave are logged into the relay logs and not into the slave's `binlog`. For the slave to write transactions in its `binlog`, that it receives from the master , you must start the replication slave with the `log-slave-updates` option as `1` in `my.cnf` in conjunction with the other binary logging parameters for Oracle GoldenGate. After the master's transactions are in the slave's `binlog` , you can set up a regular Oracle GoldenGate capture on the slave to capture and process the slave's `binlog`.

Database Character Set

MySQL provides a facility that allows users to specify different character sets at different levels.

Level	Example
Database	`create database test charset utf8;`
Table	`create table test( id int, name char(100)) charset utf8;`
Column	`create table test ( id int, name1 char(100) charset gbk, name2 char(100) charset utf8));`

Limitations of Support

- When you specify the character set of your database as utf8mb4/utf8, the default collation is `utf8mb4_unicode_ci`/`utf8_general_ci`. If you specify `collation_server=utf8mb4_bin`, the database interprets the data as binary. For example, specifying the `CHAR` column length as four means that the byte length returned is 16 (for `utf8mb4`) though when you try to insert data more than four bytes the target database warns that the data is too long. This is the limitation of database so Oracle GoldenGate does not support binary collation. To overcome this issue, specify `collation_server=utf8mb4_bin` when the character set is utf8mb4 and `collation_server=utf8_bin` for UTF-8.
- The following character sets are **not** supported:

  `armscii8`
  `keybcs2`
  `utf16le`
  `geostd8`

Prepare Database Connection

Learn about preparing MySQL database connections from Oracle GoldenGate Classic Architecture.

Configuring a Two-way SSL Connection in MySQL Capture and Delivery

To use the two way SSL in Oracle GoldenGate for MySQL capture and delivery, you need to supply the full paths of the certificate authority (`ca.pem`), the client certificate (`client-cert.pem`) and the client key (`client-key.pem`) files to the capture and delivery.

To know more about generating the certificate files, see:

https://dev.mysql.com/doc/refman/5.7/en/creating-ssl-rsa-files-using-mysql.html

You need to provide these paths in the Extract and Replicat parameter files using the `SETENV` parameter.

Following are the `SETENV` environment parameters to set the two-way SSL connection:

- `OGG_MYSQL_OPT_SSL_CA`: Sets the full path of the certification authority.
- `OGG_MYSQL_OPT_SSL_CERT`: Sets the full path of the client certificate.
- `OGG_MYSQL_OPT_SSL_KEY`: Sets the full path of the client key.

In the following example, the MySQL SSL certificate authority, client certificate, and client key paths are set to the Oracle GoldenGate MySQL Extract and Replicat parameter:

```
SETENV (OGG_MYSQL_OPT_SSL_CA='/var/lib/mysql.pem')
SETENV (OGG_MYSQL_OPT_SSL_CERT='/var/lib/mysql/client-cert.pem')
SETENV (OGG_MYSQL_OPT_SSL_KEY='/var/lib/mysql/client-key.pem')
```

For a MySQL user configured with X509 encryption scheme, the MySQL database requires the `ssl-key` and `ssl-cert` options at the time of logging in. So, when an Oracle GoldenGate credential store entry is created for this user, the SSL options in the credential store alias must mandatorily include `sslKey` and `sslCert` regardless of `sslMode` used.

Preparing Tables for Processing

This section describes how to prepare the tables for processing. Table preparation requires these tasks:

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

   > **Note:**
   >
   > If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any

existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Tables with a Primary Key Derived from a Unique Index

In the absence of a primary key on a table, MySQL will promote a unique index to primary key if the indexed column is `NOT NULL`. If there are more than one of these not-null indexes, the first one that was created becomes the primary key. To avoid Replicat errors, create these indexes in the same order on the source and target tables.

For example, assume that source and target tables named `ggvam.emp` each have columns named first, middle, and last, and all are defined as `NOT NULL`. If you create unique indexes in the following order, Oracle GoldenGate will abend on the target because the table definitions do not match.

Source:

```
mysql> create unique index uq1 on ggvam.emp(first);
mysql> create unique index uq2 on ggvam.emp(middle);
mysql> create unique index uq3 on ggvam.emp(last);
```

Target:

```
mysql> create unique index uq1 on ggvam.emp(last);
mysql> create unique index uq2 on ggvam.emp(first);
mysql> create unique index uq3 on ggvam.emp(middle);
```

The result of this sequence is that MySQL promotes the index on the source "first" column to primary key, and it promotes the index on the target "last" column to primary key. Oracle GoldenGate will select the primary keys as identifiers when it builds its metadata record, and the metadata will not match. To avoid this error, decide which column you want to promote to primary key, and create that index first on the source and target.

How to Specify Your Own Key for Oracle GoldenGate to Use

If a table does not have one of the preceding types of row identifiers, or if you prefer those identifiers not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds.

Limiting Row Changes in Tables That Do Not Have a Key

If a target table does not have a primary key or a unique key, duplicate rows can exist. In this case, Oracle GoldenGate could update or delete too many target rows, causing the source and target data to go out of synchronization without error messages to alert you. To limit the number of rows that are updated, use the `DBOPTIONS` parameter with the `LIMITROWS` option in the Replicat parameter file. `LIMITROWS` can increase the performance of Oracle GoldenGate on the target system because only one row is processed.

Triggers and Cascade Constraints Considerations

Triggers

Disable triggers on the target tables, or alter them to ignore changes made by the Oracle GoldenGate database user. Oracle GoldenGate replicates DML that results from a trigger. If the same trigger gets activated on the target table, then it becomes redundant because of the replicated version, and the database returns an error.

Cascade Constraints Considerations

Cascading updates and deletes captured by Oracle GoldenGate are not logged in binary log, so they are not captured. This is valid for both MySQL and MariaDB. For example, when you run the delete statement in the parent table with a parent child relationship between tables, the cascading deletes (if there are any) happens for child table, but they are not logged in binary log. Only the delete or update record for the parent table is logged in the binary log and captured by Oracle GoldenGate.

See https://mariadb.com/kb/en/replication-and-foreign-keys/ and https://dev.mysql.com/doc/refman/8.0/en/innodb-and-mysql-replication.html for details.

To properly handle replication of cascading operations, it is recommended to disable cascade deletes and updates on the source and code your application to explicitly delete or update the child records prior to modifying the parent record. Alternatively, you must ensure that the target parent table has the same cascade constraints configured as the source parent table, but this could lead to an out-of-sync condition between source and target, especially in cases of bi-directional replication.

Understanding What's Supported for MySQL

This chapter contains information on database and table features supported by Oracle GoldenGate.

Database Character Set

MySQL provides a facility that allows users to specify different character sets at different levels.

Level	Example
Database	`create database test charset utf8;`
Table	`create table test( id int, name char(100)) charset utf8;`
Column	`create table test ( id int, name1 char(100) charset gbk, name2 char(100) charset utf8));`

Limitations of Support

- When you specify the character set of your database as utf8mb4/utf8, the default collation is `utf8mb4_unicode_ci`/`utf8_general_ci`. If you specify `collation_server=utf8mb4_bin`, the database interprets the data as binary. For example, specifying the `CHAR` column length as four means that the byte length returned is 16 (for `utf8mb4`) though when you try to insert data more than four bytes the target database warns that the data is too long. This is the limitation of database so Oracle GoldenGate does not support binary collation. To overcome this issue, specify `collation_server=utf8mb4_bin` when the character set is utf8mb4 and `collation_server=utf8_bin` for UTF-8.
- The following character sets are **not** supported:

  `armscii8`
  `keybcs2`
  `utf16le`
  `geostd8`

Oracle GoldenGate for MySQL Supported Data Types

Oracle GoldenGate for MySQL supports the following data types:

- `BLOB`
- `BIGINT`
- `BINARY`
- `BIT(M)`
- `CHAR`
- `DATE`
- `DATETIME`
- `DECIMAL`
- `DOUBLE`
- `ENUM`
- `FLOAT`
- `INT`
- `JSON`
- `LONGBLOB`
- `LONGTEXT`

- `MEDIUMBLOB`
- `MEDIUMINT`
- `MEDIUMTEXT`
- `SMALLINT`
- `TEXT`
- `TIME`
- `TIMESTAMP`
- `TINYBLOB`
- `TINYINT`
- `TINYTEXT`
- `VARBINARY`
- `VARCHAR`
- `YEAR`

Limitations and Clarifications

When running Oracle GoldenGate for MySQL, be aware of the following:

- Functional indexes are not supported for Capture or Delivery.
- Oracle GoldenGate does not support `BLOB` or `TEXT` types when used as a primary key.
- Oracle GoldenGate supports a `TIME` type range from 00:00:00 to 23:59:59.
- Oracle GoldenGate supports timestamp data from `0001/01/03:00:00:00` to `9999/12/31:23:59:59`. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the time zone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.
- Oracle GoldenGate does not support negative dates.
- The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- When you use `ENUM` type in non-strict `sql_mode`, the non-strict `sql_mode` does not prevent you from entering an invalid `ENUM` value and an error will be returned. To avoid this situation, do one of the following:
  - Use `sql_mode` as `STRICT` and restart Extract. This prevents users from entering invalid values for any of the data types. An IE user can only enter valid values for those data types.
  - Continue using non-strict `sql_mode`, but do not use `ENUM` data types.
  - Continue using non-strict `sql_mode` and use `ENUM` data types with valid values in the database. If you specify invalid values, the database will silently accept them and Extract will abend.
- Table with single column is not supported for JSON datatype. Extract will abend in case it is configured for a table which has a single column of `JSON` datatype.

- `JSON` datatype does not support CDR. The following message gets logged in the report file if `GETBEFORECOLS` is configured and the table has columns of `JSON` datatypes:

  ```
  INFO OGG-06556 The following columns will not be considered for CDR
  ```

  The limtations for CDR applies to cases where the `GETBEFORECOLS` and `COMPARECOLS` are used.

Non-Supported MySQL Data Types

Oracle GoldenGate for MySQL does not support the following data types:

All spatial types (Geometry and so on), `JSON`, `SET`, `XML`

> **Note:**
>
> Extract abends if it is configured to capture from tables that contain any of the unsupported data types, so ensure that Extract is not configured to capture from tables containing columns of unsupported data types.

Supported Objects and Operations for MySQL

Oracle GoldenGate for MySQL supports the following objects and operations:

- Oracle GoldenGate supports the following DML operations on source and target database transactional tables:
  - Insert operation
  - Update operation (compressed included)
  - Delete operation (compressed included)
  - Truncate operation
- Oracle GoldenGate supports the extraction and replication of DDL (data definition language) operations.
- Oracle GoldenGate supports transactional tables up to the full row size and maximum number of columns that are supported by MySQL and the database storage engine that is being used. InnoDB supports up to 1017 columns.
- Generated columns are supported and captured.
- Oracle GoldenGate supports the `AUTO_INCREMENT` column attribute. The increment value is captured from the binary log by Extract and applied to the target table in a Replicat insert operation.
- Oracle GoldenGate can operate concurrently with MySQL native replication.
- Oracle GoldenGate supports the `DYNSQL` feature for MySQL.

> **Note:**
>
> XA transactions are not supported for capture and any XA transactions logged in `binlog` cause Extract to abend. So, you must not use XA transactions against a database that Extract is configured to capture.
> If XA transactions are being used for databases that are not configured for Oracle GoldenGate capture, then exclude those databases from logging into MySQL binary logs by using the parameter `binlog-ignore-db` in the MySQL server configuration file.

Limitations on Automatic Heartbeat Table support are as follows:

  - Ensure that the database in which the heartbeat table is to be created already exists to avoid errors when adding the heartbeat table.
  - In the heartbeat history lag view, the information in fields like `heartbeat_received_ts`, `incoming_heartbeat_age`, and `outgoing_heartbeat_age` are shown with respect to the system time. You should ensure that the operating system time is setup with the correct and current time zone information.

- Position by End of File (EOF) is supported in MySQL. Oracle GoldenGate Extract for MySQL finds the position corresponding to the end of the file and starts reading transactions from there. The EOF position is not exact, if data is continuously written to the binary log.

  The Extract is added and altered using:

  ```
  ADD EXTRACT group_name, TRANLOG, EOF


  ALTER EXTRACT group_name, EOF
  ```

Non-Supported Objects and Operations

Oracle GoldenGate for MySQL does not support the following objects and operations:

- Invisible columns
- The Oracle GoldenGate `BATCHSQL` feature.
- Array fetching during initial load.
- The following character sets are not supported:

  ULIB_CS_ARMSCII8, /* American National 166-9 */
  ULIB_CS_GEOSTD8, /* Geogian Standard */
  ULIB_CS_KEYBCS2, /* Kemennicky MS-DOS
- Capturing NLS LOB data using the `FETCHMOCOLS` and `FETCHMODCOLEXCEPT` TABLE options is not supported when DDL is enabled.
- Renaming tables.
- DDL statements inside stored procedures is not supported.
- When the time zone of the Oracle GoldenGate installation server does not match the time zone of the source database server, then the TIMESTAMP data sent to the target database will differ from the source database. For Oracle GoldenGate Microservices installations,

regardless of the time zones being the same, Extract will resolve the time zone to UTC. Determine the source database time zone by running the following query:

```
select @@system_time_zone;
```

This will return a time zone value, such as PDT.

Create a session variable for Oracle GoldenGate, called `TZ`, and set it equal to the time zone value of the database.

- Extraction and replication from and to views is not supported.
- Transactions applied by the slave are logged into the relay logs and not into the slave's `binlog`. If you want a slave to write transactions the `binlog` that it receives from the master , you need to start the replication slave with the log slave-updates option as 1 in `my.cnf`. This is in addition to the other binary logging parameters. After the master's transactions are in the slave's `binlog`, you can then setup a regular capture on the slave to capture and process the slave's `binlog`.

System Schemas

The following schemas or objects are not automatically replicated by Oracle GoldenGate unless they are explicitly specified without a wildcard.

- `'information_schema'`
- `'performance_schema'`
- `'mysql'`
- `'sys'`

Oracle

Prepare your database for Oracle GoldenGate, including configuring connections and logging, enabling Oracle GoldenGate in your database, setting up the flashback query, and managing server resources.

Preparing the Database for Oracle GoldenGate

Learn how to prepare your database for Oracle GoldenGate, including how to configure connections and logging, how to enable Oracle GoldenGate in your database, how to set the flashback query, and how to manage server resources.

Database Requirements

Learn about the Oracle GoldenGate requirements for Oracle Database. These apply to both the capture modes unless explicitly noted.

- If you are using the `DBMS_LOB.LOADFROMFILE` procedure to update a LOB column only and your supplemental log is on all the columns, Integrated Extract captures the key columns and LOB improving performance.
- Ensure that your database has minimal supplemental logging enabled.
- Database user privileges and configuration requirements are explained in Oracle Database Privileges.

- If the database is configured to use a bequeath connection, the `sqlnet.ora` file must contain the `bequeath_detach=true` setting.
- Oracle Databases must be in `ARCHIVELOG` mode so that Extract can process the log files.
- Oracle Databases must be in `FORCE LOGGING` mode to ensure that all transactional data is written to Redo.
- Disk space is also required for the Oracle GoldenGate Bounded Recovery feature. Bounded Recovery is a component of the general Extract checkpointing facility. It caches long-running open transactions to disk at specific intervals to enable fast recovery upon a restart of Extract. At each bounded recovery interval (controlled by the `BRINTERVAL` option of the `BR` parameter) the disk required is as follows: for each transaction with cached data, the disk space required is usually 64k plus the size of the cached data rounded up to 64k. Not every long-running transaction is persisted to disk. For complete information about Bounded Recovery, see the BR parameter in *Parameters and Functions Reference for Oracle GoldenGate*.

Configuring Connections for Extract and Replicat Processes

Extract and Replicat require a dedicated server connection in the `tnsnames.ora` file.

Before you begin, make sure that there is a dedicated user on the Oracle database side, with the required privileges. See Grant User Privileges for Oracle Database for Oracle GoldenGate Classic Architecture.

On the Oracle GoldenGate side, you direct the Extract and Replicat processes to use these connections by specifying the values for `USERID` or `USERIDALIAS` parameter in the Extract and Replicat parameter files.

The following are the security options for specifying the connection string in the Extract or Replicat parameter file.

Credential store method:

```
USERIDALIAS ggeast
```

In the case of `USERIDALIAS`, the alias `ggeast` is stored in the Oracle GoldenGate credential store with the actual connection string. The following example uses the `INFO CREDENTIALSTORE` command to display the details of the credentials configured in Oracle GoldenGate:

```
INFO CREDENTIALSTORE DOMAIN OracleGoldenGate
```

Output:

```
Domain: OracleGoldenGate
  Alias: ggeast
  Userid: ggadmin@dc1.example.com:1521/DBEAST.example.com
```

Setting up a Bequeath connection

Oracle GoldenGate can connect to a database instance without using the network listener if a Bequeath connect descriptor is added in the `tnsnames.ora`.

The following example shows the configuration for connecting to a database using Bequeath connect descriptor:

```
dbbeq =  (DESCRIPTION=
       (ADDRESS=(PROTOCOL=beq)
           (ENVS='ORACLE_SID=sales,ORACLE_HOME=/app/db_home/
oracle,LD_LIBRARY_PATH=/app/db_home/oracle/lib')
           (PROGRAM=/app/db_home/oracle/bin/oracle)
       (ARGV0=oraclesales)
       (ARGS='(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))'))
         (CONNECT_DATA=(SID=sales)))
```

In this example:

`/app/db_home` is the target Oracle database installation directory

`sales` is the database service name

The `ORACLE_SID`, `ORACLE_HOME`, and `LD_LIBRARY_PATH` in the `ENVS` parameter refers to the target.

> **Note:**
>
> Make sure that there is no white space between these environment variable settings.

Configuring Oracle GoldenGate in a Multitenant Container Database

This chapter contains additional configuration instructions when configuring Oracle GoldenGate using the per-PDB capture mode or the CDB root capture mode.

Using the Root Container Extract from PDB

To capture from a multitenant database, you must use an Extract that is configured at the root level using a `c##` account. To apply data into a multitenant database, a separate Replicat is needed for each PDB, because a Replicat connects at the PDB-level and doesn't have access to objects outside of that PDB

One Extract group can capture from multiple pluggable databases to a single trail. In the parameter file, source objects must be specified in `TABLE` and `SEQUENCE` statements with their fully qualified three-part names in the format of *`container.schema.object`*.

As an alternative to specifying three-part names, you can specify a default pluggable database with the `SOURCECATALOG` parameter, and then specify only the *`schema.object`* in subsequent `TABLE` or `SEQUENCE` parameters. You can use multiple instances of this configuration to handle multiple source pluggable databases. For example:

```
SOURCECATALOG pdb1
TABLE phoenix.tab;
SEQUENCE phoenix.seq;
SOURCECATALOG pdb2
TABLE dallas.tab;
SEQUENCE dallas.seq;
```

Applying to Pluggable Databases

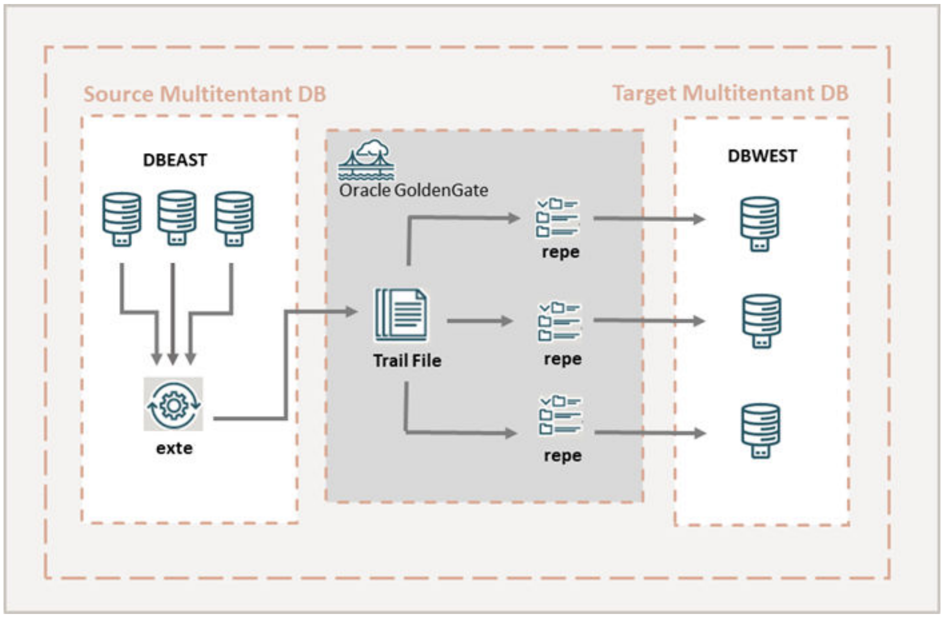
Replicat can only connect and apply to one pluggable database. To specify the correct one, use a SQL*Net connect string for the database user that you specify with the `USERID` or `USERIDALIAS` parameter. For example: `GGADMIN@FINANCE`. In the parameter file, specify only the *`schema.object`* in the `TARGET` portion of the `MAP` statements. In the `MAP` portion, identify source objects captured from more than one pluggable database with their three-part names or use the `SOURCECATALOG` parameter with two-part names. The following is an example of this configuration.

```
SOURCECATALOG pdb1
MAP schema_1.tab, TARGET 1;
MAP schema_1.seq, TARGET 1;
SOURCECATALOG pdb2
MAP schema_2.tab, TARGET 2;
MAP schema_2.seq, TARGET 2;
```

The following is an example *without* the use of `SOURCECATALOG` to identify the source pluggable database. In this case, the source objects are specified with their three-part names.

```
MAP pdb1.schema_1.tab, TARGET 1;
MAP pdb1.schema_1.seq, TARGET 1;
```

To configure replication from multiple source pluggable databases to multiple target pluggable databases, you can configure parallel Extract and Replicat streams, each handling data for one pluggable database. Alternatively, you can configure one Extract capturing from multiple source pluggable databases, which writes to one trail that is read by multiple Replicat groups, each applying to a different target pluggable database. Yet another alternative is to use one Extract writing to multiple trails, each trail read by a Replicat assigned to a specific target pluggable database :



Excluding Objects from the Configuration

To exclude pluggable databases, schemas, and objects from the configuration, you can use the `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `TABLEEXCLUDE`, `MAPEXCLUDE`, and `EXCLUDEWILDCARDOBJECTSONLY` parameters.

Requirements for Configuring Container Databases for Oracle GoldenGate

This topic describes the special requirements that apply to replication to and from multitenant container databases.

The requirements are:

- The different pluggable databases in the multitenant container database can have different character sets. Oracle GoldenGate captures data from any multitenant database with different character sets into one trail file and replicates the data without corruption due to using different character sets.
- Extract must operate in integrated capture mode. See Deciding Which Extract Method to Use for more information about Extract capture modes. Replicat can operate in any of its modes.
- Extract must connect to the root container (`cdb$root`) as a common user in order to interact with the logmining server. To specify the root container, use the appropriate SQL*Net connect string for the database user that you specify with the `USERID` or `USERIDALIAS` parameter. For example: `C##GGADMIN@FINANCE`. See Establishing Oracle GoldenGate Credentials for how to create a user for the Oracle GoldenGate processes and grant the correct privileges.
- To support source CDB 12.2, Extract must specify the trail format as release 12.3. Due to changes in the redo logs, to capture from a multitenant database that is Oracle 12.2 or higher, the trail format release must be 12.3 or higher.
- The `dbms_goldengate_auth.grant_admin_privilege` package grants the appropriate privileges for capture and apply within a multitenant container database. This includes the `container` parameter, which must be set to `ALL`, as shown in the following example:

  ```
  exec dbms_goldengate_auth.grant_admin_privilege('C##GGADMIN',container=>'all')
  ```

- DDL replication works as a normal replication for multitenant databases. However, DDL on the root container should not be replicated because Replicats must not connect to the root container, only to PDBs.

Configuring Logging Properties

Oracle GoldenGate relies on the redo logs to capture the data that it needs to replicate source transactions, and these redo logs on the source system must be configured properly before you start Oracle GoldenGate processing.

This section addresses the following logging levels that apply to Oracle GoldenGate. The logging level that you use depends on Oracle GoldenGate features that you are using.

> **Note:**
>
> Redo volume is increased as the result of this required logging. You can wait until you are ready to start Oracle GoldenGate processing to enable the logging.

This table shows the Oracle GoldenGate use cases for the different logging properties.

Logging option	Command	What it does	Use case
Forced logging mode	`ALTER DATABASE FORCE LOGGING;`	Forces the logging of all transactions and loads.	Strongly recommended for all Oracle GoldenGate use cases. `FORCE LOGGING` overrides any table-level `NOLOGGING` settings.
Minimum database-level supplemental logging	`ALTER DATABASE ADD SUPPLEMENTAL LOG DATA`	Enables minimal supplemental logging to add row-chaining information to the redo log.	Required for all Oracle GoldenGate use cases
Schema-level supplemental logging, default setting See Enabling Schema-level Supplemental Logging.	`ADD SCHEMATRANDATA`	Enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of all tables in a schema. All of these keys together are known as the *scheduling columns*.	Enables the logging for all current and future tables in the schema. If the primary key, unique key, and foreign key columns are not identical at both source and target, use `ALLCOLS`. Required when using DDL support.
Schema-level supplemental logging with unconditional logging for all supported columns. (See Enabling Schema-level Supplemental Logging for non-supported column types.)	`ADD SCHEMATRANDATA` with `ALLCOLS` option	Enables unconditional supplemental logging of all of the columns in a table, for all of the tables in a schema.	Used for bidirectional and active-active configurations where all column values are checked, not just the changed columns, when attempting to perform an update or delete. This takes more resources though allows for the highest level of real-time data validation and thus conflict detection. This method should also be used if they are going to be using the `HANDLECOLLISIONS` parameter for initial loads.
Schema-level supplemental logging, minimal setting	`ADD SCHEMATRANDATA` with `NOSCHEDULINGCOLS` option	Enables unconditional supplemental logging of the primary key and all valid unique indexes of all tables in a schema.	Use only for nonintegrated Replicat. This is the minimum required schema-level logging.
Table-level supplemental logging with built-in support for integrated Replicat See Enabling Table-level Supplemental Logging	`ADD TRANDATA`	Enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of a table. All of these keys together are known as the *scheduling columns*.	Required for all Oracle GoldenGate use cases unless schema-level supplemental logging is used. If the primary key, unique key, and foreign key columns are not identical at both source and target, use `ALLCOLS`.

Logging option	Command	What it does	Use case
Table-level supplemental logging with unconditional logging for all supported columns. (See Enabling Table-level Supplemental Logging for non-supported column types.)	`ADD TRANDATA` with `ALLCOLS` option	Enables unconditional supplemental logging of all of the columns of the table.	Used for bidirectional and active-active configurations where all column values are checked, not just the changed columns, when attempting to perform an update or delete. This takes more resources though allows for the highest level of real-time data validation and thus conflict detection. It can also be used when the source and target primary, unique, and foreign keys are not the same or are constantly changing between source and target.
Table-level supplemental logging, minimal setting	`ADD TRANDATA` with `NOSCHEDULINGCOLS` option	Enables unconditional supplemental logging of the primary key and all valid unique indexes of a table.	Use for nonintegrated Replicat and non-parallel Replicat. This is the minimum required table-level logging.

Note:

Oracle Databases must be in `ARCHIVELOG` mode so that Extract can process the log files.

Enabling Subset Database Replication Logging

Oracle strongly recommends putting the Oracle source database into forced logging mode. Forced logging mode forces the logging of all transactions and loads, overriding any user or storage settings to the contrary. This ensures that no source data in the Extract configuration gets missed.

There is a fine-granular database supplemental logging mode called Subset Database Replication available in LogMiner, which is the basic recommended mode for all Oracle GoldenGate and XStream clients. It replaces the previously used Minimum Supplemental Logging mode.

To know more, see ALTER DATABASE in the *Oracle Database SQL Language Reference*.

The subset database replication logging is enabled at `CDB$ROOT` (and all user-PDBs inherit it) currently.

> **Note:**
>
> Database-level primary key (PK) and unique index (UI) logging is only discouraged if you are replicating a subset of tables. You can use it with Live Standby, or if Oracle GoldenGate is going to replicate all tables, like to reduce the downtime for a migration or upgrade.

Perform the following steps to verify and enable, if necessary, subset database replication logging and forced logging.

1. Log in to SQL*Plus as a user with `ALTER SYSTEM` privilege.
2. Issue the following command to determine whether the database is in supplemental logging mode and in forced logging mode. If the result is `YES` for both queries, the database meets the Oracle GoldenGate requirement.

   ```
   SELECT SUPPLEMENTAL_LOG_DATA_MIN, FORCE_LOGGING FROM V$DATABASE;
   ```

3. If the result is `NO` for either or both properties, continue with these steps to enable them as needed:

   ```
   ALTER PLUGGABLE DATABASE pdbname ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE
   REPLICATION;;
   ALTER DATABASE FORCE LOGGING;
   ```

4. Issue the following command to verify that these properties are now enabled.

   ```
   SELECT SUPPLEMENTAL_LOG_DATA_MIN, FORCE_LOGGING FROM V$DATABASE;
   ```

   The output of the query must be `YES` for both properties.
5. Switch the log files.

   ```
   ALTER SYSTEM SWITCH LOGFILE;
   ```

To perform dictionary validation, run the following command:

```
SELECT con_id, MINIMAL, SUBSET_REP, PRIMARY_KEY, UNIQUE_INDEX, FOREIGN_KEY,
ALL_COLUMN FROM CDB_SUPPLEMENTAL_LOGGING;
```

The output of this query should be `YES`.

```
SUBSET_REP = YES
```

For the following query:

```
SELECT NAME, LOG_MODE, FORCE_LOGGING, SUPPLEMENTAL_LOG_DATA_MIN,
SUPPLEMENTAL_LOG_DATA_PK PK, SUPPLEMENTAL_LOG_DATA_UI UI,
SUPPLEMENTAL_LOG_DATA_FK FK,
SUPPLEMENTAL_LOG_DATA_ALL,
SUPPLEMENTAL_LOG_DATA_SR FROM V$DATABASE;
```

For the query for `SUPPLEMENTAL_LOG_DATA_SR` the output should be `YES` and for `SUPPLEMENTAL_LOG_DATA_MIN` the output should be `IMPLICIT`.

To switch from earlier minimum supplemental logging to the new subset supplemental logging:

1. Drop the earlier higher levels on `CDB$ROOT`.

   ```
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
   ```

2. Add only `subset database replication` mode:

   ```
   ALTER PLUGGABLE DATABASE pdbname ADD SUPPLEMENTAL LOG DATA SUBSET DATABASE
   REPLICATION;
   ```

3. Ensure that all PDBs inherit this `subset database replication` mode.

Enabling Schema-level Supplemental Logging

Oracle GoldenGate supports schema-level supplemental logging. Schema-level logging is required for an Oracle source database when using the Oracle GoldenGate DDL replication feature. In all other use cases, it is optional, but then you must use table-level logging instead.

By default, schema-level logging automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of all tables in a schema. Options enable you to alter the logging as needed.

> **Note:**
>
> Oracle strongly recommends using schema-level logging rather than table-level logging, because it ensures that any new tables added to a schema are captured if they satisfy wildcard specifications. This method is also recommended because any changes to key columns are automatically reflected in the supplemental log data too. For example, if a key changes, there is no need to issue `ADD TRANDATA`.

Perform the following steps on the source system to enable schema-level supplemental logging.

1. Start the command line on the source system.
2. Issue the `DBLOGIN` command with the alias of a user in the credential store who has privilege to enable schema-level supplemental logging.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   See `USERIDALIAS` in *Parameters and Functions Reference for Oracle GoldenGate* for more information about `USERIDALIAS` and additional options.

3. When using `ADD SCHEMATRANDATA` or `ADD TRANDATA` on a multitenant database, you can either log directly into the PDB and perform the command. Alternately, if you are logging in at the root level (using a `C##` user), then you must include the PDB. Issue the `ADD`

`SCHEMATRANDATA` command for each schema for which you want to capture data changes with Oracle GoldenGate.

```
ADD SCHEMATRANDATA pdb.schema [ALLCOLS | NOSCHEDULINGCOLS]
```

Where:

- Without options, `ADD SCHEMATRANDATA` schema enables the unconditional supplemental logging on the source system of the primary key and the conditional supplemental logging of all unique key(s) and foreign key(s) of all current and future tables in the given schema. Unconditional logging forces the primary key values to the log whether or not the key was changed in the current operation. Conditional logging logs all of the column values of a foreign or unique key if at least one of them was changed in the current operation. The default is optional to support nonintegrated Replicat but is required to support integrated Replicat because primary key, unique keys, and foreign keys must all be available to the inbound server to compute dependencies. For more information about integrated Replicat, see Deciding Which Replicat Method to Use.
- `ALLCOLS` can be used to enable the unconditional supplemental logging of all of the columns of a table and applies to all current and future tables in the given schema. Use to support integrated Replicat when the source and target tables have different scheduling columns. (*Scheduling columns* are the primary key, the unique key, and the foreign key.)
- `NOSCHEDULINGCOLS` logs only the values of the primary key and all valid unique indexes for existing tables in the schema and new tables added later. This is the minimal required level of schema-level logging and is valid only for Replicat in nonintegrated mode.

In the following example, the command enables default supplemental logging for the `hr` schema.

```
ADD SCHEMATRANDATA pdbeast.hr ALLCOLS
```

In the following example, the command enables the supplemental logging only for the primary key and valid unique indexes for the `HR` schema.

```
ADD SCHEMATRANDATA pdbeast.hr NOSCHEDULINGCOLS
```

Enabling Table-level Supplemental Logging

Enable table-level supplemental logging on the source system in the following cases:

- To enable the required level of logging when not using schema-level logging (see Enabling Schema-level Supplemental Logging). Either schema-level or table-level logging must be used. By default, table-level logging automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of a table. Options enable you to alter the logging as needed.
- To prevent the logging of the primary key for any given table.
- To log non-key column values at the table level to support specific Oracle GoldenGate features, such as filtering and conflict detection and resolution logic.
- If the key columns change on a table that only has table-level supplemental logging, you must perform `ADD TRANDATA` on the table prior to allowing any DML activity on the table.

Perform the following steps on the source system to enable table-level supplemental logging or use the optional features of the command.

1. Run the command line on the source system.
2. Issue the `DBLOGIN` command using the alias of a user in the credential store who has privilege to enable table-level supplemental logging.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   See `USERIDALIAS` in *Parameters and Functions Reference for Oracle GoldenGate*for more information about `DBLOGIN` and additional options.
3. Issue the `ADD TRANDATA` command.

   ```
   ADD TRANDATA [PDB.]schema.table [, COLS (columns)] [, NOKEY] [, ALLCOLS |
   NOSCHEDULINGCOLS]
   ```

   Where:

   - `PDB` is the name of the root container or pluggable database if the table is in a multitenant container database.
   - `schema` is the source schema that contains the table.
   - `table` is the name of the table. See Specifying Object Names in Oracle GoldenGate Input for instructions for specifying object names.
   - `ADD TRANDATA` without other options automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of the table. Unconditional logging forces the primary key values to the log whether or not the key was changed in the current operation. Conditional logging logs all of the column values of a foreign or unique key if at least one of them was changed in the current operation. The default is optional to support nonintegrated Replicat (see also `NOSCHEDULINGCOLS`) but is required to support integrated Replicat because primary key, unique keys, and foreign keys must all be available to the inbound server to compute dependencies. For more information about integrated Replicat, see Deciding Which Replicat Method to Use.
   - `ALLCOLS` enables the unconditional supplemental logging of all of the columns of the table. Use to support integrated Replicat when the source and target tables have different scheduling columns. (*Scheduling columns* are the primary key, the unique key, and the foreign key.)
   - `NOSCHEDULINGCOLS` is valid for Replicat in nonintegrated mode only. It issues an `ALTER TABLE` command with an `ADD SUPPLEMENTAL LOG DATA ALWAYS` clause that is appropriate for the type of unique constraint that is defined for the table, or all columns in the absence of a unique constraint. This command satisfies the basic table-level logging requirements of Oracle GoldenGate when schema-level logging will not be used. See Ensuring Row Uniqueness in Source and Target Tables for how Oracle GoldenGate selects a key or index.
   - `COLS columns` logs non-key columns that are required for a `KEYCOLS` clause or for filtering and manipulation. The parentheses are required. These columns will be logged in addition to the primary key unless the `NOKEY` option is also present.
   - `NOKEY` prevents the logging of the primary key or unique key. Requires a `KEYCOLS` clause in the `TABLE` and `MAP` parameters and a `COLS` clause in the `ADD TRANDATA` command to log the alternate `KEYCOLS` columns.

4. If using `ADD TRANDATA` with the `COLS` option, create a unique index for those columns on the target to optimize row retrieval. If you are logging those columns as a substitute key for a `KEYCOLS` clause, make a note to add the `KEYCOLS` clause to the `TABLE` and `MAP` statements when you configure the Oracle GoldenGate processes.

Enabling Oracle GoldenGate in the Database

The database services required to support Oracle GoldenGate capture and apply must be enabled explicitly for all Oracle database versions. This is required for Extract and all Replicat modes.

To enable Oracle GoldenGate, set the following database initialization parameter. All instances in Oracle RAC must have the same setting.

```
ENABLE_GOLDENGATE_REPLICATION=true
```

This parameter alters the `DBA_FEATURE_USAGE_STATISTICS` view. For more information about this parameter, see Initialization Parameters.

Setting Flashback Query

To process certain update records, Extract fetches additional row data from the source database.

Oracle GoldenGate fetches data for the following:

- User-defined types
- Nested tables
- XMLType objects

By default, Oracle GoldenGate uses Flashback Query to fetch the values from the undo (rollback) tablespaces. That way, Oracle GoldenGate can reconstruct a read-consistent row image as of a specific time or SCN to match the redo record.

For best fetch results, configure the source database as follows:

1. Set a sufficient amount of redo retention by setting the Oracle initialization parameters `UNDO_MANAGEMENT` and `UNDO_RETENTION` as follows (in seconds).

   ```
   UNDO_MANAGEMENT=AUTO

   UNDO_RETENTION=86400

   UNDO_RETENTION can be adjusted upward in high-volume environments.
   ```

2. Calculate the space that is required in the undo tablespace by using the following formula.

   ```
   undo_space = UNDO_RETENTION * UPS + overhead
   ```

   Where:

   - *`undo_space`* is the number of undo blocks.
   - `UNDO_RETENTION` is the value of the `UNDO_RETENTION` parameter (in seconds).
   - *`UPS`* is the number of undo blocks for each second.
   - *`overhead`* is the minimal overhead for metadata (transaction tables, etc.).

   Use the system view `V$UNDOSTAT` to estimate *`UPS`* and *`overhead`*.

3. For tables that contain `LOBs`, do one of the following:
   - Set the `LOB` storage clause to `RETENTION`. This is the default for tables that are created when `UNDO_MANAGEMENT` is set to `AUTO`.
   - If using `PCTVERSION` instead of `RETENTION`, set `PCTVERSION` to an initial value of 25. You can adjust it based on the fetch statistics that are reported with the `STATS EXTRACT` command. If the value of the `STAT_OPER_ROWFETCH CURRENTBYROWID` or `STAT_OPER_ROWFETCH_CURRENTBYKEY` field in these statistics is high, increase `PCTVERSION` in increments of 10 until the statistics show low values.
4. Grant either of the following privileges to the Oracle GoldenGate Extract user:

```
GRANT FLASHBACK ANY TABLE TO db_user

GRANT FLASHBACK ON schema.table TO db_user
```

Oracle GoldenGate provides the following parameters to manage fetching.

Parameter or Command	Description
`STATS EXTRACT` command with `REPORTFETCH` option	Shows Extract fetch statistics on demand.
`STATOPTIONS` parameter with `REPORTFETCH` option	Sets the `STATS EXTRACT` command so that it always shows fetch statistics.
`MAXFETCHSTATEMENTS` parameter	Controls the number of open cursors for prepared queries that Extract maintains in the source database, and also for `SQLEXEC` operations.
`MAXFETCHSTATEMENTS` parameter	Controls the default fetch behavior of Extract: whether Extract performs a flashback query or fetches the current image from the table.
`FETCHOPTIONS` parameter with the `USELATESTVERSION` or `NOUSELATESTVERSION` option	Handles the failure of an Extract flashback query, such as if the undo retention expired or the structure of a table changed. Extract can fetch the current image from the table or ignore the failure.
`REPFETCHEDCOLOPTIONS` parameter	Controls the response by Replicat when it processes trail records that include fetched data or column-missing conditions.

Managing Server Resources

Extract interacts with an underlying logmining server in the source database and Replicat interacts with an inbound server in the target database. This section provides guidelines for managing the shared memory consumed by the these servers.

The shared memory that is used by the servers comes from the Streams pool portion of the System Global Area (SGA) in the database. Therefore, you must set the database initialization parameter `STREAMS_POOL_SIZE` high enough to keep enough memory available for the number of Extract and Replicat processes that you expect to run in integrated mode. Note that Streams pool is also used by other components of the database (like Oracle Streams, Advanced Queuing, and Datapump export/import), so make certain to take them into account while sizing the Streams pool for Oracle GoldenGate.

By default, one Extract requests the logmining server to run with `MAX_SGA_SIZE` of 1GB. Thus, if you are running three Extracts in the same database instance, you need at least 3 GB of memory allocated to the Streams pool. As a best practice, keep 25 percent of the Streams pool available. For example, if there are 3 Extracts, set `STREAMS_POOL_SIZE` for the database to the following value:

```
3 GB * 1.25 = 3.75 GB
```

Ensuring Row Uniqueness in Source and Target Tables

Oracle GoldenGate requires a unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority, depending on the number and type of constraints that were logged (see Configuring Logging Properties).

1. Primary key if it does not contain any extended (32K) `VARCHAR2`/`NVARCHAR2` columns. Primary key without invisible columns.
2. Unique key: Unique key without invisible columns.

   In the case of a non-integrated Replicat, the selection of the unique key is as follows:

   - First unique key alphanumerically with no virtual columns, no UDTs, no function-based columns, no nullable columns, and no extended (32K) `VARCHAR2`/`NVARCHAR2` columns. To support a key that contains columns that are part of an invisible index, you must use the `ALLOWINVISIBLEINDEXKEYS` parameter in the Oracle GoldenGate `GLOBALS` file.
   - First unique key alphanumerically with no virtual columns, no UDTs, no extended (32K) `VARCHAR2`/`NVARCHAR2` columns, or no function-based columns, but can include nullable columns. To support a key that contains columns that are part of an invisible index, you must use the `ALLOWINVISIBLEINDEXKEYS` parameter in the Oracle GoldenGate `GLOBALS` file.
3. Not Nullable Unique keys: At least one column from one of the unique keys must be not nullable. This is because `NOALLOWNULLABLEKEYS` is the default.

   > **Note:**
   >
   > `ALLOWNULLABLEKEYS` is not valid for integrated Replicat.

4. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding virtual columns, UDTs, function-based columns, extended (32K) `VARCHAR2`/`NVARCHAR2` columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration by an Oracle GoldenGate user.

   Unless otherwise excluded due to the preceding restrictions, invisible columns are allowed in the pseudo key.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

If a table does not have an appropriate key, or if you prefer the existing key(s) not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Establishing Oracle GoldenGate Credentials

Learn how to create database users for the processes that interacts with the database, assign the correct privileges, and secure the credentials from unauthorized use.

Assigning Credentials to Oracle GoldenGate

The Oracle GoldenGate processes require one or more database credentials with the correct database privileges for the database version, database configuration, and Oracle GoldenGate features that you are using.

Create users for the source and target database instances, each one dedicated to Oracle GoldenGate. The assigned user can be the same user for all the Oracle GoldenGate processes that must connect to a source or target Oracle Database.

See `ALTER CREDENTIALSTORE` command usage to manage credentials in a credential store for Oracle GoldenGate users.

Oracle GoldenGate Users (Database)

A user is required in the source database for the Manager process if you are using Oracle GoldenGate DDL support. This user performs maintenance on the Oracle GoldenGate database objects that support DDL capture.

A user is required in either the source or target database for the `DEFGEN` utility. The location depends on where the data definition file is being generated. This user performs local metadata queries to build a data-definitions file that supplies the metadata to remote Oracle GoldenGate instances.

Additional users or privileges may be required to use the following features, if Extract will run in classic capture mode:

- RMAN log retention, see #unique_270.
- TDE support, see #unique_271.
- ASM, see #unique_272.

Grant User Privileges for Oracle Database for Oracle GoldenGate Classic Architecture

The user privileges that are required for connecting to Oracle database from Oracle GoldenGate depend on the type of user.

The user privileges that are required for connecting to Oracle database from Oracle GoldenGate depend on the type of user.

Privileges should be granted depending on the actions that the user needs to perform as the GoldenGate Administrator User on the source and target databases. For example, to grant DML operation privileges to insert, update, and delete transactions to a user, use the `GRANT ANY INSERT`/`UPDATE`/`DELETE` privileges and to further allow users to work with tables and indexes as part of DML operations, use the `GRANT CREATE`/`DROP`/`ALTER ANY TABLE`/`INDEX` privileges.

If the GoldenGate Administrator user has the DBA role, additional object privileges are not needed. However, there might be security constraints granting the DBA role to the GoldenGate Administration user. The DBA role is not necessarily required for Oracle GoldenGate.

If there are many objects being replicated, you might consider using the ANY privilege for DML and DDL operations. This simplifies the provision of privileges to the GoldenGate Administrator users, as you only need to grant a few privileges depending on the database operations.

The following table describes some of the essential privileges for GoldenGate Administrator user for Oracle database. For explanation purposes, the table uses `c##ggadmin` as an example of a common user for a multitenant container database and `ggadmin` as the pluggable database (PDB) user. `PDBEAST` and `PDBWEST` are used as examples of PDB names.

The following table describes the essential privileges for GoldenGate Administrator user for using Oracle GoldenGate with on source and target Oracle databases:

Privilege	**Extract**	**Replicat All Modes**	**Purpose**
`RESOURCE`	Yes	Yes	Required to create objects In Oracle Database 12cR1 and later, instead of `RESOURCE`, grant the following privilege: `ALTER USER user QUOTA {size | UNLIMITED} ON tablespace;`
`CONNECT`	Yes	Yes	Common user `SYSTEM` connects to the root container. This privilege is essential when the DBA role is not assigned to the user.
`CREATE SESSION`	Yes	Yes	Required to connect to the database.
`CREATE VIEW`	Yes	Yes	Required to add the heartbeat table view. If you want to be specific to each object, you can also provide the privileges for each object individually. You may consider creating a specific database role to maintain such privileges.
`ALTER SYSTEM`	Yes	Yes	Perform administrative changes, such as enabling logging.
`ALTER USER`	Yes	Yes	Required for multitenant architecture and *`GGADMIN`* should be a valid Oracle GoldenGate administrator schema.

Privilege	Extract	Replicat All Modes	Purpose
`EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE ('REPUSER', CONTAINER=>'PDBEAST');`	Yes	Yes	• Required for Oracle Autonomous Database Extract and Replicat. Extracts in the root container (`CDB$ROOT)`) might require a value of `ALL` or a specific PDB (example: `pdbeast`). • Grant privileges for Extract and Replicat users. • Grant privilges to capture from Virtual Private Database • Grants privilges to capture redacted data
`INSERT`, `UPDATE`, `DELETE` on target tables	NA	Yes	Apply replicated DML to target objects. See Details of Support for Objects and Operations in Oracle DML

`GRANT INSERT ANY TO...`
`GRANT UPDATE ANY TO...`
`GRANT DELETE ANY TO...`
	NA	Yes	Grant these privileges to the Replicat user, instead of granting `INSERT`, `UPDATE`, `DELETE` to every table, if replicating every table.

DDL privileges on target objects (if using DDL support)	NA	Yes	Issue replicated DDL on target objects. See Details of Support for Objects and Operations in Oracle DDL.
`GRANT [CREATE|ALTER|DROP] ANY [TABLE|INDEX|VIEW|PROCEDURE] to GGADMIN;`	Yes	Yes	Grants privileges for DDL Replication for tables.
`CREATE ANY TABLE`	Yes	Yes	Grants privileges for creating table in any schema. To allow creating tables only in a specific schema, use the `CREATE TABLE` privilege.
`CREATE ANY VIEW`	Yes	Yes	Grants privilges to create view in any database schema. To allow creating views in a specific schema, use the `CREATE VIEW` privilege.
`SELECT ANY DICTIONARY`	Yes	Yes	Allow all privileges to work properly on dictionary tables.

Example: Permissions granted for the Oracle database common user

Privilges granted for the Oracle database common user, which is `c##ggadmin` in the following example:

```
CREATE USER c##ggadmin IDENTIFIED BY passw0rd CONTAINER=all DEFAULT
TABLESPACE GG_DATA TEMPORARY TABLESPACE temp;
GRANT RESOURCE to c##ggadmin;
GRANT CREATE SESSION to c##ggadmin;
GRANT CREATE VIEW to c##ggadmin;
GRANT CREATE TABLE to c##ggadmin;
GRANT CONNECT to c##ggadmin CONTAINER=all;
GRANT DV_GOLDENGATE_ADMIN; --- for data vault user
GRANT DV_GOLDENGATE_REDO_ACCESS; --- for data vault user
GRANT ALTER SYSTEM to c##ggadmin;
GRANT ALTER USER to c##ggadmin;
ALTER USER c##ggadmin SET CONTAINER_DATA=all CONTAINER=current;
ALTER USER c##ggadmin QUOTA unlimited ON GG_DATA;
GRANT SELECT ANY DICTIONARY to c##ggadmin;
GRANT SELECT ANY TRANSACTION to c##ggadmin;
EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE('c##ggadmin');
```

In this example, DBA privilege is not provided but the user will be able to access the `DBA_SYS_PRIVS` package, if required.

Privileges granted for PDB user `ggadmin` are provided in the following example:

Example: Grant privileges using the DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE package

This procedure grants the privileges needed by a user to be an Oracle GoldenGate administrator The following example grants explicit privileges for Extract on Oracle multitenant database:

```
BEGIN
DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE
(GRANTEE => 'c##ggadmin',  PRIVILEGE_TYPE => 'CAPTURE',
 GRANT_SELECT_PRIVILEGES => TRUE,  DO_GRANTS => TRUE,  CONTAINER => 'ALL'  );
END;
```

See `DBMS_GOLDENGATE_AUTH` in *Oracle Database PL/SQL Packages and Types Reference* for more information.

Oracle Database Privileges

The following privileges apply to Oracle database.

Privilege	Extract	Replicat All Modes	Purpose
`CREATE SESSION`	No	No	Connect to the database
`CREATE VIEW`	No	No	Required to add the heartbeat table view.

Privilege	Extract	Replicat All Modes	Purpose
`RESOURCE`	No	No	Create objects. In Oracle Database 12cR1 and later, instead of `RESOURCE`, grant the following privilege: `ALTER USER user QUOTA {size | UNLIMITED} ON tablespace;`
`ALTER SYSTEM`	No	No	Perform administrative changes, such as enabling logging.
`ALTER USER`	No	No	Required for multitenant architecture and *`GGADMIN`* should be a valid Oracle GoldenGate administrator schema.
`EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE ('REPUSER', CONTAINER=>'PDB1');`	Yes	Yes	This is required for Autonomous Databases (ATP and ADW) Extract and Replicat. Extracts in the Root container (`CDB$ROOT)`) might require a value of `ALL` or a specific PDB.
Privileges granted through `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE`	No	No	(Extract) Grants privileges for Extract, including the logmining server. (Replicat) Grants privileges for both non-integrated and integrated Replicat, including the database inbound server.
Any or all of optional privileges of `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE`	No	No	• Capture from Virtual Private Database • Capture redacted data

Grant the following privileges connected as `SYS` user to Extract and Replicat users:
`EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE('ggadmin user','*',GRANT_OPTIONAL_PRIVILEGES=>'*');`
`GRANT DV_GOLDENGATE_ADMIN, DV_GOLDENGATE_REDO_ACCESS to 'ggadmin user';`
	No	No	Capture from Data Vault

Privilege	Extract	Replicat All Modes	Purpose

Grant Replicat the privileges in `DBMS_MACADM.ADD_AUTH_TO_REALM` if applying to a realm.
`Connect as Database Vault owner and execute the following sctipts,`
`BEGIN`
`DVSYS.DBMS_MACADM.ADD_AUTH_TO_REALM( REALM_NAME => 'Oracle Default Component Protection Realm',GRANTEE => 'GGADMIN USER',AUTH_OPTIONS => 1) ;`
`END ;`
`/`
`EXECUTE DBMS_MACADM.AUTHORIZE_DDL('SYS', 'SYSTEM');`
	No	No	Capture from Data Vault

If DDL replication is performed, grant the following as Database Vault owner: `EXECUTE DBMS_MACADM.AUTHORIZE_DDL (‘GGADMIN USER', ‘SCHEMA FOR DDL’);`	No	No	Capture from Data Vault
`INSERT`, `UPDATE`, `DELETE` on target tables	NA	No	Apply replicated DML to target objects
`GRANT INSERT ANY TO...`, `GRANT UPDATE ANY TO...` and `GRANT DELETE ANY TO...`	NA	No	Use these privileges for the Replicat user, instead of granting `INSERT`, `UPDATE`, `DELETE` to every table, if replicating every table.
DDL privileges on target objects (if using DDL support)	NA	No	Issue replicated DDL on target objects
`LOCK ANY TABLE`	NA	No	Lock target tables. Only required for initial load using direct bulk load to SQL*Loader.
`SELECT ANY DICTIONARY`	No	No	Allow all privileges to work properly on dictionary tables.
`SELECT ANY TRANSACTION`	No	NA	Use a newer Oracle ASM API.

Here's an example of the list of permissions granted for the Oracle database root container:

```
DROP USER c##ggadmin CASCADE;
CREATE USER c##ggadmin IDENTIFIED BY passw0rd CONTAINER=all DEFAULT
TABLESPACE GG_DATA TEMPORARY TABLESPACE temp;
ALTER USER c##ggadmin SET CONTAINER_DATA=all CONTAINER=current;
GRANT CREATE SESSION to c##ggadmin;
GRANT CREATE VIEW to c##ggadmin;
GRANT CONNECT to c##ggadmin CONTAINER=all;
GRANT RESOURCE to c##ggadmin;
GRANT ALTER SYSTEM to c##ggadmin ;
GRANT SELECT ANY DICTIONARY to c##ggadmin ;
EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE('c##ggadmin');
ALTER USER c##ggadmin QUOTA unlimited ON GG_DATA;

SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE='c##ggadmin' ORDER BY 2;
```

In this example, DBA privilege is not provided but the user will be able to access the `DBA_SYS_PRIVS` package.

About the DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE Package

Most of the privileges that are needed for Extract and Replicat to operate are granted through the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE` package.

The first example is the default, which grants to both Extract and Replicat. The second shows how to explicitly grant to either Extract or Replicat (in this case, Extract).

```
GRANT_ADMIN_PRIVILEGE ('ggadmin')
GRANT_ADMIN_PRIVILEGE ('ggadmin','exte');
```

The following example shows Extract on Oracle 12*c* multitenant database:

```
BEGIN
DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE
(GRANTEE => 'c##ggadmin',  PRIVILEGE_TYPE => 'CAPTURE',
 GRANT_SELECT_PRIVILEGES => TRUE,  DO_GRANTS => TRUE,  CONTAINER => 'ALL'  );
END;
```

Optional Grants for dbms_goldengate_auth.grant_admin_privilege

This procedure grants the privileges needed by a user to be a Oracle GoldenGate administrator. See `DBMS_GOLDENGATE_AUTH` in *Oracle Database PL/SQL Packages and Types Reference* for more information.

Securing the Oracle GoldenGate Credentials

To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, an Oracle GoldenGate database user.

Oracle GoldenGate provides different options for securing the log-in credentials assigned to Oracle GoldenGate processes. The recommended option is to use a credential store. You can create one credential store and store it in a shared location where all installations of Oracle GoldenGate can access it, or you can create a separate one on each system where Oracle GoldenGate is installed.

The credential store stores the user name and password for each of the assigned Oracle GoldenGate users. A user ID is associated with one or more aliases, and it is the alias that is supplied in commands and parameter files, not the actual user name or password. The credential file can be partitioned into domains, allowing a standard set of aliases to be used for the processes, while allowing the administrator on each system to manage credentials locally.

See Creating and Populating the Credential Store in *Oracle GoldenGate Security Guide* for more information about creating a credential store and adding user credentials.

Additional Oracle GoldenGate Configuration for Your Database

This chapter contains additional configuration considerations that may apply to your database environment.

Installing Support for Oracle Sequences

To support Oracle sequences, you must install some database procedures.

To Install Oracle Sequence Objects

1. In SQL*Plus, connect to the source and target Oracle systems as `SYSDBA`.
2. If you already assigned a database user to support the Oracle GoldenGate DDL replication feature, you can skip this step. Otherwise, in SQL*Plus on both systems create a database user that can also be the DDL user.

   ```
   CREATE USER DDLuser IDENTIFIED BY password;
   GRANT CONNECT, RESOURCE, DBA TO DDLuser;
   ```

3. From the Oracle GoldenGate installation directory on each system, run GGSCI.
4. In GGSCI, issue the following command on each system.

   ```
   EDIT PARAMS ./GLOBALS
   ```

5. In each `GLOBALS` file, enter the `GGSCHEMA` parameter and specify the schema of the DDL user that you created earlier in this procedure.

   ```
   GGSCHEMA schema
   ```

6. Save and close the files.
7. In SQL*Plus on the source system, issue the following statement in SQL*Plus.

   ```
   ALTER TABLE sys.seq$ ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```

To capture the sequence from a multitenant database

1. Create an Oracle GoldenGate user in each PDB that you need to capture sequences from.
2. Add the user to the GLOBALS parameter file. It is easier if you use the same user for each PDB, if you don't then you need to change the GLOBALS file each time you do step 3.
3. Log into Admin Client or GGSCI.
4. Connect to the root container on the source using `DBLOGIN`.
5. Issue the `FLUSH SEQUENCE` command for each PDB.

If you don't want to keep these database accounts, you can drop the user or deactivate the account.

Here is an example of the entire process:

```
Environment information
            OGG 19.1 Oracle 12c to Oracle 12c Replication, Integrated
            Extract, Parallel Replicat
```

```
            Source: CDB GOLD, PDB CERTMISSN
            Target: CDB PLAT, PDB CERTDSQ
            Source Oracle GoldenGate Configuration
            Container User: C##GGADMIN
            PDB User for Sequences: GGATE
```

When prompted, enter `GGATE`

```
GLOBALS
            GGSCHEMA GGATE
         Flush Sequence
            GGSCI> DBLOGIN USERIDALIAS GGADMIN DOMAIN GOLD_QC_CDB$ROOT
            GGSCI> FLUSH SEQUENCE CERTMISSN.SRCSCHEMA1.
Target OGG Configuration
         PDB User: GGATE
         Run @sequence
                sqlplus / as sysdba
                SQL> alter session set container=CERTDSQ;
                SQL> @sequence
```

When prompted enter `GGATE`.

Replicating sequences is required for High Availability (HA) and DR scenarios:

- For migrations, you need rebuild the sequences on the target during the switchover, or increase them to a higher value just prior to the switchover.
- Make sure you place the sequences into their own Replicat.

Handling Special Data Types

It addresses special configuration requirements for different Oracle data types for Extract.

Multibyte Character Types

Multi-byte characters are supported as part of a supported character set. If the semantics setting of an Oracle source database is `BYTE` and the setting of an Oracle target is `CHAR`, use the Replicat parameter `SOURCEDEFS` in your configuration, and place a definitions file that is generated by the `DEFGEN` utility on the target. These steps are required to support the difference in semantics, whether or not the source and target data definitions are identical. Replicat refers to the definitions file to determine the upper size limit for fixed-size character columns.

Oracle Spatial Objects

To replicate tables that contain one or more columns of `SDO_GEORASTER` object type from an Oracle source to an Oracle target, follow these instructions to configure Oracle GoldenGate to process them correctly.

1. Create a `TABLE` statement and a `MAP` statement for the georaster tables and also for the related raster data tables.
2. If the `METADATA` attribute of the `SDO_GEORASTER` data type in any of the values exceeds 1 MB, use the `DBOPTIONS` parameter with the `XMLBUFSIZE` option to increase the size of the memory buffer that stores the embedded `SYS.XMLTYPE` attribute of the `SDO_GEORASTER` data type. If the buffer is too small, Extract abends. See `XMLBUFSIZE` in *Parameters and Functions Reference for Oracle GoldenGate*.

3. To ensure the integrity of the target georaster tables and the spatial data, keep the trigger enabled on both source and target. Use the `REPERROR` option of the `MAP` parameter to handle the "ORA-01403 No data found" error that occurs as a result of keeping the trigger enabled on the target. It occurs when a row in the source georaster table is deleted, and the trigger cascades the delete to the raster data table. Both deletes are replicated. The replicated parent delete triggers the cascaded (child) delete on the target. When the replicated child delete arrives, it is redundant and generates the error. To use `REPERROR`, do the following:

    - Use a `REPERROR` statement in each `MAP` statement that contains a raster data table.
    - Use Oracle error 1403 as the SQL error.
    - Use any of the response options as the error handling.

A sufficient way to handle the errors on raster tables caused by active triggers on target georaster tables is to use `REPERROR` with `DISCARD` to discard the cascaded delete that triggers them. The trigger on the target georaster table performs the delete to the raster data table, so the replicated one is not needed.

```
MAP geo.st_rdt, TARGET geo.st_rdt, REPERROR (-1403, DISCARD) ;
```

If you need to keep an audit trail of the error handling, use `REPERROR` with `EXCEPTION` to invoke exceptions handling. For this, you create an exceptions table and map the source raster data table twice:

- once to the actual target raster data table (with `REPERROR` handling the 1403 errors).
- again to the exceptions table, which captures the 1403 error and other relevant information by means of a `COLMAP` clause.

For more information about using an exceptions table, see *Administering Oracle GoldenGate for Windows and UNIX*.

For more information about `REPERROR` options, see *Parameters and Functions Reference for Oracle GoldenGate*.

TIMESTAMP

To replicate timestamp data, Oracle Database normalizes `TIMESTAMP WITH LOCAL TIME ZONE` data to the local time zone of the database that receives it, the target database in case of Oracle GoldenGate. To preserve the original time stamp of the data that it applies, Replicat sets its session to the time zone of the source database. You can override this default and supply a different time zone by using the `SOURCETIMEZONE` parameter in the Replicat parameter file. To force Replicat to set its session to the target time zone, use the `PRESERVETARGETTIMEZONE` parameter.

To prevent Oracle GoldenGate from abending on `TIMESTAMP WITH TIME ZONE` as `TZR`, use the Extract parameter `TRANLOGOPTIONS` with `INCLUDEREGIONIDWITHOFFSET` to replicate `TIMESTAMP WITH TIMEZONE` as `TZR` from an Oracle source that is at least version 10g to an earlier Oracle target, or from an Oracle source to a non-Oracle target. This option allows replicating to Oracle versions that do not support `TIMESTAMP WITH TIME ZONE` as TZR and to database systems that only support time zone as a UTC offset.

You can also use the `SOURCETIMEZONE` parameter to specify the source time zone for data that is captured by an Extract that is earlier than version 12.1.2. Those versions do not write the source time zone to the trail.

Large Objects (LOB)

The following are some configuration guidelines for Extract LOBs.

1. Store large objects out of row if possible.
2. Extract captures LOBs from the redo log. For `UPDATE` operations on a LOB document, only the changed portion of the LOB is logged. To force whole LOB documents to be written to the trail when only the changed portion is logged, use the `TRANLOGOPTIONS` parameter with the `FETCHPARTIALLOB` option in the Extract parameter file. When Extract receives partial LOB content from the logmining server, it fetches the full LOB image instead of processing the partial LOB. Use this option when replicating to a non-Oracle target or in other conditions where the full LOB image is required.

XML

The following are tools for working with XML within Oracle GoldenGate constraints.

- Although Extract does not support the capture of changes made to an XML schema, you may be able to evolve the schemas and then resume replication of them without the need for a resynchronization, see Supporting Changes to XML Schemas.
- Extract captures XML from the redo log. For `UPDATE` operations on an XML document, only the changed portion of the XML is logged if it is stored as `OBJECT RELATIONAL` or `BINARY`. To force whole XML documents to be written to the trail when only the changed portion is logged, use the `TRANLOGOPTIONS` parameter with the `FETCHPARTIALXML` option in the Extract parameter file. When Extract receives partial XML content from the logmining server, it fetches the full XML document instead of processing the partial XML. Use this option when replicating to a non-Oracle target or in other conditions where the full XML image is required.

User Defined Types

If Oracle Database is compatible with releases greater than or equal to 12.0.0.0.0, then Extract captures data from redo (no fetch), see Setting Flashback Query.

If replicating source data that contains user-defined types with the `NCHAR`, `NVARCHAR2`, or `NCLOB` attribute to an Oracle target, use the `HAVEUDTWITHNCHAR` parameter in the Replicat parameter file. When this type of data is encountered in the trail, `HAVEUDTWITHNCHAR` causes Replicat to connect to the Oracle target in `AL32UTF8`, which is required when a user-defined data type contains one of those attributes. `HAVEUDTWITHNCHAR` is required even if `NLS_LANG` is set to `AL32UTF8` on the target. By default Replicat ignores `NLS_LANG` and connects to an Oracle Database in the native character set of the database. Replicat uses the `OCIString` object of the Oracle Call Interface, which does not support `NCHAR`, `NVARCHAR2`, or `NCLOB` attributes, so Replicat must bind them as `CHAR`. Connecting to the target in `AL32UTF8` prevents data loss in this situation. `HAVEUDTWITHNCHAR` must appear before the `USERID` or `USERIDALIAS` parameter in the parameter file.

Handling Other Database Properties

This topic describes the database properties that may affect Oracle GoldenGate and the parameters that you can use to resolve or work around the condition.

The following table lists the database properties and the associated concern/resolution.

Database Property	Concern/Resolution
Table with interval partitioning	To support tables with interval partitioning, make certain that the `WILDCARDRESOLVE` parameter remains at its default of `DYNAMIC`.
Table with virtual columns	Virtual columns are not logged, and Oracle does not permit DML on virtual columns. You can, however, capture this data and map it to a target column that is not a virtual column by doing the following: Include the table in the Extract `TABLE` statement and use the `FETCHCOLS` option of `TABLE` to fetch the value from the virtual column in the database. In the Replicat `MAP` statement, map the source virtual column to the non-virtual target column.
Table with inherently updateable view	To replicate to an inherently updateable view, define a key on the unique columns in the updateable view by using a `KEYCOLS` clause in the same `MAP` statement in which the associated source and target tables are mapped.
Redo logs or archives in different locations	The `TRANLOGOPTIONS` parameter contains options to handle environments where the redo logs or archives are stored in a different location than the database default or on a different platform from that on which Extract is running. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.
`TRUNCATE` operations	To replicate `TRUNCATE` operations, choose one of two options: • Standalone `TRUNCATE` support by means of the `GETTRUNCATES` parameter replicates `TRUNCATE TABLE`, but no other `TRUNCATE` options. Use only if not using Oracle GoldenGate DDL support. • The full DDL support replicates `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL.
Sequences	To replicate DDL for sequences (`CREATE`, `ALTER`, `DROP`, `RENAME`), use Oracle GoldenGate DDL support. To replicate just sequence values, use the `SEQUENCE` parameter in the Extract parameter file. This does *not* require the Oracle GoldenGate DDL support environment. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Controlling the Checkpoint Retention

The `CHECKPOINTRETENTIONTIME` option of the `TRANLOGOPTIONS` parameter controls the number of days that Extract retains checkpoints before purging them automatically.

Partial days can be specified using decimal values. For example, 8.25 specifies 8 days and 6 hours. The default is seven days.

Excluding Replicat Transactions

In a bidirectional configuration, Replicat must be configured to mark its transactions, and Extract must be configured to exclude Replicat transactions so that they do not propagate back to their source. There are two methods to accomplish this as follows:

Method 1

Valid only for Oracle to Oracle implementations.

Replicat can be in either integrated or nonintegrated mode. Use the following parameters:

- Use `DBOPTIONS` with the `SETTAG` option in the Replicat parameter file. The inbound server tags the transactions of that Replicat with the specified value, which identifies those transactions in the redo stream. The default value for `SETTAG` is 00.

- Use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG` option in an Extract parameter file. The logmining server associated with that Extract excludes redo that is tagged with the `SETTAG` value. Multiple `EXCLUDETAG` statements can be used to exclude different tag values, if desired.

  For Oracle to Oracle, this is the recommended method.

Method 2

Valid for any implementation; Oracle or non-Oracle database configurations.

Alternatively, you could use the Extract `TRANLOGOPTIONS` parameter with the `EXCLUDEUSER` or `EXCLUDEUSERID` option to ignore the Replicat DDL and DML transactions based on its user name or ID. Multiple `EXCLUDEUSER` statements can be used. The specified user is subject to the rules of the `GETREPLICATES` or `IGNOREREPLICATES` parameter.

For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Advanced Configuration Options for Oracle GoldenGate

You may need to configure Oracle GoldenGate with advanced options to suit your business needs.

See the following:

- For additional configuration guidelines to achieve specific replication topologies, see *Administering Oracle GoldenGate*. This guide includes instructions for the following configurations:
  - Using Oracle GoldenGate for live reporting
  - Using Oracle GoldenGate for real-time data distribution
  - Configuring Oracle GoldenGate for real-time data warehousing
  - Using Oracle GoldenGate to maintain a live standby database
  - Using Oracle GoldenGate for active-active high availability

  That guide also contains information about:
  - Oracle GoldenGate architecture
  - Oracle GoldenGate commands
  - Oracle GoldenGate initial load methods
  - Configuring security
  - Using customization features
  - Configuring data filtering and manipulation
- If either the source or target database is non-Oracle, follow the installation and configuration instructions in the Oracle GoldenGate installation and setup guide for that database, and then refer to the Oracle GoldenGate administration and reference documentation for further information.

Supported Oracle Data Types, Objects, and Operations for DDL and DML

This chapter contains support information for Oracle GoldenGate on Oracle Database.

Details of Support for Oracle Data Types and Objects

This topic describes data types, objects and operations that are supported by Oracle GoldenGate.

Within the database, you can use the Dictionary view `DBA_GOLDENGATE_SUPPORT_MODE` to get information about supported objects. There are different types for replication support:

- Support by Capturing from Redo
- Procedural Replication Support

Most data types are supported (`SUPPORT_MODE=FULL`), which implies that Oracle GoldenGate captures the changes out of the redo. In some unique cases, the information cannot be captured, but the information can be fetched with a connection to the database (`SUPPORT_MODE=ID KEY`). Tables supported with `ID KEY` require a connection to the source database or an ADG Standby database for fetching to support those tables. If using downstream Integrated Extract, with `NOUSERID` a customer must specify a `FETCHUSERID` or `FETCHUSERIDALIAS` connection.

Other changes can be replicated with Procedural Replication (`SUPPORT_MODE=PLSQL`) that requires additional parameter setting of Extract. See About Procedural Replication for details. In the unlikely case that there is no native support, no support by fetching and no procedural replication support, there is no Oracle GoldenGate support.

To know more information about capture modes, see Deciding Which Capture Method to Use.

Besides the DBA_GOLDENGATE_SUPPORT_MODE at the source database you should check the DBA_GOLDENGATE_NOT_UNIQUE dictionary view at the target side. If there are tables without any uniqueness and unbounded data_types (BAD_COLUMN='Y'), the table records cannot be uniquely identified and cannot be used for logical replication.

Detailed support information for Oracle data types, objects, and operations starts with Details of Support for Objects and Operations in Oracle DML.

There might be a few cases where replication support exists, but there are limitations of processing such as in case of using `SQLEXEC`. The following table lists these limitations:

Support Datatypes	**No Support**

`NUMBER, BINARY FLOAT, BINARY DOUBLE`
`UROWID`
	Special cases of:
	• XML types
	• UDTs
	• Object tables
	• Collections or nested tables

`DATE` and `TIMESTAMP`	Tables with restricted uniqueness
`(N)CHAR`, `(N) VARCHAR2 LONG`, `RAW`, `LONG RAW` `(N)CLOB`, `CLOB`, `BLOB`, `SECUREFILE`, `BASICFILE` and `BFILE`	X
`XML` columns, `XMLType`	X
`UDT`s	X
`ANYDATA`	X
Hierarchy-enabled tables	X
`RET` Types	X

Support Datatypes	No Support
`DICOM`	X
`SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`	X
Identity columns	X
`SDO_RDF_TRIPLE_S`	X

> **Note:**
>
> `SECUREFILE` LOBs updated using `DBMS_LOG.FRAGMENT` or `SECUREFILE` LOBs that are set to `NOLOGGING` are fetched instead of read from the redo.

Supported Capture from Redo:

- `NUMBER`, `BINARY FLOAT`, `BINARY DOUBLE`, and (logical) `UROWID`
- `DATE` and `TIMESTAMP`
- `CHAR`, `VARCHAR2`, `LONG`, `NCHAR`, and `NVARCHAR2`
- `RAW`, `LONG RAW`, `CLOB`, `NCLOB`, `BLOB`, `SECUREFILE`, `BASICFILE`, and `BFILE` (LOB size limited to 4GB)
- `XML` columns stored as `CLOB`, `Binary` and Object-Relational (OR)
- `XMLType` columns and `XMLType` tables stored as `XML CLOB`, `XML` Object Relational, and `XML Binary`
- `UDTs` (user-defined or abstract data types) on `BYTE` semantics with source database compatibility 12.0.0.0.0 or higher
- `ANYDATA` data type with source database compatibility 12.0.0.0.0 or higher
- Hierarchy-enabled tables are managed by the Oracle XML database repository with source database compatibility 12.2.0.0.0 or higher and enabled procedural replication
- `REF` types with source database compatibility 12.2.0.0.0 or higher
- `DICOM` with source database compatibility 12.0.0.0.0 or higher
- `SDO _TOPO_GEOMETRY` or `SDO_GEORASTER` with source database compatibility 12.2.0.0.0 or higher and enabled procedural replication
- Identity columns with source database compatibility 18.1.0.0.0 or higher
- `SDO_RDF_TRIPLE_S` with source database compatibility 19.1.0.0.0 or higher

Supported (Fetch from database)

`SECUREFILE` LOBs

- Modified with `DBMS_LOB.FRAGMENT_*` procedures
- `NOLOGGING` LOBs
- Deduplicated LOBs with a source database release less than 12gR2

`UDTs` that contain following data types:

- `TIMESTAMP WITH TIMEZONE`, `TIMESTAMP WITH LOCAL TIMEZONE`, `TIMESTAMP WITH TIMEZONE` with region ID
- `INTERVAL YEAR TO MONTH`, `INTERVAL DAY TO SECOND`
- `BINARY FLOAT`, `BINARY DOUBLE`
- `BFILE`

Object tables contains the following attributes:

- Nested table
- `SDO_TOP_GEOMETRY`
- `SDO_GEORASTER`

Additional Considerations

- `NUMBER` can be up to the maximum size permitted by Oracle. The support of the range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- Non-logical `UROWID` columns will be identified by Extract. A warning message is generated in the report file. The column information is not part of the trail record. All other supported datatypes of the record are part of the trail record and are replicated.
- `TIMESTAMP WITH TIME ZONE` as `TZR` (region ID) for initial loads, `SQLEXEC` or operations where the column can only be fetched from the database. In those cases, the region ID is converted to a time offset by the source database when the column is selected. Replicat applies the timestamp as date and time data into the target database with a time offset value.
- `VARCHAR` expansion from 4K to 32K (extended or long `VARCHAR`)
  - 32K long columns cannot be used as row identifiers:
    - Columns as part of a key or unique index
    - Columns in a `KEYCOLS` clause of the `TABLE` or `MAP` parameter.
  - 32K long columns as resolution columns in a CDR (conflict resolution and detection)
  - If an extended `VARCHAR` column is part of unique index or constraint, then direct path inserts to this table may cause Replicat to abend with a warning. Verify that the extended `VARCHAR` caused the abend by checking `ALL_INDEXES` or `ALL_IND_COLUMNS` for a unique index or `ALL_CONS_COLUMNS` or `ALL_CONSTRAINTS` for a unique constraint. Once you determine that an extended `VARCHAR`, you can temporarily drop the index or disable the constraint:
    - Unique Index: `DROP INDEX` *`index_name`*`;`
    - Unique Constraint: `ALTER TABLE` *`table_name`* `MODIFY CONSTRAINT` *`constraint_name`* `DISABLE;`
- Oracle GoldenGate does not support the filtering, column mapping, or manipulation of objects larger than 4K.
- `BFILE` column are replicating the locator. The file on the server file system outside of the database and is not replicated.

- Multi-byte character data: The source and target databases must be logically identical in terms of schema definition for the tables and sequences being replicated. Transformation, filtering, and other manipulation cannot be used.
- The character sets between the two databases must be one of the following:
    - Identical on the source and on the target
    - Equivalent, which is not the same character set but containing the same set of characters
    - Target is a superset of the source

  Multi-byte data can be used in any semantics: bytes or characters.
- The structure of the UDTs and Abstract Data Types (ADTs) itself must be the same on both the source and target. `UDTs` can have different source and target schemas. `UDTs`, including values inside object columns or rows, cannot be used within filtering criteria in `TABLE` or `MAP` statements, or as input or output for the Oracle GoldenGate column-conversion functions, `SQLEXEC`, or other built-in data manipulation tools. Support is only provided for like-to-like Oracle source and targets.

  To fully support object tables created using the `CREATE TABLE as SELECT (CTAS)` statement, Integrated Extract must be configured to capture DML from the CTAS statement. Oracle object table can be mapped to a non-Oracle object table in a supported target database.
- `XML` column type cannot be used for filtering and manipulation. You can map the `XML` representation of an object to a character column by means of a `COLMAP` clause in a `TABLE` or `MAP` statement.

  Oracle recommends the `AL32UTF8` character set as the database character set when working with `XML` data. This ensures the correct conversion by Oracle GoldenGate from source to target. With DDL replication enabled, Oracle GoldenGate replicates the CTAS statement and allows it to select the data from the underlying target tables. OIDs are preserved if `TRANSLOGOPTIONS GETCTASDML` parameter is set. For `XMLType` tables, the row object IDs must match between source and target.

Non-Supported Oracle Data Types

Oracle GoldenGate does not support the following data types.

- Time offset values outside the range of +12:00 and -12:00..Oracle GoldenGate supports time offset values between +12:00 and -12:00.
- Tables that only contain a single column and that column one of the following:
    - UDT
    - LOB (CLOB, NCLOB, BLOB, BFILE)
    - XMLType column
    - VARCHAR2 (MAX) where the data is greater than 32KB
- Tables with LOB, UDT, XML, or XMLType column without one of the following:
    - Primary Key
    - Scalar columns with a unique constraint or unique index

  Table where the combination of all scalar columns do not guarantee uniqueness are unsupported.
- Tables with the following XML characteristics:

  - Tables with a primary key constraint made up of XML attributes
  - XMLType tables with a primary key based on an object identifier (PKOID).
  - XMLType tables, where the row object identifiers (OID) do not match between source and target
  - XMLType tables created by an empty CTAS statement.
  - XML schema-based XMLType tables and columns where changes are made to the XML schema (XML schemas must be registered on source and target databases with the `dbms_xml` package).
  - The maximum length for the entire `SET` value of an update to an XMLType larger than 32K, including the new content plus other operators and XQuery bind values.
  - SQL*Loader direct-path insert for XML-Binary and XML-OR.
- Tables with following UDT characteristics:
  - UDTs that contain CFILE or OPAQUE (except of XMLType)
  - UDTs with CHAR and VARCHAR attributes that contain binary or unprintable characters
  - UDTs using the RMTTASK parameter
- UDTs and nested tables with following condition:
  - Nested table UDTs with CHAR, NVARCHAR2 or NCLOB attributes.
  - Nested tables with CLOB, BLOB, extended (32k) VARCHAR2 or RAW attributes in UDTs.
  - Nested table columns/attributes that are part of any other UDT.
- When data in a nested table is updated, the row that contains the nested table must be updated at the same time. Otherwise there is no support.
- When VARRAYS and nested tables are fetched, the entire contents of the column are fetched each time, not just the changes. Otherwise there is no support.
- Object table contains the following attributes:
  - Nested table
  - SDO_TOPO_GEOMETRY
  - SDO_GEORASTER

Details of Support for Objects and Operations in Oracle DML

This section outlines the Oracle objects and operations that Oracle GoldenGate supports for the capture and replication of DML operations.

Multitenant Container Database

Oracle GoldenGate captures from, and delivers to, a **multitenant container database**. See Configuring Oracle GoldenGate in a Multitenant Container Database .

Application Container are not supported.

Tables, Views, and Materialized Views

Oracle GoldenGate supports the following DML operations made to regular tables, index-organized tables, clustered tables, and materialized views.

- `INSERT`
- `UPDATE`
- `DELETE`
- Associated transaction control operations

> **Tip:**
>
> You can use the `DBA_GOLDENGATE_SUPPORT_MODE` data dictionary view to display information about the level of Oracle GoldenGate capture process support for the tables in your database. The `PLSQL` value of `DBA_GOLDENGATE_SUPPORT_MODE` indicates that the table is supported natively, but requires procedural supplemental logging.
>
> Besides the `DBA_GOLDENGATE_SUPPORT_MODE` at the source database, you should check the `DBA_GOLDENGATE_NOT_UNIQUE` dictionary view at the target side. If there are tables without any uniqueness and unbounded data types (`BAD_COLUMN='Y'`), the table records cannot be uniquely identified and cannot be used for logical replication.
>
> For more information, see the `DBA_GOLDENGATE_SUPPORT_MODE`. If you need to display all tables that have no primary and no non-null unique indexes, you can use the `DBA_GOLDENGATE_NOT_UNIQUE`. For more information, see `DBA_GOLDENGATE_NOT_UNIQUE`.

Limitations of Support for Regular Tables

These limitations apply to Extract.

- Oracle GoldenGate supports tables that contain any number of rows.
- A row can be up to 4 MB in length. If Oracle GoldenGate is configured to include both the before and after image of a column in its processing scope, the 4 MB maximum length applies to the total length of the full before image plus the length of the after image. For example, if there are `UPDATE` operations on columns that are being used as a row identifier, the before and after images are processed and cannot exceed 4 MB in total. Before and after images are also required for columns that are not row identifiers but are used as comparison columns in conflict detection and resolution (CDR). Character columns that allow for more than 4 KB of data, such as a `CLOB`, only have the first 4 KB of data stored in-row and contribute to the 4MB maximum row length. Binary columns that allow for more than 4kb of data, such as a `BLOB` the first 8 KB of data is stored in-row and contributes to the 4MB maximum row length.
- Oracle GoldenGate supports the maximum number of columns per table that is supported by the database.
- Oracle GoldenGate supports the maximum column size that is supported by the database.
- Oracle GoldenGate supports tables that contain only one column, except when the column contains one of the following data types:
  - `LOB`
  - `LONG`
  - `LONG VARCHAR`
  - `Nested table`
  - User Defined Type (UDT)

  - `VARRAY`
  - `XMLType`
- Set `DBOPTIONS ALLOWUNUSEDCOLUMN` before you replicate from and to tables with unused columns.
- Oracle GoldenGate supports tables with these partitioning attributes:
  - Range partitioning
  - Hash Partitioning Interval Partitioning
  - Composite Partitioning
  - Virtual Column-Based Partitioning
  - Reference Partitioning
  - List Partitioning
- Oracle GoldenGate supports tables with virtual columns, but does not capture change data for these columns or apply change data to them: The database does not write virtual columns to the transaction log, and the Oracle Database does not permit DML on virtual columns. For the same reason, initial load data cannot be applied to a virtual column. You can map the data from virtual columns to non-virtual target columns.
- Oracle GoldenGate will not consider unique/index with virtual columns.
- Oracle GoldenGate supports replication to and from Oracle Exadata. To support Exadata Hybrid Columnar Compression, the source database compatibility must be set to 11.2.0.0.0 or higher.
- Oracle GoldenGate supports Transparent Data Encryption (TDE).
- Oracle GoldenGate supports `TRUNCATE` statements as part of its DDL replication support, or as standalone functionality that is independent of the DDL support.
- Oracle GoldenGate supports the capture of direct-load `INSERT`, with the exception of SQL*Loader direct-path insert for XML Binary and XML Object Relational. Supplemental logging must be enabled, and the database must be in archive log mode. The following direct-load methods are supported.
  - `/*+ APPEND */` hint
  - `/*+ PARALLEL */` hint
  - `SQLLDR` with `DIRECT=TRUE`
- Oracle GoldenGate fully supports capture from compressed objects for Extract.
- Oracle GoldenGate supports XA and PDML distributed transactions.
- Oracle GoldenGate supports DML operations on tables with `FLASHBACK ARCHIVE` enabled. However, Oracle GoldenGate does not support DDL that creates tables with the `FLASHBACK ARCHIVE` clause or DDL that creates, alters, or deletes the flashback data archive itself.

Limitations of Support for Views

These limitations apply to Extract.

- Oracle GoldenGate supports capture from a view when Extract is in initial-load mode (capturing directly from the source view, not the redo log).
- Oracle GoldenGate does not capture change data from a view, but it supports capture from the underlying tables of a view.

Limitations of Support for Materialized Views

Materialized views are supported by Extract with the following limitations.

- Materialized views created `WITH ROWID` are not supported.
- The materialized view log can be created `WITH ROWID`.
- The source table must have a primary key.
- Truncates of materialized views are not supported. You can use a `DELETE FROM` statement.
- DML (but not DDL) from a full refresh of a materialized view is supported. If DDL support for this feature is required, open an Oracle GoldenGate support case.
- For Replicat the `Create MV` command must include the `FOR UPDATE` clause
- Either materialized views can be replicated or the underlying base table(s), but not both.

Limitations of Support for Clustered Tables

Indexed clusters are supported by Extract while hash clusters are not supported.

Sequences and Identity Columns

- Identity columns are supported from Oracle database 18c onward and requires Extract, Parallel Replicat in Integrated mode, or Integrated Replicat.
- Oracle GoldenGate supports the replication of sequence values and identity columns in a unidirectional and active-passive high-availability configuration.
- Oracle GoldenGate ensures that the target sequence values will always be higher than those of the source (or equal to them, if the cache is zero).

Limitations of Support for Sequences

These limitations apply to Extract.

- Oracle GoldenGate does not support the replication of sequence values in an active-active bi-directional configuration.
- The cache size and the increment interval of the source and target sequences must be identical. The cache can be any size, including 0 (`NOCACHE`).
- The sequence can be set to cycle or not cycle, but the source and target databases must be set the same way.
- Tables with default sequence columns are excluded from replication for Extract.

Non-supported Objects and Operations in Oracle DML

The following are additional Oracle objects or operations that are not supported by Extract:

- `REF` are supported natively for compatibility with Oracle Database 12.2 and higher, but not primary-key based `REF`s (`PKREF`s)
- Sequence values in an active-active bi-directional configuration
- Database Replay
- Tables created as `EXTERNAL`

Details of Support for Objects and Operations in Oracle DDL

This topic outlines the Oracle objects and operation types that Oracle GoldenGate supports for the capture and replication of DDL operations.

Supported Objects and Operations in Oracle DDL

DDL capture support is integrated into the database logmining server. You must set the database parameter compatibility to 11.2.0.4.0 or higher. Extract supports DDL that includes password-based column encryption, such as:

- `CREATE TABLE t1 (a number, b varchar2(32) ENCRYPT IDENTIFIED BY my_password);`
- `ALTER TABLE t1 ADD COLUMN c varchar2(64) ENCRYPT IDENTIFIED BY my_password;`

The following additional statements apply to Extract with respect to DDL support.

- All Oracle GoldenGate topology configurations are supported for Oracle DDL replication.
- Active-active (bi-directional) replication of Oracle DDL is supported between two (and only two) databases that contain identical metadata.
- Oracle GoldenGate supports DDL on the following objects:
    - clusters
    - directories
    - functions
    - indexes
    - packages
    - procedure
    - tables
    - tablespaces
    - roles
    - sequences
    - synonyms
    - triggers
    - types
    - views
    - materialized views
    - users
    - invisible columns
- Oracle Edition-Based Redefinition (EBR) database replication of Oracle DDL is supported for Extract for the following Oracle Database objects:
    - functions
    - library
    - packages (specification and body)
    - procedure
    - synonyms
    - types (specification and body)
    - views

- From Oracle GoldenGate 21c onward, DDLs that are greater than 4 MB in size will be provided replication support.
- From Oracle GoldenGate 23c onwards, SQL domains are supported.
- Oracle GoldenGate is capable of managing tables with 4000 columns if the row size is less than 4MB.
- Oracle GoldenGate supports Global Temporary Tables (GTT) DDL operations to be visible to Extract so that they can be replicated. You must set the `DDLOPTIONS` parameter to enable this operation because it is not set by default.
- Oracle GoldenGate supports dictionary for use with `NOUSERID` and `TRANLOGOPTIONS GETCTASDML`. This means that Extract receives object metadata from the LogMiner dictionary without querying the dictionary objects. Oracle GoldenGate uses the dictionary automatically when the source database compatibility parameter is greater than or equal to 11.2.0.4.

  When using dictionary and trail format in the Oracle GoldenGate release 12.2.*x*, Extract requires the Logminer patch to be applied on the mining database if the Oracle Database release is earlier than 12.1.0.2.
- Oracle GoldenGate supports replication of invisible columns in Extract. Trail format release 12.2 is required. Replicat must specify the `MAPINVISIBLECOLUMNS` parameter or explicitly map to invisible columns in the `COLMAP` clause of the `MAP` parameter.

  If `SOURCEDEFS` or `TARGETDEFS` is used, the metadata format of a definition file for Oracle tables must be compatible with the trail format. Metadata format 12.2 is compatible with trail format 12.2, and metadata format earlier than 12.2 is compatible with trail format earlier than 12.2. To specify the metadata format of a definition file, use the `FORMAT RELEASE` option of the `DEFSFILE` parameter when the definition file is generated in `DEFGEN`.
- DDL statements to create a namespace context (`CREATE CONTEXT`) are captured by Extract and applied by Replicat.
- Extract in pump mode supports the following DDL options:
  - `DDL INCLUDE ALL`
  - `DDL EXCLUDE ALL`
  - `DDL EXCLUDE OBJNAME`

  The `SOURCECATALOG` and `ALLCATALOG` option of `DDL EXCLUDE` is also supported.

  If no DDL parameter is specified, then all DDLs are written to trail. If `DDL EXCLUDE OBJNAME` is specified and the object owner is does not match an exclusion rule, then it is written to the trail.
- Starting with Oracle database 21c, the following DDL is available to support blocking of DML/DDL changes that are not replicated by Oracle GoldenGate:

  ```
  ALTER DATABASE [ENABLE | DISABLE] goldengate blocking mode;
  ```

  When Oracle GoldenGate blocking mode is enabled, DMLs that use `support_mode NONE` in tables and execute unsupported Oracle PL/SQL statements will fail with the following error:

  ```
  ORA-26981: "operation was unsupported during Oracle GoldenGate blocking
  mode"
  ```

For Oracle database 21c, the following features cause a table to have `support_mode NONE` in Oracle GoldenGate:

  - `BFILE` as an attribute of ADT column, or typed table
  - Table with no scalars
  - OLAP AW$ table
  - Sharded queue table
  - Sorted Hash Cluster Table
  - Primary key constraint on ADT attribute in relational table
  - Primary key/unique key constraint on long `raw`/`varchar` (over 4000 bytes)
  - `V$DATABASE` column, `Goldengate_Blocking_Mode` can be queried to determine the current blocking mode status.

- For DDL auto capture mode:
  - It is relevant only for `DDL INCLUDE MAPPED` because Extract captures DDLs based on `TABLE` and `TABLEEXCLUDE` parameter.
  - Only table-related DDLs can be auto-captured.
  - DDLs to enable auto capture at table level:

    ```
    CREATE/ALTER TABLE … ENABLE LOGICAL REPLICATION ALLKEYS;
    ```

    or

    ```
    CREATE/ALTER TABLE … ENABLE LOGICAL REPLICATION ALLOW NOVALIDATE KEYS;
    ```

- The following operations are supported for partition related DDLs and partition maintenance operations
  - Drop partition:

    If a partition is recreated with the same name, then it will get a new object number. The internal caches are cleared to minimize space consumption when a drop partition DDL is processed.
  - Truncate partition:

    Partition name and object number stays the same. Base table object version stays the same.
  - Rename partition:

    The partition object number stays the same but gets a new name. The base table's object version gets bumped. In memory name cache will get invalidated upon seeing this DDL and repopulated upon the next DML. The cache, which stores if a given partition object number is interesting or not will also need to be reevaluated as a the new partition name may switch from filtered to not filtered or vice versa.
  - Exchange partition:

    Exchanges data in a partition with that in a table or vice versa. The obj# of the partition being exchanged does not change. Dataobj# does change but is not used by Extract. The partition itself still belongs to the same table.
  - Merge partition:

Merges one or more partitions into a new partition. The DDL creates the new partition and drops the partitions from which it was merged. In memory caches should be cleared to save space and the user should ensure proper filter rules for the newly created partition.

- Split partition:

  The partition being split keeps its original name and object number and new partition is created for the split data. The user must ensure partition filter rules are correct for the newly created partition.

- Coalesce partition:

  Reduces the number of partitions in a hash partitioned table. The specific partition that is coalesced is selected by the database, and is dropped after its contents have been redistributed. The remaining partitions keep their same name and object number. The internal caches should be cleared to minimize space consumption.

- Modify partition:

  Modifies default and real attributes of partitions, apart from adding or dropping of values for list partitions. All modifications leave the partitions name and object number intact.

- Move partition:

  Partition data is moved to a new tablespace. Partition name and number remain the same.

- Redef table:

  `dbms_redefinition` can be used to partition a table through the use of an interim table. The partitions are created on the interim table and after the `finish_redef` operation, the tables swap names. The partitions created on the interim table keep their names and object numbers when the tables are swapped. The Extract filter cache, needs to be reevaluated upon `finish_redef` as the partitions now belong to the base table. The user must ensure proper filter rules.

- Redef partition:

  When redefining a table, the partitions follow from the original table to the interim table. For example, consider the case where the original table has partitions, which live in the `USER` tablespace, and the interim table is created with no partitions and the table lives in the `NEW` tablespace. In this case, after the `finish_redef` operation, when the tables are swapped the partition still lives in the `USER` tablespace. Redef partition allows a partition to be moved to the interim table's `NEW` tablespace. The partition retains its name and object number.

- System generated partition names:

  When partitions are created automatically for hash partitions and operations such as split partition, the partition name is in the form of `SYS_P` *`sequence value`*. Similarly, subpartitions are of the form `SYS_SUBP` *`sequence value`*. It is recommended that the partition is renamed before excepting DML to conform to filter rules.

Non-supported Objects and Operations in Oracle DDL

Here's a list of non-supported objects and operations in Oracle DDL.

Excluded Objects

The following names or name prefixes are considered Oracle-reserved and must be excluded from the Oracle GoldenGate DDL configuration. Oracle GoldenGate will ignore objects that contain these names.

Excluded schemas:

```
"ANONYMOUS", // HTTP access to XDB
"APPQOSSYS", // QOS system user
"AUDSYS", // audit super user
"BI", // Business Intelligence
"CTXSYS", // Text
"DBSNMP", // SNMP agent for OEM
"DIP", // Directory Integration Platform
"DMSYS", // Data Mining
"DVF", // Database Vault
"DVSYS", // Database Vault
"EXDSYS", // External ODCI System User
"EXFSYS", // Expression Filter
"GSMADMIN_INTERNAL", // Global Service Manager
"GSMCATUSER", // Global Service Manager
"GSMUSER", // Global Service Manager
"LBACSYS", // Label Security
"MDSYS", // Spatial
"MGMT_VIEW", // OEM Database Control
"MDDATA",
"MTSSYS", // MS Transaction Server
"ODM", // Data Mining
"ODM_MTR", // Data Mining Repository
"OJVMSYS", // Java Policy SRO Schema
"OLAPSYS", // OLAP catalogs
"ORACLE_OCM", // Oracle Configuration Manager User
"ORDDATA", // Intermedia
"ORDPLUGINS", // Intermedia
"ORDSYS", // Intermedia
"OUTLN", // Outlines (Plan Stability)
"SI_INFORMTN_SCHEMA", // SQL/MM Still Image
"SPATIAL_CSW_ADMIN", // Spatial Catalog Services for Web
"SPATIAL_CSW_ADMIN_USR",
"SPATIAL_WFS_ADMIN", // Spatial Web Feature Service
"SPATIAL_WFS_ADMIN_USR",
"SYS",
"SYSBACKUP",
"SYSDG",
"SYSKM",
"SYSMAN", // Adminstrator OEM
"SYSTEM",
"TSMSYS", // Transparent Session Migration
"WKPROXY", // Ultrasearch
"WKSYS", // Ultrasearch
"WK_TEST",
"WMSYS", // Workspace Manager
"XDB", // XML DB
"XS$NULL",
"XTISYS", // Time Index
```

Special schemas:

```
"AURORA$JIS$UTILITY$", // JSERV
"AURORA$ORB$UNAUTHENTICATED", // JSERV
"DSSYS", // Dynamic Services Secured Web Service
"OSE$HTTP$ADMIN", // JSERV
"PERFSTAT", // STATSPACK
"REPADMIN",
"TRACESVR" // Trace server for OEM
```

Excluded tables (the * wildcard indicates any schema or any character):

```
"*.AQ$*", // advanced queues
"*.DR$*$*", // oracle text
"*.M*_*$$", // Spatial index
"*.MLOG$*", // materialized views
"*.OGGQT$*",
"*.OGG$*", // AQ OGG queue table
"*.ET$*", // Data Pump external tables
"*.RUPD$*", // materialized views
"*.SYS_C*", // constraints
"*.MDR*_*$", // Spatial Sequence and Table
"*.SYS_IMPORT_TABLE*",
"*.CMP*$*", // space management, rdbms >= 12.1
"*.DBMS_TABCOMP_TEMP_*", // space management, rdbms < 12.1
"*.MDXT_*$*" // Spatial extended statistics tables
```

Other Non-supported DDL

Oracle GoldenGate does not support the following:

- DDL on nested tables.
- DDL on identity columns.
- `ALTER DATABASE` and `ALTER SYSTEM` (these are not considered to be DDL) Using dictionary, you can replicate `ALTER DATABASE DEFAULT EDITION` and `ALTER PLUGGABLE DATABASE DEFAULT EDITION`. All other `ALTER [PLUGABLE] DATABASE` commands are ignored.
- DDL on a standby database.
- Database link DDL.
- DDL that creates tables with the `FLASHBACK ARCHIVE` clause and DDL that creates, alters, or deletes the flashback data archive itself. DML on tables with `FLASHBACK ARCHIVE` is supported.
- Some DDL will generate system generated object names. The names of system generated objects may not always be the same between two different databases. So, DDL operations on objects with system generated names should only be done if the name is exactly the same on the target.

Integrating Oracle GoldenGate Microservices Architecture into a Cluster

If you installed Oracle GoldenGate in a cluster, take the following steps to integrate Oracle GoldenGate within the cluster solution.

Oracle GoldenGate Microservices Architecture provides REST-enabled services with features including remote configuration, administration, and monitoring through HTML5 web pages, command line interfaces, and APIs.

General Requirements in a Cluster

1. Configure the Oracle Grid Infrastructure Bundled Agent (XAG) to automatically manage the GoldenGate processes on the cluster nodes. See Configuring Oracle GoldenGate with Oracle Grid Infrastructure Bundled Agents (XAG) to know more.

   Using the XAG makes sure that the required cluster file system is mounted before the Oracle GoldenGate processes are started. If an application virtual IP (VIP) is used in the cluster the bundled agent will also ensure the VIP is started on the correct node.

2. Configure the Oracle GoldenGate Manager process with the `AUTOSTART` and `AUTORESTART` parameters so that Manager starts the replication processes automatically.

3. Mount the shared drive on one node only. This prevents processes from being started on another node. Use the same mount point on all nodes. If you are using the Oracle Grid Infrastructure Bundled Agent, the mounting of the required file systems are automatically carried out.
4. Ensure that all database instances in the cluster have the same `COMPATIBLE` parameter setting.
5. Configure Oracle GoldenGate as directed in this documentation.

Adding Oracle GoldenGate as a Windows Cluster Resource

When installing Oracle GoldenGate in a Windows cluster, follow these instructions to establish Oracle GoldenGate as a cluster resource and configure the Manager service correctly on all nodes.

- In the cluster administrator, add the Manager process to the group that contains the database instance to which Oracle GoldenGate will connect.
- Make sure all nodes on which Oracle GoldenGate will run are selected as possible owners of the resource.
- Make certain the Manager Windows service has the following dependencies (can be configured from the Services control panel):
  - The database resource
  - The disk resource that contains the Oracle GoldenGate directory
  - The disk resource that contains the database transaction log files
  - The disk resource that contains the database transaction log backup files

PostgreSQL

Oracle GoldenGate for PostgreSQL supports capture and delivery of initial load and transactional data for supported PostgreSQL database versions.

Oracle GoldenGate for PostgreSQL supports the mapping, filtering, and transformation of source data, unless noted otherwise in this document, as well as replicating data derived from other source databases supported by Oracle GoldenGate, into PostgreSQL databases.

Preparing the Database for Oracle GoldenGate

This chapter describes how to prepare your PostgreSQL database and environment for Oracle GoldenGate.

> **Note:**
>
> `PgBouncer` is not supported for Oracle GoldenGate connections.

> **Note:**
>
> Oracle GoldenGate does not support connections to PostgreSQL that use `pgpool`.

Database Configuration

To support Oracle GoldenGate, the following parameters in the PostgreSQL database configuration file, `$PGDATA/postgresql.conf`, needs to be configured.

- For remote connectivity of an Extract or Replicat, set the PostgreSQL `listen_addresses` to allow for remote database connectivity. For example:

  ```
  listen_addresses=remotehost_ip_address
  ```

  > **Note:**
  >
  > Ensure that client authentication is set to allow connections from an Oracle GoldenGate host by configuring the `pg_hba.conf` file. For more information, refer to this document: The pg_hba.conf File

- To support Oracle GoldenGate capture, **write-ahead logging** must be set to `logical`, which adds information necessary to support transactional record decoding.

  The number of **maximum replication slots** must be set to accommodate one open slot per Extract, and in general, no more than one Extract is needed per database. If for example PostgreSQL Native Replication is already in use and is using all of the currently configured replication slots, increase the value to allow for the registration of an Extract.

  Maximum **write-ahead senders** should be set to match the maximum replication slots value.

  Optionally, **commit timestamps** can be enabled in the write-ahead log, which when set at the same time logical write-ahead logging is enabled, will track the first DML commit record from that point on, with the correct timestamp value. Otherwise, the first record encountered by Oracle GoldenGate capture will have an incorrect commit timestamp.

  ```
  wal_level = logical              # set to logical for Capture

  max_replication_slots = 1        # max number of replication slots,
                                   # one slot per Extract/client

  max_wal_senders = 1              # one sender per max repl slot

  track_commit_timestamp = on      # optional, correlates tx commit time
                                   # with begin tx log record (useful for
                                   # timestamp-based positioning)
  ```

- After making any of the preceding changes, restart the database.

Database Settings for PostgreSQL Cloud Databases

Use these instructions to manage the database settings for Azure Database for PostgreSQL, Amazon Aurora PostgreSQL, Amazon RDS for PostgreSQL, and Google Cloud SQL for PostgreSQL.

Azure Database for PostgreSQL

When configuring Oracle GoldenGate for PostgreSQL Capture against an Azure Database for PostgreSQL, **logical decoding** must be enabled and set to `LOGICAL`.

Read the Microsoft Documentation for instructions.

Other database settings for Azure Database for PostgreSQL can be managed through the **Server parameters** section of the database instance.

For connections to an Azure Database for PostgreSQL instance, the default Azure Connection Security settings require SSL connections. To adhere to this requirement, further steps are required to support SSL connections with Oracle GoldenGate. Follow the content listed under Configuring SSL Support for PostgreSQL for more information.

Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL

For Amazon Aurora PostgreSQL and Amazon RDS for PostgreSQL, database settings are modified within parameter groups. Review the Amazon AWS documentation for information on how to edit database settings within a new parameter group and assign it to a database instance.

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html

- Ensure that the database configuration settings listed previously are correct, by verifying them in the parameter group assigned to the instance.
- The `wal_level` setting for Amazon database services is configured with a parameter called `rds.logical_replication`, whose default is `0` and should be set to `1` if the database is to be used as source database for Oracle GoldenGate Capture.

Limitation:

On Amazon Aurora PostgreSQL version 12.17, if upper case `SHOW` command is executed, it reports the following error:

```
"ERROR: must be superuser or replication role to run this operation."
```

You must use lower case `SHOW` command to avoid this error.

Google Cloud SQL for PostgreSQL

When configuring an Oracle GoldenGate for PostgreSQL Extract for a Google Cloud SQL for PostgreSQL database, logical decoding must be set and is done by setting the `cloudsql.logical_decoding` variable to `ON`. Follow the instructions provided by Google on how to enable this database flag. For more information, see https://cloud.google.com/sql/docs/postgres/flags#postgres-l.

Establishing Oracle GoldenGate Credentials

Learn how to create database users for the processes that interact with the database, assign the correct privileges, and secure the credentials from any unauthorized use.

Assigning Credentials to Oracle GoldenGate

Oracle GoldenGate processes require a database user to capture and deliver data to a PostgreSQL database and it is recommended to create a dedicated PostgreSQL database user for Extract and Replicat.

The following database user privileges are required for Oracle GoldenGate to capture from and apply to a PostgreSQL database.

Privilege	Extract	Replicat	Purpose
Database Replication Privileges			

Privilege	Extract	Replicat	Purpose
`CONNECT`	Yes	Yes	Required for database connectivity. `GRANT CONNECT ON DATABASE` *`dbname`* `TO` *`gguser;`*
`WITH REPLICATION`	Yes	NA	Required for the user to register Extract with a replication slot. `ALTER USER` *`gguser`* `WITH REPLICATION;`
`WITH SUPERUSER`	Yes	NA	Required to enable table level supplemental logging (`ADD TRANDATA`) but can be revoked after `TRANDATA` is enabled for the table(s). `ALTER USER` *`gguser`* `WITH SUPERUSER;` For Azure Database for PostgreSQL, only the Admin user has `SUPERUSER` authority and is the only user that can enable `TRANDATA`.
`USAGE ON SCHEMA`	Yes	Yes	For metadata access to tables in the schema to be replicated. `GRANT USAGE ON SCHEMA` *`tableschema`* `TO` *`gguser;`*
`SELECT ON TABLES`	Yes	Yes	Grant select access on tables to be replicated. `GRANT SELECT ON ALL TABLES IN SCHEMA` *`tableschema`* `TO` *`gguser;`*
`INSERT, UPDATE, DELETE,TRUNCATE` on target tables. Alternatively, if replicating every table, then you can use the `GRANT INSERT, UPDATE, DELETE, TRUNCATE ON ALL TABLES IN SCHEMA TO...` to the Replicat user, instead of granting `INSERT, UPDATE, DELETE` to every table.	NA	Yes	Apply replicated DML to target objects. `GRANT INSERT, UPDATE, DELETE, TRUNCATE ON TABLE` *`tablename`* `TO gguser;`
Heartbeat and Checkpoint Table Privileges			

Privilege	Extract	Replicat	Purpose
`CREATE ON DATABASE`	Yes	Yes	Required by the Extract and Replicat user to add an Oracle GoldenGate schema for heartbeat and checkpoint table creation. `GRANT CREATE ON DATABASE` *`dbname`* `TO gguser;` Alternatively, if `GGSCHEMA` is the same as the user, then the objects can be created under the user by issuing `CREATE SCHEMA AUTHORIZATION` *`ggsuser;`*
`CREATE, USAGE ON SCHEMA`	Yes	Yes	For heartbeat and checkpoint table creation/deletion if the Extract or Replicat user does not own the objects. `GRANT CREATE, USAGE ON SCHEMA` *`ggschema`* `TO` *`gguser;`*
`EXECUTE ON ALL FUNCTIONS`	Yes	Yes	For heartbeat update and purge function execution if the user calling the functions does not own the objects. `GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA` *`ggschema`* `TO` *`gguser;`*
`SELECT, INSERT, UPDATE, DELETE`	Yes	Yes	For heartbeat and checkpoint table inserts, updates and deletes if the user does not own the objects. `GRANT SELECT, INSERT, UPDATE, DELETE, ON ALL TABLES IN SCHEMA` *`ggschema`* `TO` *`gguser;`*

Securing the Oracle GoldenGate Credentials

To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, an Oracle GoldenGate database user.

Oracle GoldenGate provides different options for securing the log-in credentials assigned to Oracle GoldenGate processes. The recommended option is to use a credential store. You can create one credential store and store it in a shared location where all installations of Oracle GoldenGate can access it, or you can create a separate one on each system where Oracle GoldenGate is installed.

The credential store stores the user name and password for each of the assigned Oracle GoldenGate users. A user ID is associated with one or more aliases, and it is the alias that is supplied in commands and parameter files, not the actual user name or password. The credential file can be partitioned into domains, allowing a standard set of aliases to be used for the processes, while allowing the administrator on each system to manage credentials locally.

Prepare a Database Connection

Oracle GoldenGate connects to a PostgreSQL database through an ODBC (Open Database Connectivity) driver and requires a system Data Source Name (DSN) be created with the correct database connection details for each source and target PostgreSQL database.

Ensure that you have installed and configured the driver prior to creating a DSN, by following the instruction in Configure a Database Connection in Linux .

> **Note:**
>
> Do not use PgBouncer setup for Extract connections to the PostgreSQL database because PgBouncer does not understand the replication protocol, because of which the Extract connection is not identified as replication connection.

> **Note:**
>
> Oracle GoldenGate does not support connections to PostgreSQL that use `Pgpool`.

This section contains instructions for setting up the DSN connections that Extract and Replicat will use.

Configure a Database Connection in Linux

To create a database connection in Linux, set up a data source name (DSN) inside the `/etc/odbc.ini` file.

1. Create a DSN for each source or target database in the `/etc/odbc.ini` file.

   ```
   sudo vi /etc/odbc.ini

   #Sample DSN entries
   [ODBC Data Sources]
   ```

```
PG_src=Oracle GoldenGate PostgreSQL Wire Protocol
PG_tgt=Oracle GoldenGate PostgreSQL Wire Protocol


[ODBC]
IANAAppCodePage=4
InstallDir=/u01/app/ogg


[PG_src]
Driver=/u01/app/ogg/lib/ggpsql25.so
Description=Oracle GoldenGate PostgreSQL Wire Protocol
Database=sourcedb
HostName=remotehost
PortNumber=5432


[PG_tgt]
Driver=/u01/app/ogg/lib/ggpsql25.so
Description=Oracle GoldenGate PostgreSQL Wire Protocol
Database=targetdb
HostName=remotehost
PortNumber=5432
```

In the preceding examples:

`PG_src` and `PG_tgt` are user defined names of a source and target database DSN that will be referenced by Oracle GoldenGate processes, such as Extract or Replicat. DSN names are allowed up to 32 alpha-numeric characters in length, excluding special keyboard characters except for the underscore and dash.

`IANAAppCodePage=4` is the default setting but can be modified according to the following guidance, when the database character set is not Unicode.

https://docs.progress.com/bundle/datadirect-connect-odbc-71/page/IANAAppCodePage_9.html#IANAAppCodePage_9

`InstallDir` is the location of the Oracle GoldenGate installation folder.

`Driver` is the location of the Oracle GoldenGate installation home, `$OGG_HOME/lib/ggpsql25.so` file.

`Database` is the name of the source or target database.

`HostName` is the database host IP address or host name.

`PortNumber` is the listening port of the database.

You can also provide a `LogonID` and `Password` for the Extract or Replicat user, but these will be stored in clear text and it is recommended instead to leave these fields out of the DSN and instead store them in the Oracle GoldenGate wallet as a credential alias, and reference them with the `USERIDALIAS` parameter in Extract and Replicat.

2. Save and close the `odbc.ini` file.

Configuring SSL Support for PostgreSQL

SSL can be enabled by configuring the PostgreSQL configuration file (`$PGDATA/postgresql.conf`). For details, see Configuring SSL Support (PostgreSQL) in the *Securing the Oracle GoldenGate Environment*.

> **Note:**
>
> Azure Database for PostgreSQL defaults to enforce SSL connections. To adhere to this requirement, perform the requirements listed here, or optionally, you can disable enforcing SSL connections from the **Connection security** settings of the database instance using the Microsoft Azure Portal.

Preparing Tables for Processing

The following table attributes must be addressed in an Oracle GoldenGate environment for PostgreSQL.

Disabling Triggers and Cascade Constraints on the Target

If Oracle GoldenGate is configured to capture DML operations from source tables that occur due to trigger operations or cascade constraints, then disable the triggers and cascade delete and cascade update constraints on the target tables.

If not disabled, the same trigger or constraint gets activated on the target table and becomes redundant because of the replicated data. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

In the Replicat `MAP` statements, map the source tables to appropriate targets, and map the child tables that the source tables reference with triggers or foreign-key cascade constraints. Triggered and cascaded child operations must be mapped to appropriate targets to preserve data integrity. Include the same parent and child source tables in the Extract `TABLE` parameters.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration. For PostgreSQL LOB types such as `text`, `xml`, `bytea`, `char`, `varchar`, Oracle

GoldenGate supports these columns as a primary key in source or target tables up to a length of 8191 bytes.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.
> For tables that have no uniqueness and have repeat rows with the same values, Replicat will Abend on update and delete operations for these rows.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

Enabling Table-Level Supplemental Logging

Enabling Supplemental logging is a process in which Oracle GoldenGate sets source database table level logging to support change data capture of source DML operations, and depending on the level of logging, to include additional, unchanged columns which would be needed in cases such as bi-directional replication with conflict detection and resolution configured.

There are four levels of table level logging in PostgreSQL, which equate to the `REPLICA IDENTITY` setting of a table, and those include `NOTHING`, `USING INDEX`, `DEFAULT`, and `FULL`.

Oracle GoldenGate requires `FULL` logging for use cases that require uncompressed trail records and Conflict Detection and Resolution, but in cases where tables have a Primary Key or Unique Index whose changes are being replicated in a simple uni-directional configuration or where full before-images or uncompressed records are not needed, then the `DEFAULT` level is acceptable. `NOTHING` and `USING INDEX` logging levels are not supported by Oracle GoldenGate and cannot be set with `ADD TRANDATA`.

The following is the syntax for issuing `ADD TRANDATA` from GGSCI.

```
GGSCI> DBLOGIN SOURCEDB dsn_name USERIDALIAS alias_name
GGSCI> ADD TRANDATA schema.tablename ALLCOLS
```

> **Note:**
>
> For tables that have a primary key or unique index, the `ALLCOLS` option is required in order to set `FULL` logging for the table, otherwise `DEFAULT` logging is set.

`FULL` logging is always set for tables without a primary key or unique index, regardless of whether `ALLCOLS` is specified or not.

To check the level of supplemental logging:

```
GGSCI> INFO TRANDATA schema.tablename
```

Configuring Replicat

This chapter contains instructions for configuring the Replicat apply process to deliver data to a target PostgreSQL database.

About Replicat

The Oracle GoldenGate Replicat for PostgreSQL reads data from Oracle GoldenGate source trail files and delivers the data to a target PostgreSQL database. The source trail data can be from any database that Oracle GoldenGate capture supports.

Available Replicats for PostgreSQL are Classic and Coordinated.

For more information on the differences between types of Replicats, review the Creating an Online Replicat Group content in the *Administering Oracle GoldenGate* guide.

Replicat Deployment Options

- **Local deployment**: For a local deployment, the target database and Oracle GoldenGate are installed on the same server. No extra consideration is needed for local deployments.
- **Remote deployment**: For a remote deployment, the target database and Oracle GoldenGate are installed on separate servers. Remote deployments are the only option available for supporting cloud databases, such as Azure for PostgreSQL or Amazon Aurora PostgreSQL.

  For remote deployments, operating system endianness between the database server and Oracle GoldenGate server needs to be the same.

  With remote deployments, low network latency is important, and it is recommended that the network latency between the Oracle GoldenGate server and the target database server be less than 1 millisecond.

Prerequisites for Creating a Replicat

Review the Installing the DataDirect driver for PostgreSQL in *Installing Oracle GoldenGate* and ensure that the DataDirect drivers are installed correctly, which varies depending on the operating system.

Ensure that the PostgreSQL Client Authentication Configuration file, `$PGDATA/pg_hba.conf`, on the database server is configured to allow connections from the Oracle GoldenGate server, if installed remotely. See https://www.postgresql.org/docs/13/auth-pg-hba-conf.html for more information.

Creating a Checkpoint Table

A checkpoint table is used by a Replicat in the target database for recovery positioning when restarting a Replicat. A checkpoint table is optional (but recommended) for a Classic Replicat and required for a Coordinated Replicat.

The checkpoint table needs to be created under an existing schema in the database and by default will attempt to create the table in the schema listed in the `GLOBALS` file, `GGSCHEMA` parameter. Ensure that the schema listed in `GLOBALS` exists in the database and that the Replicat user has permissions to use the schema and create tables in it.

These steps demonstrate creating a checkpoint table for a Classic and Coordinated Replicat.

1. Using GGSCI, connect to the DSN for the target database.

   ```
   GGSCI> DBLOGIN SOURCEDB dsn USERIDALIAS alias
   ```

2. Add the checkpoint table using the GGSCI command.

   ```
   GGSCI> ADD CHECKPOINTTABLE ggadmin.oggcheck
   ```

Creating a Replicat

These steps create a Replicat to deliver transactional data to a target PostgreSQL database.

1. In GGSCI, create the Replicat parameter file.

   ```
   EDIT PARAMS repnm
   ```

   In this sample, `repnm` is a name of the Replicat. For Classic Replicat, the name can be no more than 8 alpha-numeric characters in length. For Coordinated Replicat, the name must be five or less alpha-numeric characters in length.

2. Enter the Replicat parameters in the order shown, starting a new line for each parameter statement.

   Sample basic parameters for Classic Replicat:

   ```
   REPLICAT repnm
   TARGETDB dsn_name USERIDALIAS alias
   BATCHSQL
   GETTRUNCATES
   MAP schema.object, TARGET schema.object;
   ```

   Sample basic parameters for Coordinated Replicat:

   ```
   REPLICAT repnm
   TARGETDB dsn_name USERIDALIAS alias
   BATCHSQL
   GETTRUNCATES
   MAP schema.object1, TARGET schema.object1, THREAD (1);
   MAP schema.object2, TARGET schema.object2, THREAD (2);
   MAP schema.object3, TARGET schema.object3, THREAD (3);
   ```

Parameter	Description
`REPLICAT repnm`	`repnm` is the name of the Replicat and cannot be more than 8 alpha-numeric characters in length for Classic Replicat and 5 or less for Coordinated Replicat. For more information, see REPLICAT in *Reference for Oracle GoldenGate*.
`TARGETDB dsn_name`	Specifies the name of the database connection DSN.

Parameter	Description
`USERIDALIAS alias`	Specifies the alias of the database login credential of the user that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials.
`BATCHSQL` `GETTRUNCATES`	Optional parameters for Replicat that supports transaction batching and replication of truncate operations.

`MAP schema.object, TARGET schema.object;`
or
`MAP schema.*, TARGET schema.*;`
	Specifies the relationship between a source table and the corresponding target object or objects.
	• `MAP` specifies the source table or a wildcarded set of tables.
	• `TARGET` specifies the target table or a wildcarded set of tables.
	• `schema` is the schema name or a wildcarded set of schemas.
	• `object` is the name of a table or a wildcarded set of tables.
	• `THREAD` assigns table operations to a specific coordinated Replicat thread.
	Terminate the parameter statement with a semi-colon.
	To exclude objects from a wildcard specification, use the `MAPEXCLUDE` parameter.
	For more information and for additional options that control data filtering, mapping, and manipulation, see `MAP` in *Parameters and Functions Reference for Oracle GoldenGate*.

3. Enter any optional Replicat parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command.
4. Save and close the file.
5. Add the Replicat, which in this example, will be a Coordinated Replicat.

```
GGSCI> ADD REPLICAT repnm, COORDINATED, EXTTRAIL ./dirdat/ep,
CHECKPOINTTABLE ggadmin.oggcheck
```

6. Start the Replicat.

Additional Considerations

This chapter contains additional configuration considerations that may apply to your database environment.

Adding a Heartbeat Table

Oracle GoldenGate for PostgreSQL supports a heartbeat table configuration, with some limitations regarding the update and purge tasks, which will be pointed out later. Adding a heartbeat table to both the source and target systems is optional but is useful in determining latency issues and to which process in the replication stream such issues may be occurring.

To add a heartbeat table for a database, review the required privileges in the Database Privileges for PostgreSQL and ensure that the correct database privileges are assigned to the Extract or Replicat user.

A schema in the database needs to be created and this should match the name of the schema used for the `GGSCHEMA` parameter of the `GLOBALS` file. The schema can be a unique schema that is not the same as the Extract or Replicat user, or can be the same as that user, but should always be reserved for Oracle GoldenGate objects only and should not be part of the user table schemas being replicated.

1. Using GGSCI, connect to the DSN for the source and target databases.

   ```
   GGSCI> DBLOGIN SOURCEDB dsn USERIDALIAS alias
   ```

2. Add the heartbeat table using the GGSCI command.

   ```
   GGSCI> ADD HEARTBEATTABLE
   ```

   Optionally, for a target only database, one that is used for unidirectional replication only, you can include the `TARGETONLY` option, which will not create a heartbeat record update function.

   ```
   GGSCI> ADD HEARTBEATTABLE TARGETONLY
   ```

To learn about heartbeat update and purge functions, see:

Running the Heartbeat Update and Purge Function

The heartbeat table and associated functions are created from the `ADD HEARTBEATTABLE` command, however for PostgreSQL, there is no automatic scheduler to call the functions.

One main function controls both the heartbeat record update and the heartbeat history table purge functions. The default settings for both of these features are 60 seconds for the update frequency and 1 day for the history record purge, which deletes all records older than 30 days by default.

To call the main heartbeat record function, users should create an operating system level job that executes `“select ggschema.gg_hb_job_run();”`. When this function is called, it will take into account the update frequency settings and history record purge settings and use those values regardless of the scheduler settings for the function call.

For example, users can create a **Cron Job** with the following syntax, and have it run every minute.

```
*****PGPASSWORD="gguserpasswd" psql -U gguser -d dbname -h remotehost -p 5432
-c "select ggschema.gg_hb_job_run();" >/dev/null
 2>&1
```

pgAdmin, or **pg_cron** are other programs that could be used to schedule the function call.

Enabling Bi-Directional Loop Detection

Loop detection is a requirement for bi-directional and multi-directional implementations of Oracle GoldenGate, so that an Extract for one source database does not recapture transactions sent by a Replicat from another source database.

With the CDC Extract capture method, by default, any transaction committed by a Replicat into a database where an Extract is configured, will recapture that transaction from the Replicat as long as supplemental logging is enabled for those tables that the Replicat is delivering to.

In order to ignore recapturing transactions that are applied by a Replicat, you must use the `TRANLOGOPTIONS FILTERTABLE` parameter for the CDC Extract. The table used as the filtering table will be the Oracle GoldenGate checkpoint table that you must create for the Replicat.

To create a Filter Table and enable Supplemental Logging:

1. On each source database, ensure that a checkpoint table for use by Replicats has been created. For example:

   ```
   ADD CHECKPOINTTABLE ggadmin.oggcheck
   ```

   It is recommended that you use the same schema name as used in the `GGSCHEMA` parameter of the `GLOBALS` file.
2. Enable supplemental logging for the checkpoint table. For example:

   ```
   ADD TRANDATA ggadmin.oggcheck ALLCOLS
   ```

3. Ensure that the Replicat is created with the checkpoint table information.

   ```
   ADD REPLICAT reptgt1, EXTTRAIL ./dirdat/e2, CHECKPOINTTABLE
   ggadmin.oggcheck
   ```

4. Configure each Extract with the `IGNOREREPLICATES` (on by default) and `FILTERTABLE` parameters, using the Replicat's checkpoint table for the filtering table.

   ```
   TRANLOGOPTIONS IGNOREREPLICATES
   TRANLOGOPTIONS FILTERTABLE ggadmin.oggcheck
   ```

   > **Note:**
   >
   > Oracle GoldenGate for PostgreSQL supports only one `FILTERTABLE` statement per Extract, so for multi-directional implementations, ensure each Replicat uses the same checkpoint table in the database that they deliver to.

Deleting an Extract

When removing an individual Extract from use against a source PostgreSQL database, or uninstalling Oracle GoldenGate, the Extract that was registered with a replication slot in the database must be unregistered in order to remove the replication slot entry, otherwise an ever-increasing database log size can occur.

Perform the following steps to remove and unregister the Extract when no longer needed.

1. Using GGSCI, connect to the DSN for the source and target databases.

   ```
   GGSCI> DBLOGIN SOURCEDB dsn USERIDALIAS alias
   ```

2. Delete the Extract first.

   ```
   GGSCI> DELETE EXTRACT extname
   ```

3. Unregister the Extract.

   ```
   GGSCI> UNREGISTER EXTRACT extname
   ```

Removing Table-level Supplemental Logging

If a table is no longer required to be captured by Oracle GoldenGate and the `TABLE` parameter for the table has been removed from the Extract parameter file, or `TABLEEXCLUDE` is used to exclude the table from a wildcard statement, then supplemental logging can be removed from the table.

> **Note:**
>
> If the Extract resolves a table that does not have supplemental logging enabled, it will Abend depending on the type of DML operation.

Using `DELETE TRANDATA` to remove supplemental logging sets the Replicat Identity level of the table to `NOTHING`. Supplemental logging can be disabled using the `DELETE TRANDATA` command within GGSCI.

The following is the syntax for issuing `DELETE TRANDATA` from GGSCI.

```
GGSCI> DBLOGIN SOURCEDB dsn_name USERIDALIAS alias_name
 GGSCI> DELETE TRANDATA schema.tablename
```

To check the level of supplemental logging:

```
GGSCI> INFO TRANDATA schema.tablename
```

Understanding What's Supported for PostgreSQL

This chapter contains information on supported features for Oracle GoldenGate for PostgreSQL:

Supported Databases

The following are supported databases and limitations for Oracle GoldenGate for PostgreSQL:

- Only user databases are supported for capture and delivery.
- Oracle GoldenGate does not support capture from archived logs.
- Capture and delivery are not supported against replica, standby databases.

- High Availability:
    - Oracle GoldenGate Extract does *not* support seamless role transitioning from a primary to a secondary Extract with PostgreSQL high availability configurations. However, manual procedural operations could be followed to provide continuity from the new primary Extract.
    - For more information, see the details available in the my Oracle Support note, *Oracle GoldenGate Procedures for PostgreSQL HA Failover (Doc ID 2818379.1)*.

Details of Supported PostgreSQL Data Types

This topic describes data types that are supported by Oracle GoldenGate, as well as those that are not supported.

Supported PostgreSQL Data Types

Here's a list of PostgreSQL data types that Oracle GoldenGate supports along with the limitations of this support.

- `bigint`
- `bigserial`
- `bit(n)`
- `bit varying(n)`
- `boolean`
- `bytea`
- `char (n)`
- `cidr`
- `citext`
- `date`
- `decimal`
- `double precision`
- `Enumerated Types`
- `inet`
- `integer`
- `interval`
- `json`
- `jsonb`
- `macaddr`
- `macaddr8`
- `money`
- `numeric`
- `real`
- `serial`

- `smallint`
- `smallserial`
- `text`
- `time` with/without timezone
- `timestamp` with/without timezone
- `uuid`
- `varchar(n)`
- `varbit`
- `xml`

Limitations of Support

- If columns of `char`, `varchar`, `text`, or `bytea` data types are part of a primary or unique key, then the maximum individual lengths for these columns must not exceed 8191 bytes.
- Columns of data type `CITEXT` that are part of the Primary Key are supported up to 8000 bytes in size. `CITEXT` columns that are greater than 8000 bytes and are part of the Primary Key are not supported.
- `real`, `double`, `numeric`, `decimal: NaN` input values are not supported.
- The following limitations apply to `bit`/`varbit` data types:
  - They are supported up to 4k in length. For lengths greater than 4k the data is truncated and only the lower 4k bits are captured.
  - The source bit(n) column can be applied only onto a character type column on a non-PostgreSQL target and can be applied onto a `char` type or a `bit/varbit` column on PostgreSQL target.
- The following limitations are applicable to both `timestamp` with time zone and `timestamp` without time zone:
  - The `timestamp` data with BC or AD tags in the data is not supported.
  - The `timestamp` data older than 1883-11-18 12:00:00 is not supported.
  - The `timestamp` data with more than 4 digits in the YEAR component is not supported.
  - `Infinity/-Infinity` input strings for `timestamp` columns are not supported.
- The following are the limitations when using `interval`:
  - The capture of mixed sign `interval` data from `interval` type columns is not supported. You can use `DBOPTIONS ALLOWNONSTANDARDINTERVALDATA` in the Extract parameter file to capture the mixed sign `interval` data (or any other format of `interval` data, which is not supported by Oracle GoldenGate) as a string (not as standard `interval` data).

    The following are a few examples of data that gets written to the trail file, on using the `DBOPTIONS ALLOWNONSTANDARDINTERVALDATA` in the Extract param file:
  - `+1026-9 +0 +0:0:22.000000` is interpreted as 1026 years, 9 months, 0 days, 0 hours, 0 minutes, 22 seconds.
  - `-0-0 -0 -8` is interpreted as 0 years, 0 months, 0 days, -8 hours.
  - `+1-3 +0 +3:20` is interpreted as 1 year, 3 months, 0 days, 3 hours, 20 minutes.

- Replicat: If the source interval data was captured using `DBOPTIONS ALLOWNONSTANDARDINTERVALDATA` and written as a string to the trail, the corresponding source column is allowed to be mapped to either a `char` or a `binary` type column on the target.
- `date` limitations are:
  - The `date` data with BC or AD tags in the data is not supported.
  - `Infinity`/`-Infinity` input strings for `date` columns are not supported.
- Columns of `text`, `json`, `xml`, `bytea`, `char (>8191)`, `varchar (>8191)` are treated as LOB columns and have the following limitations:
  - When using `GETUPDATEBEFORES`, the before image of LOB columns is never logged.
  - When using `NOCOMPRESSUPDATES`, `LOB` columns are logged in the after image only if they were modified.
- The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.

Non-Supported PostgreSQL Data Types

Oracle GoldenGate for PostgreSQL does not support the following data types:

- `Arrays`
- `box`
- `bpchar`
- `circle`
- `Composite Types`
- `Domain Types`
- `line`
- `lseq`
- `Object Identifiers Types`
- `path`
- `pg_lsn`
- `pg_snapshot`
- `point`
- `polygon`
- `Pseudo-Types`
- `Range Types`
- `tsquery`
- `tsvector`
- `User-defined Types (UDTs)`

- Extensions and Additional Supplied Modules listed at: https://www.postgresql.org/docs/current/contrib.html are not supported by Oracle GoldenGate unless explicitly listed under Supported PostgreSQL Data Types.

> **Note:**
>
> If the Extract parameter file contains a table with unsupported data types, the Extract will stop with an error message. To resume replication, remove the table from the Extract file or remove the column from the table with an unsupported data type.

> **Note:**
>
> If an Extension or Additional Supplied Module is supported, it will be explicitly added to the Supported PostgreSQL data types list.

Supported Objects and Operations for ProtgreSQL

This topic describes objects and operations that are supported by Oracle GoldenGate.

- Oracle GoldenGate for PostgreSQL only supports DML operations (Insert/Update/Deletes). DDL replication is not supported.
- Oracle GoldenGate for PostgreSQL supports replication of truncate operations beginning with PostgreSQL 11 and above, and requires the `GETTRUNCATES` parameter in Extract and Replicat.
- Case Sensitive/Insensitive names usage includes:
  - Unquoted names are case-insensitive and are implicitly lowercase. For example, `CREATE TABLE MixedCaseTable` and `SELECT * FROM mixedcasetable` are equivalent.
  - Quoted table and column names are case-sensitive and need to be listed correctly in Extracts and Replicats and with Oracle GoldenGate commands. For example, `TABLE appschema.”MixedCaseTable”` and `ADD TRANDATA appschema.”MixedCaseTable”` would be required to support a case-sensitive table name.

Tables and Views

Tables to be included for capture and delivery must meet the following requirements and must only include data types listed under *Supported PostgreSQL Data Types*.

- Oracle GoldenGate for PostgreSQL supports capture of transactional DML from user tables, and delivery to user tables.
- Oracle GoldenGate for PostgreSQL supports delivery to partitioned tables.
- Globalization is supported for object names (table /schema/database/column names) and column data.

Limitations:

- Oracle GoldenGate for PostgreSQL does not support capture and delivery for views.
- Oracle GoldenGate for PostgreSQL does not support capture from partitioned tables.

Sequences and Identity Columns

- Sequences are supported on source and target tables for unidirectional, bidirectional, and multi- directional implementations.
- Identity columns created using the `GENERATED BY DEFAULT AS IDENTITY` clause are supported on source and target tables, for unidirectional, bidirectional, and multi-directional implementations.
- Identity columns created using the `GENERATED ALWAYS AS IDENTITY` clause are not supported in target database tables and the Identity property should be removed from target tables or changed to `GENERATED BY DEFAULT AS IDENTITY`.
- For bidirectional and multi-directional implementations, define the Identity columns and sequences with an `INCREMENT BY` value equal to the number of servers in the configuration, with a different `MINVALUE` for each one.

  For example, `MINVALUE` /`INCREMENT BY` values for a bidirectional, two-database configuration would be as follows:

  Database1, set the `MINVALUE` at 1 with an `INCREMENT BY` of 2.

  Database2, set the `MINVALUE` at 2 with an `INCREMENT BY` of 2.

  For example, `MINVALUE` /`INCREMENT BY` values for a multi-directional, three-database configuration would be as follows:

  Database1, set the `MINVALUE` at 1 with an `INCREMENT BY` of 3.

  Database2, set the `MINVALUE` at 2 with an `INCREMENT BY` of 3.

  Database3, set the `MINVALUE` at 3 with an `INCREMENT B`Y of 3.

SQL Server

With Oracle GoldenGate for SQL Server, you can perform initial loads and capture transactional data from supported SQL Server versions and replicate the data to a SQL Server database or other supported Oracle GoldenGate targets, such as an Oracle Database.

Oracle GoldenGate for SQL Server supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

SQL Server Supported Versions

Certified versions of SQL Server can be found on the published certification matrix available for each release of Oracle GoldenGate, which is available at the following link:

https://www.oracle.com/middleware/technologies/fusion-certification.html

Oracle GoldenGate Extract supports Enterprise Edition and some versions of SQL Server Standard Edition. Review the Exceptions and Additonal Information column of the certification matrix to see the details of which Standard Edition versions of SQL Server are supported for Extract.

Oracle GoldenGate Delivery supports both SQL Server Enterprise and Standard editions.

Oracle GoldenGate supports remote capture and delivery for Azure SQL Database Managed Instance and remote delivery for Azure SQL Database.

Oracle GoldenGate supports remote capture and delivery for Amazon RDS for SQL Server.

Globalization Support

Oracle GoldenGate provides globalization support that lets it process data in its native language encoding. The Oracle GoldenGate apply process (Replicat) can convert data from one character set to another when the data is contained in character column types.

Requirements for Installing Oracle GoldenGate for SQL Server

To operate with SQL Server databases, Oracle GoldenGate supports the following instance, database, and other configurations.

Instance Requirements

- The SQL Server server name (`@@SERVERNAME`) must not be `NULL`.
- (Extract) For Oracle GoldenGate to capture transactional data, the SQL Server Agent must be running on the source SQL Server instance and the SQL Server Change Data Capture job must be running against the database. If SQL Server Transactional Replication is also enabled for the database, then the SQL Server Log Reader Agent must be running.
- If your data for `TEXT`, `NTEXT`, `IMAGE`, or `VARCHAR(MAX)`, `NVARCHAR(MAX)` and `VARBINARY(MAX)` columns will exceed the SQL Server default size set for the `max text repl size` option, then extend the size. Use `sp_configure` to view or adjust the current value of `max text repl size`.

  > **Note:**
  >
  > For Amazon RDS for SQL Server, to adjust instance settings, you need to use Parameter Groups instead of `sp_configure`.

- It is recommended to install the most recent Service Pack or Cummulative Update for your SQL Server instance to ensure proper functionality. For SQL Server 2012, 2014, 2016, and 2017, Microsoft has identified and fixed several important issues that directly affect the SQL Server Change Data Capture feature. This situation impacts the ability for Oracle GoldenGate to correctly capture data. The current known issues that require Microsoft patches include KB3030352, KB3166120, and KB4073684.

Database Requirements

Observe the following requirements and limitations for supporting Oracle GoldenGate:

- Only user databases are supported for capture and delivery.
- Ensure that `Auto Create Statistics` and `Auto Update Statistics` are enabled for the database.
- The database must be set to the compatibility level of the SQL Server instance version.
- Oracle GoldenGate supports SQL Server databases configured with Transparent Data Encryption (TDE).
- (Extract) The source database can be set to any recovery model that supports the change data capture feature in Microsoft SQL Server.
- If the source database was created by restoring a backup from a different instance you must synchronize the database owner SID with the SID on the new instance. Alternatively, you can use `sp_changedbowner` to set the restored database to a current login.

- (AlwaysOn) Extract supports capturing from the primary database, or a read-only, synchronous-commit mode. Asynchronous-commit mode are not supported for capture.
- Replicat performance consideration: Beginning with SQL Server 2016, Microsoft changed the default setting for the database option `TARGET_RECOVERY_TIME` from 0 to 60 seconds. It has been demonstrated in internal testing that this can reduce the Replicat's throughput. If you experience Replicat throughput degradation, consider adjusting the `TARGET_RECOVERY_TIME` setting to 0.

Limitations:

- Oracle GoldenGate does not support capture or delivery of system databases.
- Oracle GoldenGate does not support capture from contained databases.
- Source database names cannot exceed 121 characters. This limitation is due to the SQL Server stored procedures that are used to enable supplemental logging.
- If you are configuring the Oracle GoldenGate heartbeat functionality, the SQL Server database name must not exceed 107 characters.
- Capture from SQL Server databases enabled with In-Memory OLTP (in-memory optimization) is not supported. When you add a Memory Optimized Data file group to your database, Oracle GoldenGate is not allowed to enable supplemental logging for any table in the database. Conversely, if supplemental logging has been enabled for any table in the database prior to the creation of a Memory Optimized Data file group, SQL Server does not allow a Memory Optimized Data file group to be created.
- (AlwaysOn) Capture from databases configured in asynchronous-commit mode of an AlwaysOn Availability group are not supported.

Table Requirements

Tables to be included for capture and delivery must include only the data types that are listed in Supported SQL Server Data Types.

- Oracle GoldenGate supports capture of transactional DML from user tables, and delivery to user tables and writeable views.
- DDL operations are not supported.
- Oracle GoldenGate supports the maximum permitted table names and column lengths for tables that are tracked by SQL Server Change Data Capture.
- The sum of all column lengths for a table to be captured from must not exceed the length that SQL Server allows for enabling Change Data Capture for the table. If the sum of all column lengths exceeds what is allowed by SQL Server procedure `sys.sp.cdc_enable_table`, then `ADD TRANDATA` cannot be enabled for that table. The maximum allowable record length decreases as more columns are present, so there is an inverse relationship between maximum record length and the number of columns in the table.

Prepare Database Users and Privileges

Learn about required database users, privileges, and permissions for Oracle GoldenGate for SQL Server, including supported SQL Server cloud databases.

Oracle GoldenGate for SQL Server

Oracle GoldenGate processes require a database user in order to capture from and apply data to a SQL Server database and it is recommended to create a dedicated database user to be used exclusively by Oracle GoldenGate processes.

Oracle GoldenGate for SQL Server supports SQL Server authentication for all of its certified platforms and Windows authentication for Classic Architecture only when Oracle GoldenGate is installed on a Windows server.

- To use Windows authentication for Oracle GoldenGate Classic Architecture, the Extract and Replicat processes inherit the login credentials of the Manager process. By default, the Manager process runs interactively as the user logged on to the Windows server or optionally can be added as a Windows Service with a default service name of `GGSMGR`. Whichever method that the Manager process is using, the user that it is running as needs to have the required SQL Server privileges listed above.
- To use SQL Server authentication, create a dedicated SQL Server login for Extract and Replicat and assign the privileges listed below.

SQL Server and Azure SQL Managed Instance

The following user requirements and minimum database privileges and permissions are required for Oracle GoldenGate to capture from and apply to a SQL Server or Azure SQL Managed Instance database.

1. Create a dedicated login for Oracle GoldenGate for SQL Server or Azure SQL Managed Instance.
2. Add the login as a user to the `msdb` database and to the source or target database.
3. Create a schema in the source or target database, to be used for objects required for Oracle GoldenGate. This schema should map to the `GGSCHEMA` value used in the `GLOBALS` parameter file.
4. Enable the following privileges and permissions for the Oracle GoldenGate user based on whether the user is for an Extract, or for a Replicat.

Table 3-4 Privileges and Permissions for Oracle GoldenGate User

Privilege	Extract	Replicat	Syntax
msdb Database Roles and Privileges			
`SQLAgentReaderRole`	Yes	No	`ALTER ROLE SQLAgentReaderRole ADD MEMBER` *`gguser`*`;`
`SQLAgentUserRole`	Inherited	Yes	`ALTER ROLE SQLAgentUserRole ADD MEMBER` *`gguser`*`;`
`SELECT ON sysjobactivity`	Yes	No	`GRANT SELECT ON msdb.dbo.sysjobactivity TO` *`gguser`*`;`
`SELECT ON sysjobs`	Yes	No	`GRANT SELECT ON msdb.dbo.sysjobs TO` *`gguser`*`;`
User Database Roles and Privileges			

Table 3-4 (Cont.) Privileges and Permissions for Oracle GoldenGate User

Privilege	Extract	Replicat	Syntax

`SYSADMIN`	Yes	No	Required for a one time change to enable database level Change Data Capture (CDC) if not already enabled and can be revoked once `TRANDATA` has been enabled.
			`ALTER SERVER ROLE sysadmin ADD MEMBER` *`gguser`*`;`
			Database Administrators with sysadmin credentials can manually enable the database for CDC using the following, which would negate the need for the Extract user to have this privilege:
			`EXEC msdb.sys.sp_cdc_enable_db 'source_database'`

`DBOWNER`	Yes	Yes	`ALTER ROLE db_owner ADD MEMBER` *`gguser`*`;`

Amazon RDS User Permissions and Requirements

The following user requirements and minimum database privileges and permissions are required for Oracle GoldenGate to capture from and apply to an Amazon RDS for SQL Server database:

1. Create a dedicated login for Oracle GoldenGate for Amazon RDS for SQL Server.
2. Add the login as a user to the `msdb` database and to the source or target database.
3. Create a schema in the source or target database, to be used for objects required for Oracle GoldenGate. This schema should map to the `GGSCHEMA` value used in the `GLOBALS` parameter file.
4. Enable the following privileges and permissions for the Oracle GoldenGate user based on whether the user is for an Extract, or for a Replicat.

Table 3-5 Privileges and Permissions for Oracle GoldenGate User

Privilege	Extract	Replicat	Syntax
msdb Database Roles and Privileges			

`EXECUTE ON rds_cdc_enable_db`	Yes	No	`GRANT EXECUTE ON msdb.dbo.rds_cdc_enable_db TO` *`gguser`*`;`
			Database administrators with master credentials can manually enable the database for Change Data Capture using the following command, which would negate the need for the Extract user to have this permission:
			`EXEC msdb.dbo.rds_cdc_enable_db 'source_database'`

`SQLAgentOperatorRole`	Yes	No	`ALTER ROLE SQLAgentOperatorRole ADD MEMBER` *`gguser`*`;`

Table 3-5 (Cont.) Privileges and Permissions for Oracle GoldenGate User

Privilege	Extract	Replicat	Syntax
`SQLAgentUserRole`	Inherited	Yes	`ALTER ROLE SQLAgentUserRole ADD MEMBER` *`gguser`*`;`
`SELECT ON sysjobactivity`	Yes	No	`GRANT SELECT ON msdb.dbo.sysjobactivity TO` *`gguser`*`;`
`SELECT ON sysjobs`	Yes	No	`GRANT SELECT ON msdb.dbo.sysjobs TO` *`gguser`*`;`
User Database Roles and Privileges			
`DBOWNER`	Yes	Yes	`ALTER ROLE db_owner ADD MEMBER` *`gguser`*`;`

Azure SQL Database

The following user requirements and minimum database privileges and permissions are required for Oracle GoldenGate to apply to an Azure SQL Database:

1. Create a dedicated login for Oracle GoldenGate for Azure SQL Database.
2. Add the login as a user to the target database.
3. Create a schema in the target database, to be used for objects required for Oracle GoldenGate. This schema should map to the `GGSCHEMA` value used in the `GLOBALS` parameter file.
4. Enable the following privileges and permissions for the Oracle GoldenGate user.

Table 3-6 Privileges and Permissions for Oracle GoldenGate User

Privilege	Extract	Replicat	Syntax
User Database Roles and Privileges			
`DBOWNER`	NA	Yes	`ALTER ROLE db_owner ADD MEMBER` *`gguser`*`;`

Database Connectivity

Oracle GoldenGate uses ODBC and OLE DB to connect to a database:

- ODBC: The Extract process uses ODBC to connect to a source SQL Server database to obtain metadata and perform other process queries. The Replicat process uses ODBC to connect to a target SQL Server database to obtain metadata, but can optionally use it for its delivery of transactions as well. ODBC must be properly configured.
- OLE DB: By default, the Replicat process attempts to use OLE DB to connect to a target SQL Server database to perform DML operations. If the driver used only supports ODBC, then the Replicat will apply DML via ODBC. To use OLE DB in an ODBC-only driver, install the Microsoft OLE DB Driver 18 for SQL Server. Using OLE DB allows the use of the `DBOPTIONS USEREPLICATIONUSER` parameter, which supports the `Not for Replication` flag of certain table properties.
- Using the Microsoft SQL Server Native Client 11 OLE DB driver to connect to a SQL Server 2014 instance in OLEDB mode may lead to a memory leak issue (Microsoft article

2881661). Microsoft has provided a fix in SQL Server 2014 CU1 (Microsoft article 2931693). To avoid a possible memory leak, you may choose one of the following options:

  - For SQL Server 2014, upgrade the SQL Server instance to at least Cumulative Update 1.
  - Use a Microsoft supported ODBC driver.

- For Azure SQL Database, use a Microsoft supported ODBC driver.
- Always On availability group listeners are supported and are required to support read-only routing for capture against a synchronous mode secondary replica.

Configuring an Extract Database Connection

Extract connects to a source SQL Server database through an ODBC (Open Database Connectivity) connection. To create this connection, set up a data source name (DSN) using the following steps.

See Creating a Database Connection on Windows and Creating a Database Connection on Linux for instructions.

Configuring a Replicat Database Connection

Replicat can connect to the target database to perform DML operations in the following ways:

- Through ODBC.
- Through OLE DB if the SQL Server driver supports it.
- Through OLE DB as the SQL Server replication user. `NOT FOR REPLICATION` must be set on `IDENTITY` columns, foreign key constraints, and triggers.

Before you select a method to use, review the following guidelines and procedures to evaluate the advantages and disadvantages of each.

Connecting with ODBC or Default OLE DB

If Replicat connects through the default ODBC connection or through the OLE DB connection, the following limitations apply:

- To keep `IDENTITY` columns identical on source and target when using ODBC or default OLE DB, Replicat creates special operations in its transaction to ensure that the seeds are incremented on the target. These steps may reduce delivery performance.
- You must adjust or disable triggers and constraints on the target tables to eliminate the potential for redundant operations.

To use Replicat with either ODBC or OLE DB, follow these steps:

1. ODBC is used by default, unless the Microsoft OLE DB Driver for SQL Server is installed, in which case OLE DB is used. To force ODBC connectivity, add `DBOPTIONS USEODBC` to the Replicat.
2. Disable triggers and constraints on the target tables. See Disabling Triggers and Cascade Constraints on the Target.
3. To use `IDENTITY` columns in a bidirectional SQL Server configuration, define the `IDENTITY` columns to have an increment value equal to the number of servers in the configuration, with a different seed value for each one. For example, a two-server installation would be as follows:
   - Sys1 sets the seed value at 1 with an increment of 2.

- Sys2 sets the seed value at 2 with an increment of 2.

A three-server installation would be as follows:

- Sys1 sets the seed value at 1 with an increment of 3.
- Sys2 sets the seed value at 2 with an increment of 3.
- Sys3 sets the seed value at 3 with an increment of 3.

Connectiong with the OLE DB USEREPLICATIONUSER Option

If Replicat connects as the SQL Server replication user through OLE DB with the `USEREPLICATIONUSER` option, and `NOT FOR REPLICATION` is enabled for `IDENTITY` columns, triggers with foreign key constraints, the following benefits and limitations apply.

- `IDENTITY` seeds are not incremented when Replicat performs an insert. For SQL Server bidirectional configurations, stagger the seed and increment values like the example in Step 3 of the previous section.
- Triggers are disabled for the Replicat user automatically on the target to prevent redundant operations. However triggers fire on the target for other users.
- Foreign key constraints are not enforced on the target for Replicat transactions. `CASCADE` updates and deletes are not performed. These, too, prevent redundant operations.
- `CHECK` constraints are not enforced on the target for Replicat transactions. Even though these constraints are enforced on the source before data is captured, consider whether their absence on the target could cause data integrity issues.

> **Note:**
>
> Normal `IDENTITY`, trigger, and constraint functionality remains in effect for any users other than the Replicat replication user.

To use Replicat with `USEREPLICATIONUSER`, follow these steps:

> **Note:**
>
> This feature is only supported for Oracle GoldenGate on Windows.

> **Note:**
>
> Install the Microsoft OLE DB Driver for SQL Server software on the Oracle GoldenGate server.
> https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15

1. In SQL Server Management Studio (or other interface) set the `NOT FOR REPLICATION` flag on the following objects. For active-passive configurations, set it only on the passive database. For active-active configurations, set it on both databases.
    - Foreign key constraints

- Check constraints
- `IDENTITY` columns
- Triggers (requires textual changes to the definition; see the SQL Server documentation for more information.)

2. Partition `IDENTITY` values for bidirectional configurations.
3. In the Replicat `MAP` statements, map the source tables to appropriate targets, and map the child tables that the source tables reference with triggers or foreign-key cascade constraints. Triggered and cascaded child operations are replicated by Oracle GoldenGate, so the referenced tables must be mapped to appropriate targets to preserve data integrity. Include the same parent and child source tables in the Extract `TABLE` parameters.

> **Note:**
>
> If referenced tables are omitted from the `MAP` statements, no errors alert you to integrity violations, such as if a row gets inserted into a table that contains a foreign key to a non-replicated table.

4. In the Replicat parameter file, include the `DBOPTIONS` parameter with the `USEREPLICATIONUSER` option.

Creating a Database Connection on Linux

Before creating a database connection for Oracle GoldenGate processes running on Linux, install the latest version of *Microsoft ODBC driver for SQL Server (Linux)*.

Select the following link for download and installation steps:

https://docs.microsoft.com/en-us/sql/connect/odbc/linux-mac/installing-the-microsoft-odbc-driver-for-sql-server?view=sql-server-ver15

For the installation, choose the steps listed under Red Hat Enterprise Linux and Oracle.

After the ODBC software is installed, follow the example below to create an ODBC DSN for Linux:

1. Create a template file for your data source(s):

   ```
   vi odbc_template_file.ini
   ```

2. Describe the data source in the template file. Multiple unique DSN entries can be listed in the template file, if needed.

   In the following example, `mydsn_2019_source` is the DSN name, which will be used with `DBLOGIN` and `SOURCEDB` or `TARGETDB` to connect to the Extract or Replicat to the database.

   ```
   [mydsn_2019_source]
   Driver = ODBC Driver 18 for SQL Server
   Server = myserver,1433
   Database = source_database
   TrustServerCertificate=YES
   ```

3. Install the data source using the following command.

   ```
   odbcinst -i -s -f odbc_template_file.ini
   ```

   This command adds the DSN to the system `odbc.ini` file. For more information, select the following link:

   https://docs.microsoft.com/en-us/sql/connect/odbc/linux-mac/connection-string-keywords-and-data-source-names-dsns?view=sql-server-2017

Creating a Database Connection on Windows

Before creating a database connection for Oracle GoldenGate processes running on Windows, install the latest version of Microsoft ODBC Driver for SQL Server.

Follow these steps to create a system DSN on the Windows server where Oracle GoldenGate is installed.

To create a SQL Server DSN

1. Open the ODBC Data Sources (64-bit) application.
2. In the ODBC Data Source Administrator dialog box, select the **System DSN** tab, and then click **Add**.
3. Under Create New Data Source, select the **ODBC Driver {*version*} for SQL Server** and then click **Finish**. The **Create a New Data Source to SQL Server** wizard appears.
4. Enter the following details, and click **Next**:
   - Name: Can be of your choosing. In a Windows cluster, use one name across all nodes in the cluster.
   - Description: (Optional) Type a description of this data source.
   - Server: Type the SQL Server connection string or server\instance name. For Always On connections, use the **listener\instance name** of the Always On Availability Group.
5. For login authentication, select one of the following options, and then click **Next**:
   a. With Integrated Windows Authentication
   b. With SQL Server authentication using a login ID and password entered by the user
6. Select **Change the default database to**, and then select the source or target database from the list. Enable the **Use ANSI** settings. Click **Next**.
7. Leave the next page set to the defaults. Click **Finish**.
8. Click **Test Data Source** to test the connection.
9. If the test is successful, close the confirmation box and the **Create a New Data Source** box.
10. Repeat this procedure for each SQL Server source and target database.

Preparing Tables for Processing

The table attributes in the following sections must be addressed in your Oracle GoldenGate environment.

Disabling Triggers and Cascade Constraints on the Target

In an environment where SQL Server is the target, consider triggers and cascade constraints that may repeat an operation that occurred on the source. For example, if the source has an insert trigger on TableA that inserts a record into TableB, and Oracle GoldenGate is configured to capture and deliver both TableA and TableB, the insert trigger on the target table, TableA, must be disabled. Otherwise, Replicat inserts into TableA, and the trigger fires and insert into TableB. Replicat will then try to insert into TableB, and then terminate abnormally.

When a trigger or cascade constraint repeats an operation that occurred on the source, you do not have to disable the trigger or constraint when the following conditions are both true:

- You use the `DBOPTIONS USEREPLICATIONUSER` parameter in Replicat.
- You use OLE DB connection for Replicat. The use of the OLE DB connection is the default configuration. Note that the trigger, constraint, or `IDENTITY` property must have `NOT FOR REPLICATION` enabled.

In the following scenario, disable the triggers and constraints on the target:

- Uni-directional replication where all tables on the source are replicated.

In the following scenarios, enable the triggers and constraints on the target:

- Uni-directional replication where tables affected by a trigger or cascade operation are not replicated, and the only application that loads these tables is using a trigger or cascade operation.
- Uni-directional or -bi-directional replication where all tables on the source are replicated. In this scenario, set the target table cascade constraints and triggers to enable `NOT FOR REPLICATION`, and use the `DBOPTIONS USEREPLICATIONUSER` parameter in Replicat.

Ensuring Row Uniqueness in Source and Target Table

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
2. If neither of these key types exist , Oracle GoldenGate constructs a pseudokey of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration. For SQL Server, Oracle GoldenGate requires the row data in target tables that do not have a primary key to be less than 8000 bytes.

> **Note:**
>
> If there are types of keys on a table or if there are no keys at all on a table, Oracle GoldenGate logs a message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Using `KEYCOLS` to Specify a Custom Key

If a table does not have an applicable row identifier, or if you prefer that identifiers are not used, you can define a substitute key, providing that the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key overrides any existing primary or unique key that Oracle GoldenGate finds.

Improving `IDENTITY` Replication with Array Processing

Because only one table per session can have `IDENTITY_INSERT` set to `ON`, Replicat must continuously toggle `IDENTITY_INSERT` when it applies `IDENTITY` data to multiple tables in a session. To improve the performance of Replicat in this situation, use the `BATCHSQL` parameter. `BATCHSQL` causes Replicat to use array processing instead of applying SQL statements one at a time.

Preparing the Database for Oracle GoldenGate — CDC Capture

Learn how to enable supplemental logging in the source database tables that are to be used for capture by the Extract for SQL Server and how to purge older CDC staging data.

You can learn more about CDC Capture with this Oracle By Example:

Using the Oracle GoldenGate for SQL Server CDC Capture Replication http://www.oracle.com/webfolder/technetwork/tutorials/obe/fmw/goldengate/12c/sql_cdcrep/sql_cdcrep.html.

Enabling CDC Supplemental Logging

With the CDC Extract, the method of capturing change data is via SQL Server Change Data Capture tables, so it is imperative that you follow the procedures and requirements below, so that change data is correctly logged, maintained, and captured by Extract.

You will enable supplemental logging with the `ADD TRANDATA` command so that Extract can capture the information that is required to reconstruct transactions.

`ADD TRANDATA` must be issued for all tables that are to be captured by Oracle GoldenGate, and to do so requires that a valid schema be used in order to create the necessary Oracle GoldenGate tables and stored procedures.

Enabling supplemental logging for a CDC Extract does the following:

- Enables SQL Server Change Data Capture at the database level, if it's not already enabled.
  - `EXECUTE sys.sp_cdc_enable_db`
- Creates a Change Data Capture staging table for each base table enabled with supplemental logging by running `EXECUTE sys.sp_cdc_enable_table`, and creates a trigger for each CDC table. The CDC table exists as part of the system tables within the database and has a naming convention like, `cdc.OracleGG_basetableobjectid_CT`.
- Creates a tracking table of naming convention *`schema`*`.OracleGGTranTables`. This table is used to store transaction indicators for the CDC tables, and is populated when the trigger for a CDC table is fired. The table will be owned by the schema listed in the `GLOBALS` file's, `GGSCHEMA` parameter.
- Creates a unique fetch stored procedure for each CDC table, as well as several other stored procedures that are required for Extract to function. These stored procedures will be owned by the schema listed in the `GLOBALS` file's, `GGSCHEMA` parameter.
- Also, as part of enabling CDC for tables, SQL Server creates two jobs per database:

  `cdc.`*`dbname`*`_capture`

  `cdc.`*`dbname`*`_cleanup`
- The CDC Capture job is the job that reads the SQL Server transaction log and populates the data into the CDC tables, and it is from those CDC tables that the Extract will capture the transactions. So it is of extreme importance that the CDC capture job be running at all times. This too requires that SQL Server Agent be set to run at all times and enabled to run automatically when SQL Server starts.
- Important tuning information of the CDC Capture job can be found in CDC Capture Method Operational Considerations.
- The CDC Cleanup job that is created by Microsoft does not have any dependencies on whether the Oracle GoldenGate Extract has captured data in the CDC tables or not. Therefore, extra steps need to be followed in order to disable or delete the CDC cleanup job immediately after `TRANDATA` is enabled, and to enable Oracle GoldenGate's own CDC cleanup job. See Retaining the CDC Table History Data for more information.

The following steps require a database user who is a member of the SQL Server System Administrators (`sysadmin`) role.

1. In the source Oracle GoldenGate installation, ensure that a `GLOBALS` (all CAPS and no extension) file has been created with the parameter `GGSCHEMA` *`schemaname`*. Ensure that the schema name used has been created (`CREATE SCHEMA` *`schemaname`*) in the source

database. This schema will be used by all subsequent Oracle GoldenGate components created in the database, therefore it is recommended to create a unique schema that is solely used by Oracle GoldenGate, such as `'ogg'`. It is recommended not to use the SQL Server schema **cdc** and to create a new schema specific to Oracle GoldenGate.

2. On the source system, run `GGSCI`
3. Issue the following command to log into the database:

   ```
   DBLOGIN SOURCEDB DSN [,{USERID user, PASSWORD password | USERIDALIAS alias}]
   ```

   Where:

   - `SOURCEDB DSN` is the name of the SQL Server data source.
   - `USERID user` is the database login and `PASSWORD password` is the password that is required if the data source connects via SQL Server authentication. Alternatively, `USERIDALIAS alias` is the alias for the credentials if they are stored in a credentials store. If using `DBLOGIN` with a DSN that is using Integrated Windows authentication, the connection to the database for the GGSCI session will be that of the user running GGSCI. In order to issue `ADD TRANDATA` or `DELETE TRANDATA`, this user must be a member of the SQL Server `sysadmin` server role.
4. In GGSCI, issue the following command for each table that is, or will be, in the Extract configuration. You can use a wildcard to specify multiple table names.

   ```
   ADD TRANDATA owner.table
   ```

   ```
   ADD TRANDATA owner.*
   ```

   Optionally, you can designate the filegroup in which the SQL Server Change Data Capture staging tables will be placed, by using the `FILEGROUP` option with an existing filegroup name.

   ```
   ADD TRANDATA owner.table FILEGROUP cdctables
   ```

   See `ADD TRANDATA`

Purging CDC Staging Data

When enabling supplemental logging, data that is required by Extract to reconstruct transactions are stored in a series of SQL Server CDC system tables, as well Oracle GoldenGate objects that are used to track operations within a transaction. And as part of enabling supplemental logging, SQL Server will create its own Change Data Capture Cleanup job that runs nightly by default, and purges data older than 72 hours. The SQL Server CDC Cleanup job is unaware that an Extract may still require data from these CDC system tables and can remove that data before the Extract has a chance to capture it.

If data that Extract needs during processing has been deleted from the CDC system tables, then one of the following corrective actions might be required:

- Alter Extract to capture from a later point in time for which CDC data is available (and accept possible data loss on the target).
- Resynchronize the source and target tables, and then start the Oracle GoldenGate environment over again.

To remedy this situation, Oracle GoldenGate for SQL Server includes Oracle GoldenGate for SQL Server includes the `ogg_cdc_cleanup_setup.bat` program that is used to create an Oracle GoldenGate Cleanup job associated stored procedures and tables.

The Extract, upon startup, will expect, by default, that those Oracle GoldenGate Cleanup task objects exist and will stop if they do not. Extract will issue a warning if the SQL Server CDC Cleanup job exists alongside the Oracle GoldenGate Cleanup job.

The default checks by Extract for the Oracle GoldenGate CDC Cleanup task objects can be overwritten by using the `TRANLOGOPTIONS NOMANAGECDCCLEANUP` in the Extract, but this would only be recommended for development and testing purposes.

Use the following steps immediately after enabling supplemental logging and prior to starting the Extract, to create the Oracle GoldenGate CDC Cleanup job and associated objects. You can re-run these steps to re-enable this feature should any of the objects get manually deleted.

To create the Oracle GoldenGate CDC Cleanup job and objects:

The `ogg_cdc_cleanup_setup` file is located in the the home directory for Classic Architecture.

The script uses the Microsoft `sqlcmd` utility, so ensure that `sqlcmd` is installed on the system where Oracle GoldenGate is installed.

This requires an SQL Server authenticated database user who is a member of the SQL Server System Administrators (`sysadmin`) role. Windows authentication is not supported for the `.bat` script.

1. Stop and disable the database's SQL Server `cdc.`*`dbname`*`_cleanup` job from SQL Server Agent. Alternatively, you can drop it from the source database with the following command.

   ```
   EXECUTE sys.sp_cdc_drop_job 'cleanup'
   ```

2. Run the `ogg_cdc_cleanup_setup.bat` file, providing the following variable values.

   For Windows:

   ```
   ogg_cdc_cleanup_setup.bat createJob userid password databasename
   servername\instancename schema
   ```

   For Linux:

   ```
   ./ogg_cdc_cleanup_setup.sh createJob userid password databasename
   "servername,port" schema
   ```

   In the preceding examples, USER ID and password should be a valid SQL Server login and password for a user, which has `sysadmin` rights. The `databasename`, `servername\instancename`, or `servername`, `port`, are the source database name, server, and instance, or server and TCP/IP port where SQL Server is running. If only the server name is listed, then the default instance will be connected to. The schema is the schema name listed in the GLOBALS file, with the `GGSCHEMA` parameter. This schema should be the same for all Oracle GoldenGate objects, including supplemental logging, checkpoint tables, heartbeat tables, and the Oracle GoldenGate CDC Cleanup job.

   For example:

   ```
   ogg_cdc_cleanup_setup.bat createJob ggsuser ggspword db1 server1\inst1 ogg
   ```

   When using `server,port` in the connection string, enclose the string in double quotes, for example:

   ```
   ogg_cdc_cleanup_setup.bat createJob login password source database
   "sql2016.samplestring.us-west-1.rds.amazonaws.com,1433" OGG schema name
   ```

The Oracle GoldenGate CDC Cleanup job when created, is scheduled to run every ten minutes, with a default retention period of seventy two hours. The job will not purge data for an Extract's recovery checkpoint however, regardless of the retention period.

Additional information of the Oracle GoldenGate CDC Cleanup job can be found in CDC Capture Method Operational Considerations.

Enabling Bi-Directional Loop Detection

Loop detection is a requirement for bi-directional implementations of Oracle GoldenGate, so that an Extract for one source database does not recapture transactions sent by a Replicat from another source database.

With the CDC Extract capture method, by default, any transaction committed by a Replicat into a database where an Extract is configured, will recapture that transaction from the Replicat as long as supplemental logging is enabled for those tables that the Replicat is delivering to.

In order to ignore recapturing transactions that are applied by a Replicat, you must use the `TRANLOGOPTIONS FILTERTABLE` parameter for the CDC Extract. The table used as the filtering table will be the Oracle GoldenGate checkpoint table that you must create for the Replicat.

To create a Filter Table and enable Supplemental Logging:

The steps below require a database user who is a member of the SQL Server System Administrators (`sysadmin`) role.

1. On the source system, run `GGSCI`
2. Issue the following command to log into the database.

   ```
   DBLOGIN SOURCEDB DSN [,{USERID user, PASSWORD password | USERIDALIAS alias}]
   ```

   In the preceding example, the `SOURCEDB DSN` is the name of the SQL Server data source. The `USERID` user is the database login and `PASSWORD` password is the password that is required if the data source connects through SQL Server authentication. Alternatively, `USERIDALIAS` alias is the alias for the credentials if they are stored in a credentials store. If using `DBLOGIN` with a DSN that is using Integrated Windows authentication, the connection to the database for the GGSCI session is that of the user running GGSCI. In order to issue `ADD TRANDATA` or `DELETE TRANDATA`, this user must be a member of the SQL Server `sysadmin` server role.
3. Create the Oracle GoldenGate checkpoint table that is used by the Replicat to deliver data to the source database.

   Example: `ADD CHECKPOINTTABLE ogg.ggchkpt`

   It is recommended that you use the same schema name as used in the `GGSCHEMA` parameter of the `GLOBALS` file.
4. Enable supplemental logging for the newly created checkpoint table.

   Example: `ADD TRANDATA ogg.ggchkpt`
5. Add the Replicat with the checkpoint table information.

   Example: `ADD REPLICAT reptgt1, EXTTRAIL ./dirdat/e2,checkpointtable` *`ogg.ggchkpt`*
6. Configure the Extract with the `IGNOREREPLICATES` (on by default) and `FILTERTABLE` parameters, using the Replicat's checkpoint table for the filtering table.

   ```
   TRANLOGOPTIONS IGNOREREPLICATES
   ```

```
TRANLOGOPTIONS FILTERTABLE ogg.ggchkpt
```

Requirements Summary for Capture and Delivery of Databases in an Always On Availability Group

Oracle GoldenGate for SQL Server supports capture from a primary replica or a read-only, synchronous mode secondary replica of an Always On Availability Group, and delivery to the primary replica.

When capturing from either a primary or a secondary replica in an Always On Availability Group, it is important to understand that the capture process must only read hardened transactions from the log, and that there be no potential for data loss between any replica database that Oracle GoldenGate is or will capture from.

Database Connection

For both Extract and Replicat, it is recommended to create a System DSN that uses the Always On Availability Group Listener for the connection.

- For the Replicat, connecting to the Listener allows the Replicat to reconnect if the primary replica performs a failover to a new instance, without having to manually edit the DSN settings to point to the new primary.
- For the Extract connecting to the Listener not only allows reconnecting to the primary without editing the DSN to point to the new instance, but also provides the optional ability to run the Extract’s data extraction stored procedures, against a read-only secondary.
- For both Extract and Replicat connected to an Always On environment, use the `AUTORESTART` parameter for the Manager, to restart the processes after a failover.
- To route the Extract’s data extraction queries to a read-only secondary, ensure that the DSN connection uses the Listener, that you have one or more read-only secondary replicas that are configured to handle read-only routing, and that the Extract runs with the `TRANLOGOPTIONS ALWAYSONREADONLYROUTING` parameter.
  - Ensure that the Application Intent field of the DSN configuration is set to `READWRITE` and not `READONLY`
  - Refer to the following Microsoft documentation on how to configure read-only routing: https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/configure-read-only-routing-for-an-availability-group-sql-server?view=sql-server-2017

Supplemental Logging

Supplemental logging must be enabled by normal means (`ADD TRANDATA`) using GGSCI connected to the primary replica and not against a secondary replica.

- Create a DSN to the primary replica, or to the Always On Availability Group Listener, to connect using `DBLOGIN` to run `ADD TRANDATA` from GGSCI.
- The login used to enable supplemental logging must have `sysadmin` membership of the primary replica instance.
- When enabling supplemental logging against the primary replica database, the SQL Server Change Data Capture job does not automatically get created on any secondary replicas. Upon failover from a primary to a secondary, you must manually create the SQL Server Change Data Capture job and the Oracle CDC Cleanup job if in use, on the new primary replica.

```
EXECUTE sys.sp_cdc_add_job N'capture
```

– When creating the SQL Server CDC Capture job on the new primary, the default configuration settings are put in place. So if you have previously modified the default values on the former primary replica, you need to run `sys.sp_cdc_change_job` on the new primary and set the values accordingly.

> **Note:**
>
> Consult the Microsoft documentation on how to enable the CDC Capture job for AlwaysOn Secondary Replicas for more information.

Operational Requirements and Considerations

- When an instance is no longer the primary instance but has the SQL Server CDC Capture job installed, the job ceases to run after some time and does not attempt to restart. Upon failover back to that instance, the job does not automatically start, so it must be manually started.
- If secondary replica databases are not in sync with the primary replica database, the CDC capture job will not advance in the log, and therefore no records will be captured by an Extract, until such time that the primary and secondary replicas are synchronized. See this article from Microsoft for more details:

  https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/replicate-track-change-data-capture-always-on-availability?view=sql-server-2017

  > **Note:**
  >
  > When capturing from either a primary or a secondary replica in an Always On Availability Group, it is important to understand that the capture process must only read hardened transactions from the log, and that there be no potential for data loss between any replica database that Oracle GoldenGate is or will capture from.

- When running an Extract from a middle tier Windows or Linux server, set the middle tier server's date, time, and time zone to the same as that of the primary replica.
- Upon failover from a primary to a secondary replica, reinstall the Oracle GoldenGate CDC Cleanup job on the new primary by re-running the `ogg_cdc_cleanup_setup.bat` file with the `createJob` option.
- If Extract is configured to capture from a readable secondary replica, but not configured with read-only routing, the SQL Server CDC Capture job must be created against the secondary replica prior to starting the Extract, as the Extract will check if the job exists. To create the SQL Server CDC Capture job, any potential secondary that will have an Extract connected to it, must at some point be set to a writable Primary database and then follow the steps above, under supplemental logging, to manually add the SQL Server CDC Capture job.
- If uninstalling Oracle GoldenGate and disabling Change Data Capture on a database that is part of an Always On availability group, follow the extra steps provided in Disabling Change Data Capture.

CDC Capture Method Operational Considerations

This section provides information about the SQL Server CDC Capture options, features, and recommended settings.

Tuning SQL Server Change Data Capture

The following information is useful in improving the capture performance of the Extract.

- Ensure that Auto Create Statistics and Auto Update Statistics are enabled for the database. Maintaining statistics on the `cdc.OracleGG_#####_CT` , `cdc.lsn_time_mapping`, and `OracleGGTranTables` table is crucial to the performance and latency of the Extract.
- The SQL Server Change Data Capture job collects data from the SQL Server transaction log and loads it into the Change Data Capture staging tables within the database.

  As part of the job that is created, there are several available tuning parameters that can be used, and information on how to best tune the job can be found in the following article: https://technet.microsoft.com/en-us/library/dd266396(v=sql.100).aspx

  As a general recommendation, you should change the SQL Server Change Data Capture Job polling interval from the default of 5 seconds to 1 second.

  To change the default polling interval of the CDC Capture job, execute the following queries against the database:

```
EXEC [sys].[sp_cdc_change_job]
@job_type = N'capture’,
@pollinginterval = 1,
GO,
--stops cdc job
EXEC [sys].[sp_cdc_stop_job],
@job_type = N'capture’,
GO,
--restarts cdc job for new polling interval to take affect
EXEC [sys].[sp_cdc_start_job],
@job_type = N'capture’,
```

Oracle GoldenGate CDC Object Versioning

Oracle GoldenGate provides a version tracking subsystem to track the CDC objects that are created by Oracle GoldenGate when enabling supplemental logging. These objects are:

- Oracle GoldenGate change tracking tables in the format `OracleGG_`*`object id`*`_CT`.
- Stored procedures in the format *`fetch_database name_object id`*
- Stored procedures `OracleCDCExtract`, `OracleGGCreateProcs`, and `OracleGGCreateNextBatch`.
- After successfully completing the `ADD TRANDATA` command, GGSCI creates a table called `OracleGGVersion` under the `GGSCHEMA` specified in the GLOBALS file, if it does not already exist.

  Next, GGSCI inserts a record into the table that tracks the start and end time of the `TRANDATA` session. When Extract starts up, it checks for consistency between itself and the Oracle GoldenGate CDC objects by comparing its internal version number with the version numbers found in the `OracleGGVersion` table. If it finds that the version numbers do not match, it abends with a message similar to the following:

```
ERROR OGG-05337 The Oracle GoldenGate CDC object versions on database, source,
are not consistent with the expected version, 2. The following versions(s)
were found: 1. Rerun ADD TRANDATA for all tables previously enabled, including
heartbeat, heartbeat seed, and filter tables.
```

Valid and Invalid Extract Parameters for SQL Server Change Data Capture

This section describes parameters used for the CDC Capture method. For more information about supported and unsupported parameters for the CDC Capture method, review *Parameters and Functions Reference for Oracle GoldenGate*.

`TRANLOGOPTIONS LOB_CHUNK_SIZE`

The Extract parameter `LOB_CHUNK_SIZE` is added for the CDC Capture method to support large objects. If you have huge LOB data sizes, then you can adjust the `LOB_CHUNK_SIZE` from the default of 4000 bytes, to a higher value up to 65535 bytes, so that the fetch size is increased, reducing the trips needed to fetch the entire LOB.

Example: `TRANLOGOPTIONS LOB_CHUNK_SIZE 8000`

`TRANLOGOPTIONS MANAGECDCCLEANUP/NOMANAGECDCCLEANUP`

The Extract parameter `MANAGECDCCLEANUP/NOMANAGECDCCLEANUP` is used by the CDC Capture method to instruct the Extract on whether or not to maintain recovery checkpoint data in the Oracle GoldenGate CDC Cleanup job. The default value is `MANAGECDCCLEANUP` and it doesn’t have to be explicitly listed in the Extract. However, it does require creating the Oracle GoldenGate CDC Cleanup job prior to starting the Extract. `MANAGECDCCLEANUP` should be used for all production environments, where `NOMANAGECDCCLEANUP` may be used for temporary and testing implementations as needed.

Example: `TRANLOGOPTIONS MANAGECDCCLEANUP`

`TRANLOGOPTIONS EXCLUDEUSER/EXCLUDETRANS`

The SQL Server CDC Capture job does not capture user information or transaction names associated with a transaction, and as this information is not logged in the CDC staging tables, Extract has no method of excluding DML from a specific user or DML of a specific transaction name. The `EXCLUDEUSER` and `EXCLUDETRANS` parameters are therefore not valid for the CDC Capture process.

`TRANLOGOPTIONS MANAGESECONDARYTRUNCATIONPOINT/NOMANAGESECONDARYTRUNCATIONPOINT/ ACTIVESECONDARYTRUNCATIONPOINT`

The SQL Server Change Data Capture job is the only process that captures data from the transaction log when using the Oracle GoldenGate CDC Capture method. The secondary truncation point management is not handled by the Extract, and for the Change Data Capture Extract, these parameters are not valid.

`TRANLOGOPTIONS ALWAYSONREADONLYROUTING`

The `ALWAYSONREADONLYROUTING` parameter allows Extract for SQL Server to route its read-only processing to an available read-intent Secondary when connected to an Always On availability group listener.

`TRANLOGOPTIONS QUERYTIMEOUT`

Specifies how long queries to SQL Server will wait for results before reporting a timeout error message. This option takes an integer value to represent the number of seconds. The default query timeout value is 300 seconds (5 minutes). The minimum value is 0 seconds (infinite timeout). The maximum is 2147483645 seconds.

`TRANLOGOPTIONS TRANCOUNT`

Allows adjustment of the number of transactions processed per each call by Extract to pull data from the SQL Server change data capture staging tables. Based on your transaction workload, adjusting this value may improve capture rate throughput, although not all workloads will be positively impacted. The minimum value is 1, maximum is 100, and the default is 10.

Details of the Oracle GoldenGate CDC Cleanup Process

The Oracle GoldenGate CDC Cleanup job is required for a CDC Extract by default, since Extract defaults to `TRANLOGOPTIONS MANAGECDCCLEANUP`. It is installed from a Windows batch file (`ogg_cdc_cleanup_setup.bat`), which uses `sqlcmd` to connect to the source SQL Server database and create the necessary objects and job.

There should be one job for each database enabled for CDC Capture, and you must create the job and objects following the steps mentioned in the Preparing the Database for Oracle GoldenGate — CDC Capture section of this document.

Additional options for the utility are discussed in the following sections.

The steps below require a SQL Server authenticated database user who is a member of the SQL Server System Administrators (`sysadmin`) role. Windows authentication is not supported for the `.bat` batch file.

Removing an Extract from the Database

When the Oracle GoldenGate CDC Cleanup object tables exist, each CDC Extract that is started against that database will create an entry in the `OracleGGExtractCheckpoint` table. This entry tracks a particular Extract's point in time recovery checkpoint, which is used as the cutoff LSN for the Oracle GoldenGate CDC cleanup tasks. If there are multiple Extracts running, each logging more recent recovery checkpoints in the table, but one Extract has been removed from the system without removing its entry into the `OracleGGExtractCheckpoint` table, then no data will be purged newer than that deleted Extract's old recovery checkpoint for all of the CDC staging tables. So when deleting an Extract from the database, follow the steps below to remove the Extract from the `OracleGGExtractCheckpoint` table if more than one Extract is running against the database.

1. Log in to the Database with `DBLOGIN` from GGSCI:

   ```
   DBLOGIN SOURCEDB dsn_name USERIDALIAS alias_name
   ```

2. Stop the Extract:

   ```
   STOP EXTRACT extract_name
   ```

3. Delete the Extract:

   ```
   DELETE EXTRACT extract_name
   ```

By logging in to the database, the `DELETE EXTRACT` command removes the entry from the `OracleGGExtractCheckpoint` table, for that specific Extract.

Modifying the Oracle GoldenGate CDC Cleanup Job

The default schedule, retention period and operation batch size for the Oracle GoldenGate CDC Cleanup job of a database is to run every 10 minutes, with a data retention policy of 72 hours (listed as 4320 minutes), purging in batches of 500 records per transaction until the retention policy is meet, not to exceed the recovery checkpoint data of the Extract.

For variations in customer environments, or change data table data retention requirements, it may be necessary to adjust these properties to increase the purge batch size or to adjust retention policies and the job run-time schedule.

To adjust the job execution frequency, manually modify the schedule for the `OracleGGCleanup_dbname_Job` job within SQL Server Agent. If you need to adjust the retention period or purge batch size, you must manually edit the job step for the `OracleGGCleanup_dbname_Job` job within SQL Server Agent. The job step passes two parameters to the cleanup stored procedure, and you can modify the value for `@retention_minutes` to adjust the data retention policy as needed, or modify the `@threshold` value to increase or decrease the purge batch size. In high transactional environments, it may be necessary to increase the `@threshold` value to a number such as 10000. Monitoring the amount of time that it takes for the job to run within each cycle can be used to determine effective `@threshold` values.

Deleting the Oracle GoldenGate CDC Cleanup Job

If you no longer require the Oracle GoldenGate CDC Cleanup job and associated objects and need to remove them, perform the following steps:

1. Open a command prompt and change to the Oracle GoldenGate installation folder.
2. Run the `ogg_cdc_cleanup_setup.bat` file, providing the following variable values:

```
ogg_cdc_cleanup_setup.bat dropJob userid password databasename
servername\instancename schema
```

Example: `ogg_cdc_cleanup_setup.bat dropJob ggsuser ggspword db1 server1\inst1 ogg`

Restoring a Source Database Keeping CDC Data

When restoring a SQL Server database that has been enabled with CDC and you want to keep the existing CDC staging data that has already accumulated in the database, as well as the CDC settings, you must specify the `KEEP_CDC` option with the `RESTORE` statement.

This requirement is only if restoring the database to a new instance, or to the same instance but with a different database name. If you are restoring the original database on the instance, then the option is not required.

For further requirements and understanding, review the following document link from Microsoft:

https://docs.microsoft.com/en-us/sql/t-sql/statements/restore-statements-arguments-transact-sql?view=sql-server-ver15

Additionally, you must manually recreate the CDC Capture job and the Oracle GoldenGate CDC Cleanup job.

Understanding What's Supported for SQL Server

This chapter contains information on database and table features supported by Oracle GoldenGate for SQL Server.

Supported Objects and Operations for SQL Server

The following objects and operations are supported:

- Parallel Replicat is supported with Oracle GoldenGate for SQL Server.
- Oracle GoldenGate supports capture of transactional DML from user tables and delivery to user tables and writeable views.
- `TEXT, NTEXT, IMAGE, VARBINARY, VARBINARY (MAX) VARCHAR (MAX)`, and `NVARCHAR (MAX)` columns are supported in their full size for operations that are logged by SQL Server Chang Data Capture. For example, columns of `IMAGE`, `NTEXT`, and `TEXT` data types are logged as a NULL value for delete operations. For more information, review the Large Object Data Types content at the following Microsoft document:

  https://docs.microsoft.com/en-us/sql/relational-databases/system-tables/cdc-capture-instance-ct-transact-sql?view=sql-server-ver15
- Oracle GoldenGate supports the maximum row sizes that are permitted for tables that are enabled for SQL Server Change Data Capture.
- Oracle GoldenGate supports capture from tables enabled with `PAGE` and `ROW` compression. For partitioned tables that use compression, all partitions must be enabled with the same compression type.
- Oracle GoldenGate supports capture for partitioned tables if the table has the same physical layout across all partitions.
- The sum of all column lengths for a table to be captured from must not exceed the length that SQL Server allows for enabling Change Data Capture for the table. If the sum of all column lengths exceeds what is allowed by the SQL Server procedure `sys.sp.cdc_enable_table`, then `ADD TRANDATA` cannot be added for that table. The maximum allowable record length decreases as more columns are present, so there is an inverse relationship between maximum record length and the number of columns in the table.

Non-Supported Objects and Operations for SQL Server

The following objects and operations are not supported:

- For source databases, operations that are not supported by SQL Server Change Data Capture, such as `TRUNCATE` statements. Refer to Microsoft SQL Server Documentation for a complete list of the operations that are limited by enabling SQL Server Change Data Capture.
- Oracle GoldenGate for SQL Server does not support the capture or delivery of DDL changes for SQL Server and extra steps are required for Oracle GoldenGate processes on the source and target to handle any table level DDL changes, including table index rebuild operations.
- Views are not supported.
- Operations by the `TextCopy` utility and `WRITETEXT` and `UPDATETEXT` statements. These features perform operations that either are not logged by the database or are only partially logged, so they cannot be supported by the Extract process.
- Partitioned tables that have more than one physical layout across partitions.
- Partition switches against a source table. SQL Server Change Data Capture treats partition switches as DDL operations, and the data moved from one partition to another is not logged in the CDC tables, so you must follow the procedures in Requirements for Table

Level DDL Changes to manually implement a partition switch when the table is enabled for supplemental logging.

- Due to a limitation with SQL Server's Change Data Capture, column level collations that are different from the database collation, may cause incorrect data to be written to the CDC tables for character data and Extract will capture them as they are written to the CDC tables. It is recommended that you use `NVARCHAR`, `NCHAR` or `NTEXT` data type for columns containing non-ASCII data or use the same collation for table columns as the database. For more information see, About Change Data Capture (SQL Server).
- Due to a limitation with SQL Server's Change Data Capture, `NOOPUPDATES` are not captured by the SQL Server Change Data Capture agent so there are no records for Extract to capture for no-op update operations.
- Temporal tables are not supported for enabling Change Data Capture, therefore cannot be configured for Extract for source implementations.

Requirements for Table Level DDL Changes

Oracle GoldenGate for SQL Server does not support the capture or delivery of DDL changes. However, beginning with Oracle GoldenGate 21c, changes made to tables enabled with `TRANDATA` will not cause Extract to abend. Extract will continue to process change data for the table as it existed when `TRANDATA` was enabled.

Operations considered to be table-level DDL changes include, but are not limited to: `ALTER TABLE`, `TRUNCATE TABLE`, index rebuilds, and partition switches.

To avoid data inconsistencies due to table level DDL changes, the following steps are required.

1. Source: Pause or Stop application data to the table or tables to be modified.
2. Source: Ensure that there are no open transactions against the table to be modified.
3. Source: Ensure that the SQL Server CDC Capture job processes all remaining transactions for the table that is to be modified.
4. Source: Ensure that the Extract processes all the transactions for the table that is to be modified, prior to making any DDL changes.
5. Target: Ensure that the Replicat processes all the transactions for the table that is to be modified, prior to making any DDL changes.
6. Optionally, implementing an Event Marker table can be used to determine when all of the remaining transactions have been processed for the table that is to be modified, and handle the coordination of when to correctly stop the Extract and Replicat.
7. Source: Stop the Extract process.
8. Target: Stop the Replicat process.
9. Source: Disable supplemental logging for the table to be modified by running `DELETE TRANDATA`.
10. Source: Make table DDL changes to the source table.
11. Target: Make table DDL changes to the target table.
12. Source: Re-enable supplemental logging by running `ADD TRANDATA` to the table(s) after the modifications have been performed.
13. Source: Start the Extract.
14. Target: Start the Replicat.

15. Source: Resume application data to the table or tables that were modified.

Supported SQL Server Data Types

The following data types are supported for capture and delivery, unless specifically noted in the limitations that follow:

- Binary Data Types
  - `(binary, varbinary, varbinary (max))`
  - `(varbinary (max)` with `FILESTREAM)`
- Character Data Types
  - `(char, nchar, nvarchar, nvarchar (max), varchar, varchar (max))`
- Date and Time Data Types
  - `(date, datetime2, datetime, datetimeoffset, smalldatetime, time)`
- Numeric Data Types
  - `(bigint, bit, decimal, float, int, money, numeric, real, smallint, smallmoney, tinyint)`
- LOBs
  - `(image, ntext, text)`
- Other Data Types
  - `(timestamp, uniqueidentifier, hierarchyid, geography, geometry, sql_variant (Delivery only), XML)`
- Oracle GoldenGate for SQL Server can replicate column data that contains `SPARSE` settings..

Limitations:

- Oracle GoldenGate does not support filtering, column mapping, or manipulating large objects larger than 4KB. Full Oracle GoldenGate functionality can be used for objects of up to 4KB.
- Oracle GoldenGate treats XML data as a large object (LOB), as does SQL Server when the XML does not fit into a row. SQL Server extended XML enhancements (such as lax validation, `DATETIME` , union functionality) are not supported.
- A system-assigned `TIMESTAMP` column or a non-materialized computed column cannot be part of a key. A table containing a `TIMESTAMP` column must have a key, which can be a primary key or unique constraint, or a substitute key specified with a `KEYCOLS` clause in the `TABLE` or `MAP` statements. For more information see Assigning Row Identifiers.
- Oracle GoldenGate supports multibyte character data types and multi byte data stored in character columns. Multibyte data is supported only in a like-to-like, SQL Server configuration. Transformation, filtering, and other types of manipulation are not supported for multibyte character data.
- If capture of data for `TEXT`, `NTEXT`, `IMAGE`, `VARCHAR (MAX)`, `NVARCHAR(MAX)` and `VARBINARY (MAX)` columns will exceed the SQL Server default size set for the `max text repl size` option, extend the size. Use `sp_configure` to view the current value of `max text repl size` and adjust the option as needed.

> **Note:**
>
> Amazon RDS for SQL Server does not allow `max text repl size` to be greater than 64MB.

- Columns of `IMAGE`, `NTEXT`, and `TEXT` data types are logged as a `NULL` value for delete and before image update operations. Columns of `VARBINARY(MAX)`, `VARCHAR(MAX)`, and `NVARCHAR(MAX)` are logged as a `NULL` value for before image update operations unless the column was updated.

  For more information, review the Large Object Data Types content in the following Microsoft document:

  https://docs.microsoft.com/en-us/sql/relational-databases/system-tables/cdc-capture-instance-ct-transact-sql?view=sql-server-ver15
- Oracle GoldenGate supports UDT and UDA data of up to 2 GB in size. All UDTs except SQL_Variant are supported.
- Common Language Runtime (CLR), including SQL Server built-in CLR data types (such as, geometry, geography, and hierarchy ID), are supported. CLR data types are supported only in a like-to-like SQL Server configuration. Transformation, filtering, and other types of manipulation are not supported for CLR data.
- `VARBINARY (MAX)` columns with the `FILESTREAM` attribute are supported up to a size of 4 GB. Extract uses standard `Win32` file functions to read the `FILESTREAM` file.
- The range and precision of floating-point numbers depends on the host machine. In general, precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- Oracle GoldenGate supports time stamp data from 0001/01/03:00:00:00 to 9999/12/31:23:59:59. If a time stamp is converted from GMT to local time, these limits also apply to the resulting time stamp. Depending on the time zone, conversion may add or subtract hours, which can cause the time stamp to exceed the lower or upper supported limit.

Limitations on Computed Columns:

- Computed columns, either persisted or non-persisted, are not supported by Microsoft's Change Data Capture. Therefore, no data is written to the trail for columns that contain computed columns. To replicate data for non-persisted computed columns, use the `FETCHCOLS` or `FETCHMODCOLS` option of the `TABLE` parameter to fetch the column data from the table. Keep in mind that there can be discrepancies caused by differences in data values between the time that the column was changed in the data base and the time that Extract fetches the data for the transaction record that is being processed.
- Replicat does not apply DML to any computed column, even if the data for that column is in the trail, because the database does not permit DML on that type of column. Data from a source persisted computed column, or from a fetched non- persisted column, can be applied to a target column that is not a computed column.
- In an initial load, all of the data is selected directly from the source tables, not the transaction log. Therefore, in an initial load, data values for all columns, including non-persisted computed columns, is written to the trail or sent to the target, depending on the method that is used. As when applying change data, however, Replicat does not apply initial load data to computed columns, because the database does not permit DML on that type of column.

- Oracle GoldenGate does not permit a non-persisted computed column to be used in a `KEYCOLS` clause in a `TABLE` or `MAP` statement.
- If a unique key includes a non-persisted computed column and Oracle GoldenGate must use the key, the non-persisted computed column is ignored. This may affect data integrity if the remaining columns do not enforce uniqueness.
- If a unique index is defined on any non-persisted computed columns, it is not used.
- If a unique key or index contains a non-persisted computed column and is the only unique identifier in a table, Oracle GoldenGate must use all of the columns as an identifier to find target rows. Because a non-persisted computed column cannot be used in this identifier, Replicat may apply operations containing this identifier to the wrong target rows.

System Schemas for SQL Server

The following schemas or objects are not be automatically replicated by Oracle GoldenGate unless they are explicitly specified without a wildcard.

- `"sys"`
- `"cdc"`
- `"INFORMATION_SCHEMA"`
- `"guest"`

Non-Supported SQL Server Data Types and Features

- `SQL_Variant` data type is not supported for capture.
- Tables that contain unsupported data types may cause Extract to Abend. As a workaround, you must remove `TRANDATA` from those tables and remove them from the Extract's `TABLE` statement, or use the Extract's `TABLEEXCLUDE` parameter for the table.

Sybase

With Oracle GoldenGate for Sybase database, you can replicate data to and from supported Sybase versions or between a Sybase database and a database of another type. Oracle GoldenGate for Sybase supports data filtering, mapping, and transformation.

Preparing the System for Oracle GoldenGate

This chapter contains the requirements for the system and database resources that support Oracle GoldenGate.
This chapter contains the following sections:

Console Character Set

The operating system and the command console must have the same character sets. Mismatches occur on Microsoft Windows systems, where the operating system is set to one character set, but the DOS command prompt uses a different, older DOS character set. Oracle GoldenGate uses the character set of the operating system to send information to GGSCI command output; therefore a non-matching console character set causes characters not to display correctly. You can set the character set of the console before opening a GGSCI session by using the following DOS command:

```
chcp OS_character_set
```

If the characters do not display correctly after setting the code page, try changing the console font to Lucida Console, which has an extended character set.

Database Configuration

The Extract process makes calls directly to the Sybase Replication API on a source Sybase server. The source database on this server must be configured as follows to support data capture by Oracle GoldenGate.

- Because Extract uses the Sybase LTM to read the Sybase transaction log, it cannot run against a database configured with Sybase Replication Server. Only one process at a time can reserve a context that allows it to read the transaction log on the same database.
- Sybase Multisite availability leverages the Sybase Replication Server to replicate transactions to a Warm Standby and a target subscription database. Oracle GoldenGate for Sybase cannot capture from the primary Warm Standby but can capture from the Multisite availability target subscription database because SAP Sybase Rep Server is not in control of the Transaction Log for that database.
- Set the `DSQUERY` variable to the server that contains the database that Oracle GoldenGate will be using.
- The Extract process must be permitted to manage the secondary log truncation point. For more information, see Initializing the Secondary Truncation Point.
- Configure the database page size to 4k, 8k,16k, 32k, or larger. To use the `UPGRADECHECKPOINT` *`table_name`* command to updated the checkpoint the target Sybase database, it must be configured to have a row size of more than 2K pages to be programmatically created; it will fail to upgrade the checkpoint if the target database is configured to be a 2K page size. This is valid only for Replicat.

Database User for Oracle GoldenGate Processes

Oracle GoldenGate requires a database user account. Create this account and assign privileges according to the following guidelines.

- To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on, or operate as, the Oracle GoldenGate database user.
- Create a database user that is dedicated to Oracle GoldenGate. It can be the same user for all of the Oracle GoldenGate processes that must connect to a database:
  - Extract (source database)
  - Replicat (target database)
  - `DEFGEN` utility (source or target database)
- The Extract process requires permission to access the source database. Do one of the following:
  - Grant System Administrator privileges.
  - Assign a user name with `replication_role` . The command to grant replication role is either:

    ```
    sp_role 'grant', replication_role, Extract_user
    ```

    Or

    ```
    use dbname grant role replication_role to Extract_user
    ```

> **Note:**
>
> Specific DDL or DML operations may require the use of both `sa_role` and `replication_role`.

- The Replicat process requires connect and DML privileges on the target database.

Preparing Tables for Processing

The following table attributes must be addressed in an Oracle GoldenGate environment.

Disabling Triggers and Cascade Constraints

Disable triggers, cascade delete constraints, and cascade update constraints on target Sybase tables, or alter them to ignore changes made by the Oracle GoldenGate database user. Oracle GoldenGate replicates DML that results from a trigger or cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

To configure Replicat to disable target triggers at the start of its database session, take the following steps:

1. Assign the Replicat user the replication role.
2. Add the following parameter statement to the root level of the Replicat parameter file.

   ```
   SQLEXEC "set triggers off"
   ```

Assigning Row Identifiers

Oracle GoldenGate requires some form of unique row identifier on source and target tables to locate the correct target rows for replicated updates and deletes.

Limiting Row Changes in Tables that do not Have a Key

If a target table has no primary key or unique key, duplicate rows can exist. It is possible for Oracle GoldenGate to update or delete too many rows in the target table, causing the source and target data to go out of synchronization without error messages to alert you. To limit the number of rows that are updated, use the `DBOPTIONS` parameter with the `LIMITROWS` option in the Replicat parameter file. `LIMITROWS` can increase the performance of Oracle GoldenGate on the target system because only one row is processed.

Replicating Encrypted Data

Oracle GoldenGate supports columns that are encrypted with a system-encrypted password, but not columns that are encrypted with a user-defined password. Check the tables from which you want to capture data against the following Oracle GoldenGate limitations:

- The table that contains the encrypted columns must have a primary or unique key.
- Columns that use encryption cannot be part of the primary key.

Encrypted columns are encrypted in the data files and in the log, so Extract must be configured to fetch the clear-text values from the database. To trigger this fetch, use the `FETCHCOLS` and `FETCHMODCOLS[EXCEPT]` options of the Extract `TABLE` parameter. `FETCHCOLS` forces a fetch of values that are not in the log, and `FETCHMODCOLS` or `FETCHMODCOLS[EXCEPT]` forces a fetch of values that are in the logs. Used together, these parameters ensure that the encrypted columns are always fetched from the database.

The following is an example of how to configure Extract to support the encryption. In this example, the encrypted column is `cardnum`.

```
TABLE ab.payments, FETCHCOLS (cardnum), FETCHMODCOLS (cardnum);
```

Preparing the Transaction Logs

To capture DML operations, Oracle GoldenGate reads the online logs. To ensure the continuity and integrity of Oracle GoldenGate processing, the logs must be configured as directed in the following sections:

Initializing the Secondary Truncation Point

Establish a secondary log truncation point on the source system prior to running the Oracle GoldenGate Extract process. Extract uses the secondary truncation point to identify data that remains to be processed.

To initialize the secondary truncation point, log on to the database as a user with `sa_role` privileges and then issue the following command:

```
dbcc settrunc( 'ltm', valid )
```

By default, Extract will manage the secondary truncation point once it is established. Do not permit Extract to be stopped any longer than necessary; otherwise the log could eventually fill up and the database will halt. The only way to resolve this problem is to disable the secondary truncation point and manage it outside of Oracle GoldenGate, and then purge the transaction log. Data not yet processed by Extract will be lost, and you will have to resynchronize the source and target data.

To control how the secondary truncation point is managed, use the `TRANLOGOPTIONS` parameter. For more information, see Reference for Oracle GoldenGate for Windows and UNIX.

Sizing and Retaining the Logs

Retain enough log data on the source system so that Extract can start again from its checkpoints after you stop it or there is an unplanned outage. Extract must have access to the log that contains the start of the oldest uncommitted unit of work, and all logs thereafter. To determine where the Extract checkpoints are, use the `INFO EXTRACT` command. For more information about `INFO EXTRACT`, see Reference for Oracle GoldenGate for Windows and UNIX .

If data that Extract needs during processing is not retained, either in online or backup logs, one of the following corrective actions might be required:

- You might need to alter Extract to capture from a later point in time for which log data is available (and accept possible data loss on the target).
- You might need to resynchronize the source and target tables, and then start the Oracle GoldenGate environment over again.

Make certain not to use backup or archive options that cause old archive files to be overwritten by new backups on the source system. New backups should be separate files with different names from older ones. This ensures that if Extract looks for a particular log, it will still exist, and it also ensures that the data is available in case it is needed for a support case.

Enabling Transaction Logging

Use the `ADD TRANDATA` command to mark each source table for replication. This command uses the Sybase `sp_setreptable` and `sp_setrepcol` system procedures. `ADD TRANDATA` is the recommended way to mark the tables, instead of using those procedures through the database interface, but the owner or the system administrator can use them if needed. For more information, see the Sybase documentation.

To mark tables for replication with ADD TRANDATA:

1. On the source system, run GGSCI from the Oracle GoldenGate directory.
2. Log into the database from GGSCI.

   ```
   DBLOGIN SOURCEDB database USERID user PASSWORD xxx
   ```

   Where:

   - *`database`* is the name of the database.
   - *`user`* is the database owner or the system administrator. You will be prompted for the password. This command has encryption options for the password. For more information, see Reference for Oracle GoldenGate for Windows and UNIX.
   - *`xxx`* is the password for the associated *user*.

3. Issue the `ADD TRANDATA` command for each table to be marked.

   ```
   ADD TRANDATA SCHEMA.TABLE LOBSNEVER | LOBSALWAYS | LOBSALWAYSNOINDEX | LOBSIFCHANGED
   ```

   Where:

   - `LOBSNEVER | LOBSALWAYS | LOBSALWAYSNOINDEX | LOBSIFCHANGED` control whether LOB data is never propagated, only propagated if changed (the default), or always propagated. The `ADD TRANDATA` command will overwrite the LOB replication setting that is currently set for the table.

> **Note:**
>
> Some `ADD TRANDATA` options enable the `ALWAYS_REPLICATE` option of `sp_setrepcol`. If a `LOB` column contains a `NULL` value, and then another column in the table gets updated (but not the `LOB`), that `LOB` will not be captured even though `ALWAYS_REPLICATE` is enabled.

Understanding What's Supported for Sybase

This chapter contains information on database and table features supported by Oracle GoldenGate for Sybase.

Supported Sybase Data Types

This section lists the Sybase data types that Oracle GoldenGate supports and any limitations of this support.

Integers

- `BIGINT`
- `BIT`
- `DECIMAL`
- `INT` (signed and unsigned)
- `TINYINT` (signed and unsigned)
- `NUMERIC`
- `SMALLINT` (signed and unsigned)

Limitations of Support

- `NUMERIC` and `DECIMAL` (fixed-point) are supported with no integrity loss when moving data to a target column of the same data type without involving calculations or transformation. When calculations or transformation must be performed, Oracle GoldenGate supports a maximum value of a signed long integer (32-bits).
- `BIT` is supported for automatic mapping between Sybase databases. To move `BIT` data between Sybase and another database type, Oracle GoldenGate treats `BIT` data as binary. In this case, the following are required:
  - The `BIT` column must be mapped to the corresponding source or target column with a `COLMAP` clause in a `TABLE` or `MAP` statement.
- For the Sybase 157 GA release, these data types cannot be replicated:
  - `BIGINT` (as a key column)
  - `BIGDATETIME`
  - `BIGTIME`
- When replicating `TINYINT` and Extract is not in the same version of Replicat, you will need to create a `sourcedef` and/or `targetdef` file even if you are replicating between identical Sybase versions.
- See also Non-Supported Sybase Data Types.

Floating-Point Numbers

- `DOUBLE`
- `FLOAT`
- `REAL`

Limitations of Support

The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.

Character Data

- `CHAR`
- `NCHAR`
- `NVARCHAR`
- `VARCHAR`
- `UNICHAR`
- `UNIVARCHAR`

Limitations of Support

- These data types are supported to the maximum length supported by the database, this being the maximum page size.
- Fetching `NVARCHAR` replication results using the Sybase `char_length` or `datalength` functions when a Sybase database is the target and the source is a Non-Oracle database and you replicate from the source to the target may result in a data integrity issue. This occurs when you use a Sybase release earlier than Adaptive Server Enterprise 15.5 for Windows x64 platform EBF 21262: 15.5 ESD #5.3.

Dates and Timestamps

- `BIGDATETIME`
- `BIGTIME`
- `DATE`
- `DATETIME`
- `SMALLDATETIME`
- `TIME`

Limitations of Support

- Oracle GoldenGate supports timestamp data from 0001/01/03:00:00:00 to 9999/12/31:23:59:59. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the time zone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.
- Oracle GoldenGate does not support negative dates.

Large Objects

- `BINARY`
- `IMAGE`
- `TEXT`

- `UNITEXT`
- `VARBINARY`

Limitations of Support

- `TEXT`, `UNITEXT` and `IMAGE` are supported up to 2 GB in length.
- Large objects that are replicated from other databases (such as Oracle `BLOB` and `CLOB`) can be mapped to Sybase `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns. To prevent Replicat from abending if the replicated large object is bigger than the size of the target column, use the `DBOPTIONS` parameter with the `ALLOWLOBDATATRUNCATE` option in the Replicat parameter file. For more information, see Reference for Oracle GoldenGate for Windows and UNIX.
- To move data to a Sybase target from a source database that permits empty `LOB` columns, use the `DBOPTIONS` parameter with the `EMPTYLOBSTRING` option in the Replicat parameter file. This parameter accepts a string value and prevents Replicat from setting the target column to `NULL`, which is not permitted by Sybase. For more information, see Reference for Oracle GoldenGate for Windows and UNIX.
- When a source table contains multiple identical rows, it can cause `LOB` inconsistencies in the target table. This occurs when the source table lacks a primary key or other unique row identifier. The rows are inserted by Replicat on the target, but if the `LOB` data is updated in a subsequent source operation, it will only be replicated to the first row that was inserted on the target.
- Do not use `NOT NULL` constraints on the in-row LOB column. If you want to use `NOT NULL` constraints, use them on the off-row LOB column.
- If you need to fetch the in-row LOB data directly from the table you must use `FETCHCOLS/FETCHMODCOLS`.
- Oracle GoldenGate for Sybase 15.7 does not support the in-row LOB column replication (however, it can still push the data into the in-row LOB column at in the Replicat database). This means tables included in the replication cannot have any in-row LOB columns. Oracle GoldenGate will abend if any replication table includes an in-row LOB column. If you need in-row LOB support, contact Oracle Support for further information.

Money Types

- `MONEY`
- `SMALLMONEY`

Limitations of Support

Money data types are supported with no integrity loss when moving data to a target column of the same data type without involving calculations or transformation. When calculations or transformation must be performed, Oracle GoldenGate supports a maximum value of a signed long integer (32-bits).

IDENTITY Type

The `IDENTITY` data type is supported for replication in one direction only, but not for a bi-directional configuration.

text, image, and unitext Data Types

With the Sybase 15.7 version, the LOB text, image, and unitext data types are now supported in `BATCHSQL` mode. The data length of the LOB is confined to 4K. If the records that contain

LOB columns and the size exceeds more than 4K, then those records are excluded from the batches and are executed one at a time. The LOB columns in are now bound, while in previous Sybase version (15.5 or 15.0) the LOBs were not bound. You can use thee older behavior by using the `DBPOTIONS LEGACYLOBREPLICATION` parameter. This support is only applicable to Replicat running on Sybase version 15.7 and later.

User-Defined Data Types

User-defined data types are fully supported.

Non-Supported Sybase Data Types

This section lists the Sybase data types that Oracle GoldenGate does not support.

- The `TIMESTAMP` data is not supported. Timestamp columns data is captured though the data cannot be applied to the Sybase timestamp column due to a database limitation. The database populates this column automatically once that corresponding row is inserted or updated. To exclude timestamp columns from being captured by Oracle GoldenGate, use the `COLSEXCEPT` option of the `TABLE` parameter. Because the system generates the timestamps, the source and target values will be different.
- The Java `rowobject` data type is not supported.

Supported Operations and Objects for Sybase

This section lists the data operations and database objects that Oracle GoldenGate supports.

- The extraction and replication of insert, update, and delete operations on Sybase tables that contain rows of up to 512 KB in length.
- The maximum number of columns and the maximum column size per table that is supported by the database.
- Deferred inserts, deferred indirect inserts, deferred updates, and deferred deletes. It is possible that the use of deferred updates could cause primary key constraint violations for the affected SQL on the target. If these errors occur, use the Replicat parameter `HANDLECOLLISIONS` .
- `TRUNCATE TABLE` if the names of the affected tables are unique across all schemas. If the table names are not unique across all schemas, use the `IGNORETRUNCATES` parameter for those tables to prevent Replicat from abending.
- `GETTRUNCATES` and `IGNORETRUNCATES` by Extract and Replicat.
- Data that is encrypted with a system-encrypted password.
- Array fetching during initial loads, as controlled by the `FETCHBATCHSIZE` parameter.
- The `BATCHSQL` Replicat feature on ASE 15.7 SP110 and later on the following platforms:
  - AIX
  - Linux x64
  - Sun Solaris SPARC
  - Sun Solaris x64
  - Windows x64

  In certain scenarios, the `CS_NUMERIC` and `CS_DECIMAL` data types are not supported by `BatchSQL` because of a bug in the Sybase specific CT Library. LOB replication is supported in BatchSql mode for Sybase database version 157 SP110 onward. This will improve the

LOB replication performance. It is restricted to 16384 bytes of LOB data that means if LOB data is more than 16384 bytes, the data would not be processed through `BATCHSQL` mode instead the mode switched to Normal.

- Limitations on Computed Columns support are as follows:
  - Fully supports persisted computed columns. The change values are present in the transaction log and can be captured to the trail.
  - You cannot use `NOT NULL` constraints on in-row LOB columns. If you need to use `NOT NULL` constraints, do so only with off-row LOB columns.
  - Tables with non-persisted computed columns, but does not capture change data for these columns because the database does not write it to the transaction log. To replicate data for non-persisted computed columns, use the `FETCHCOLS` or `FETCHMODCOLS` option of the `TABLE` parameter to fetch the column data from the table. Keep in mind that there can be discrepancies caused by differences in data values between when the column was changed in the database and when Extract fetches the data for the transaction record that is being processed.
  - Replicat does not apply DML to any computed column, even if the data for that column is in the trail, because the database does not permit DML on that type of column. Data from a source persisted computed column, or from a fetched non-persisted column, can be applied to a target column that is not a computed column.
  - In an initial load, all of the data is selected directly from the source tables, not the transaction log. Therefore, in an initial load, data values for all columns, including non-persisted computed columns, gets written to the trail or sent to the target, depending on the method that is being used. As when applying change data, however, Replicat does not apply initial load data to computed columns, because the database does not permit DML on that type of column.
  - Persisted computed column that is defined as a key column, an index column, or that is part of a `KEYCOLS` clause in a `TABLE` or `MAP` statement are not used. If a unique key or index includes a computed column and Oracle GoldenGate must use that key, the computed column will be ignored. Additionally, if a unique key or index contains a computed column and is the only unique identifier on the table, all of the columns are used except the computed column as an identifier to find the target row. Thus, the presence of a computed column in a key or index affects data integrity if the remaining columns do not enforce uniqueness. Sybase does not support non-persisted computed columns as part of a key so neither does Oracle GoldenGate.
  - To support `TRUNCATE TABLE`, all table names should be unique across all schemas within a database. This rule applies to Extract and Replicat.
- Limitations on Automatic Heartbeat Table support are as follows:
  - Heartbeat frequency should an integer that is divisible by 60. Oracle GoldenGate heartbeat parameter frequency is accepted in minutes, although you can use in seconds. The Sybase job scheduler uses the minutes in integer not in decimal so it is converted internally to set the frequency in minutes to the nearest possible value. For example, setting this value to 65 seconds results in the frequency being set to 1 minute; 140 seconds results in the value set to 2 minutes.
  - Data truncation occurs with a Replicat abend when it exceeds more than 1500 characters for the `incoming_routing_path` and `outgoing_routing_path` of the `GG_HEARTBEAT_SEED`, `GG_HEARTBEAT`, and `GG_HEARTBEAT_HISTORY` tables. The `incoming_routing_path` and `outgoing_routing_path` size of these table is set to 1500 characters in ASCII and is a 500 max bytes in multibyte characters. Ensure that the incoming and outgoing routing path strings are within the specified limit.

  - Sybase job scheduler must be configured on the ASE server prior to running Oracle GoldenGate heartbeat functionality.
  - For heartbeat table functionality to operate correctly, the login user must have the `replication_role`, `js_admin_role`, `js_user_role` roles.

Non-Supported Operations and Objects for Sybase

This section lists the data operations and database objects that Oracle GoldenGate does not support.

- Data that is encrypted with a user-defined password.
- Extraction or replication of DDL (data definition language) operations.
- Multi-Extract configuration. Only one Extract can reserve a context to read the Sybase transaction logs.
- Because `SHOWSYNTAX` is supported in the `DYNSQL` mode, `NODYNSQL` is deprecated.
- Table names that contain data with an underscore followed by some characters then a space (for example, `'zzz_j'`) is not supported. Oracle GoldenGate cannot process records containing this type of character string with GGSCI, `DEFGEN`, `EXTRACT`, or `REPLICAT`. Additionally, this type of data cannot be used with Oracle GoldenGate wildcard (*). If you do have this type of data in your table name, you must drop this kind of table name from your database, and then they restart the application to process and respect Oracle GoldenGate wildcard.

Teradata

With Oracle GoldenGate for Teradata, you can deliver initial load and transactional data from other supported Oracle GoldenGate sources, such as an Oracle database.

Oracle GoldenGate for Teradata supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

Supported Platforms for a Replication Server

In a Teradata environment, you install Oracle GoldenGate on a server that is separate from the one where the Teradata target databases are installed. This machine will be the replication server and must be a platform that is supported by Oracle GoldenGate for the Teradata database.

Preparing the System for Oracle GoldenGate

This chapter contains guidelines for preparing the database and the system to support Oracle GoldenGate. This chapter contains the following sections:

Preparing Tables for Processing

The following table attributes must be addressed in an Oracle GoldenGate environment.

Disabling Triggers and Cascade Constraints

Disable triggers, cascade delete constraints, and cascade update constraints on target Teradata tables. Oracle GoldenGate replicates DML that results from a trigger or cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

   > **Note:**
   >
   > If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

How Oracle GoldenGate Determines the Kind of Row Identifier to Use

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key (required for tables of a Standard Edition instance).
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.

3. If neither of these key types exist , Oracle GoldenGate constructs a pseudokey of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are types of keys on a table or if there are no keys at all on a table, Oracle GoldenGate logs a message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

Using KEYCOLS to Specify a Custom Key

If a table does not have an applicable row identifier, or if you prefer that identifiers are not used, you can define a substitute key, providing that the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key overrides any existing primary or unique key that Oracle GoldenGate finds.

ODBC Configuration for Teradata

Configure ODBC on each target system including the creation of a data source name (DSN). A DSN stores information about how to connect to the database. See the *ODBC Driver for Teradata User Guide* for complete information and setup steps:

https://docs.teradata.com/search/books?filters=prodname~%2522ODBC+Driver+for+Teradata%2522&content-lang=en-US

Database User for Oracle GoldenGate Processes for Teradata

Follow these requirements for the database user for Oracle GoldenGate processes:

- Create a database user that is dedicated to Oracle GoldenGate. It can be the same user for all of the Oracle GoldenGate processes that must connect to a database:
  - Replicat (target database)
  - The `DEFGEN` utility (target database)
- To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, the Oracle GoldenGate database user.
- For Oracle GoldenGate to replicate to a target Teradata database, grant `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on all of the target tables to the Replicat database user.

Configuring Oracle GoldenGate

Learn about the prerequisites and tasks for configuring Oracle GoldenGate Replicat for Teradata database.

Creating a Checkpoint Table

Replicat maintains its checkpoints in a checkpoint table in the Teradata target database (the database where you use `DBLOGIN`). Each checkpoint is written to the checkpoint table within the

Replicat transaction. Because a checkpoint either succeeds or fails with the transaction, Replicat ensures that a transaction is only applied once, even if there is a failure of the process or the database.

Use the following GGSCI command on the target system, to create the Replicat checkpoint table.

```
ggsci> ADD CHECKPOINTTABLE schema.chkptabl Successfully created checkpoint table
schema.chkptabl
```

For more information about creating a checkpoint table, see .

Configuring Oracle GoldenGate Replicat

This section highlights the basic Replicat parameters that are required for most target database types. Additional parameters may be required, see the Oracle GoldenGate installation and configuration documentation for your target database and the *Reference for Oracle GoldenGate*.

Perform these steps on the target replication server or target database system.

1. Configure the Manager process according to the instructions in *Administering Oracle GoldenGate*.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.
3. Create a Replicat checkpoint table. There are multiple options for this purpose, see *Administering Oracle GoldenGate*.
4. Create a Replicat group. For documentation purposes, this group is called `rep`.

   ```
   ADD REPLICAT rep, EXTTRAIL remote_trail, CHECKPOINTTABLE owner.table
   ```

   Use the `EXTTRAIL` argument to link the Replicat group to the remote trail that you specified for the data pump on the source server.
5. Use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the parameters shown in #unique_434/unique_434_Connect_42_BABHIJAI plus any others that apply to your database environment.

Example 3-2 Parameters for the Replicat Group

```
-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB target_dsn_name,] [USERID user id[, PASSWORD pw]]
-- Specify tables for delivery:
MAP owner.source_table, TARGET owner.target_table;
```

Additional Oracle GoldenGate Configuration Guidelines

The following are additional considerations to make once you have installed and configured your Oracle GoldenGate environment.

Handling Massive Update and Delete Operations

Operations that update or delete a large number of rows will generate discrete updates and deletes for each row on the subscriber database. This could cause a lock manager overflow on the Teradata subscriber system, and thus terminate the Replicat process.

To avoid these errors, temporarily suspend replication for these operations and then perform them manually on the source and target systems. To suspend replication, use the following command, which suspends replication for that session only. The operations of other sessions on that table are replicated normally.

```
set session override replication on;

commit;
```

Preventing Multiple Connections

By default, the Replicat processes create a new connection for catalog queries. You can prevent this extra connection by using the `DBOPTIONS` parameter with the `NOCATALOGCONNECT` option.

Performing Initial Synchronization

Perform an initial synchronization of the source and target data before using Oracle GoldenGate to transmit transactional changes for the first time to configure an initial load, see .

Common Maintenance Tasks

This chapter contains instructions for performing some common maintenance tasks when using the Oracle GoldenGate replication solution.

Modifying Columns of a Table

To modify columns of a table:

1. Suspend activity on the source database for all tables that are linked to Oracle GoldenGate.
2. Start GGSCI.
3. In GGSCI, issue this command for the Replicat group:

   ```
   INFO REPLICAT group
   ```

4. On the `Checkpoint Lag` line, verify whether there is any Replicat lag. If needed, continue to issue `INFO REPLICAT` until lag is zero, which indicates that all of the data in the trail has been processed.
5. Stop the Replicat group.

   ```
   STOP REPLICAT group
   ```

6. Perform the table modifications on the target databases.
7. Start the Replicat process.

   ```
   START REPLICAT group
   ```

8. Allow user activity to resume on all of the source tables that are linked to Oracle GoldenGate.

Understanding What's Supported for Teradata

This chapter contains information on database and table features supported by Oracle GoldenGate.

Supported Teradata Data Types

The following table shows the Teradata data types that Oracle GoldenGate supports. Any limitations or conditions that apply follow this table.

Data type	v15.x	v16.x
`BLOB`	Yes	Yes
`BYTEINT`	Yes	Yes
`VARBYTE`	Yes	Yes
`BIGINT`	Yes	Yes
`BYTEINT`	Yes	Yes
`DATE`	Yes	Yes
`DECIMAL - 18 and under`	Yes	Yes
`DECIMAL - 19 to 38`	Yes	Yes
`DOUBLE PRECISION`	Yes	Yes
`FLOAT`	Yes	Yes
`INTEGER`	Yes	Yes
`NUMERIC - 18 and under`	Yes	Yes
`NUMERIC - 19 to 38`	Yes	Yes
`REAL`	Yes	Yes
`SMALLIINT`	Yes	Yes
`TIME`	Yes	Yes
`TIMESTAMP`	Yes	Yes
`INTERVAL`	Yes	Yes
`INTERVAL DAY`	Yes	Yes
`INTERVAL DAY TO HOUR`	Yes	Yes
`INTERVAL DAY TO MINUTE`	Yes	Yes
`INTERVAL DAY TO SECOND`	Yes	Yes
`INTERVAL HOUR`	Yes	Yes
`INTERVAL HOUR TO MINUTE`	Yes	Yes
`INTERVAL HOUR TO SECOND`	Yes	Yes
`INTERVAL MINUTE`	Yes	Yes
`INTERVAL MINUTE TO SECOND`	Yes	Yes
`INTERVAL MONTH`	Yes	Yes
`INTERVAL SECOND`	Yes	Yes
`INTERVAL YEAR`	Yes	Yes
`INTERVAL YEAR TO MONTH`	Yes	Yes
`CHAR`	Yes	Yes
`CLOB`	Yes	Yes
`CHAR VARYING`	Yes	Yes

Data type	v15.x	v16.x
`LONG VARCHAR`	Yes	Yes
`VARCHAR`	Yes	Yes
`GRAPHIC`	Yes	Yes
`LONG VARGRAPHIC`	Yes	Yes
`VARGRAPHIC`	Yes	Yes
`PERIOD (DATE)`	Yes	Yes
`PERIOD (TIME)`	Yes	Yes
`PERIOD (TIMESTAMP)`	Yes	Yes
`UDT`	Yes	Yes

Limitations of Support for Numeric Data Types

When replicating these data types from a different type of database to Teradata, truncation can occur if the source database supports a higher precision that Teradata does.

The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.

Limitations of Support for Single-byte Character Data Types

Single-byte character types are fully supported within a single-byte Latin character set between other databases and Teradata. A `VARCHAR` or `CHAR` column cannot have more than 32k-1 bytes. If using UTF-16, this is 16k-2 characters.

Conditions and Limitations of Support for Multi-byte Character Data

Conditions and limitations of support for multi-byte character data are as follows:

- Install Oracle GoldenGate on a Windows or Linux replication server.
- Use the Teradata ODBC driver version 12.0.0.x or later.
- Do not use filtering, mapping, and transformation for multi-byte data types.
- A `CHAR` or `VARCHAR` column cannot contain more than 32k-1 bytes. If using UTF-16, these columns cannot contain more than 16k-2 characters.
- Set the ODBC driver to the UTF-16 character set in the initialization file.
- When creating Replicat groups, use the `NODBCHECKPOINT` option with the `ADD REPLICAT` command. The Replicat database checkpointing feature does not support an ODBC driver that is set to the UTF-16 character set. Checkpoints will be maintained in the checkpoint file on disk.

Limitations of Support for Binary Data Types

No limitations. These data types are supported between other source databases and Teradata targets.

Limitations of Support for Large Object Data Types

The following are limitations of support for large object data types.

- To replicate large objects from other databases to Teradata, use Teradata ODBC driver version 12.0 or higher on the target system. The target must support large objects that are delivered by ODBC.
- Enable the `UseNativeLOBSupport` flag in the ODBC configuration file. See the Teradata ODBC documentation.

Limitations of Support for Date Data Types

The following are limitations of support for date data types:

- `DATE`, `TIME`, and `TIMESTAMP` are fully supported when replicated from a different type of source database to Teradata.
- `TIME with TIMESZONE`, `TIMESTAMP with TIMEZONE`, and `INTERVAL` are not supported from a different type of source database to Teradata.
- Oracle GoldenGate supports timestamp data from 0001/01/03:00:00:00 to 9999/12/31:23:59:59. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the timezone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.
- Oracle GoldenGate does not support negative dates.

Limitations of Support for IDENTITY Data Types

`IDENTITY` must be configured as `GENERATED BY DEFAULT AS IDENTITY` on the target to enable the correct value to be inserted by Replicat.

Supported Objects and Operations for Teradata

This section lists the data operations and database objects that Oracle GoldenGate supports.

- Oracle GoldenGate supports the maximum number of columns per table that is supported by the database.
- Truncating operations are supported with the use of the `GETTRUNCATES` parameter with Oracle GoldenGate 12.2.*x* and greater.
- Limitations on Automatic Heartbeat Table support are as follows:
  - The `ALTER HEARTBEATTABLE` command is not supported and if used is ignored.
  - The `ADD HEARTBEATTABLE` command with the `FREQUENCY`, `PURGE_FREQUENCY`, or `RETENTION_TIME` option is not supported. When any of these options are specified with the `ADD HEARTBEATTABLE` command, a warning is displayed that the option is ignored.
  - Since Teradata does not have any internal event/job schedulers, automatic purging of heartbeat history data does not occur. You need to explicitly delete or truncate records periodically from the heartbeat history table.

Non-Supported Operations for Teradata

This section lists the data operations that Oracle GoldenGate does not support.

- Extract (capture)

- DDL

TimesTen

With Oracle GoldenGate for Oracle TimesTen, you can deliver initial load and transactional data from other supported Oracle GoldenGate sources, such as an Oracle database.

Oracle GoldenGate for Oracle TimesTen supports data filtering, mapping, and transformations unless noted otherwise in this documentation.

Database Requirements

This section describes the database requirements for using Oracle GoldenGate for Oracle TimesTen.

Database User for Oracle GoldenGate Processes for Teradata

Follow these requirements for the database user for Oracle GoldenGate processes:

- Create a database user that is dedicated to Oracle GoldenGate. It can be the same user for all of the Oracle GoldenGate processes that must connect to a database:
  - Replicat (target database)
  - The `DEFGEN` utility (target database)
- To preserve the security of your data, and to monitor Oracle GoldenGate processing accurately, do not permit other users, applications, or processes to log on as, or operate as, the Oracle GoldenGate database user.
- For Oracle GoldenGate to replicate to a target Teradata database, grant `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on all of the target tables to the Replicat database user.

Preparing the System for Oracle GoldenGate

This chapter contains guidelines for preparing the system to support Oracle GoldenGate:

Setting the Environment Variables

Ensure that the required system environment variables are sourced before proceeding. The correct environment settings are needed for all sessions or processes that will interact with Oracle TimesTen. Every Oracle TimesTen instance (Server and Client) contains a script for setting the required environment variables. This script is located in `*instance_home_dir*/bin` and is named `ttenv.[c]sh`. It should always be dotted or sourced and never executed directly.

Example of setting the bash shell environment for a TimesTen instance homed in `/instancepath/tt181`:

```
source /instancepath/tt181/bin/ttenv.sh
```

Although it's possible to set the required environment variables manually, it is not recommended. Using the script:

- Ensures that all the necessary environment variables (there are several) are correctly set.
- Insulates you from the introduction of new variables in future Oracle TimesTen releases.

Configuring the TimesTen ODBC Connectivity

Oracle GoldenGate for TimesTen connects to TimesTen using the ODBC API (TimesTen's native API). ODBC connectivity defines the concept of a Data Source Name (DSN). A DSN is a logical name which applications use to specify the parameters to be used for connecting to a target database.

When using Oracle GoldenGate for TimesTen, you will specify the DSN of the target TimesTen database in various Oracle GoldenGate configuration settings, such as the `SOURCEDB` clause of the `DBLOGIN` command. For example:

```
DBLOGIN SOURCEDB database, USERIDALIAS useralias
```

Here, the value given for `database` will be the DSN of the target TimesTen database.

When using the Direct mode connectivity, connections must reference a *`server DSN`* defined in the `sys.odbc.ini` file of the Oracle TimesTen instance that hosts the database (the server instance).

When using the Client-Server mode, connections must reference a *`client DSN`* defined in the `sys.odbc.ini` file of either the Oracle TimesTen instance that manages the database (the server instance) or, more commonly, in the `sys.odbc.ini` of an Oracle TimesTen client instance, such as an Oracle GoldenGate hub server.

For information on defining Oracle TimesTen server and client DSNs, refer to TimesTen In-Memory Database Operations Guide.

Here is an example of the `sys.odbc.ini` entries to define a client DSN (`myttdbcs`) that connects to a database identified by the server DSN `myttdb` located on the host `tthost1.mydomain.com`. The TimesTen server's default listener port on that host is 6625.

```
[ODBC Data Sources]
myttdbcs=TimesTen 18.1 Client Driver
[myttdbcs]
TTC_SERVER=tthost1.mydomain.com/6625
TTC_SERVER_DSN=myttdb
ConnectionCharacterSet=AL32UTF8
```

Configuring ODBC on Linux

The following steps provide minimum settings required to connect Oracle GoldenGate processes. For more detailed information on the Oracle TimesTen Client and Server configuration and information, review the following Oracle TimesTen documentation:

https://docs.oracle.com/database/timesten-18.1/TTOPR/client_server.htm#TTOPR177

1. Edit the `$TIMESTEN_HOME/conf/sys.odbc.ini` file.

   ```
   vi $TIMESTEN_HOME/conf/sys.odbc.ini
   ```

2. Describe the data source in the template file. In the following example, `TTCS_181` is used as the client name for which `DBLOGIN` and `SOURCEDB` and `TARGETDB` are used to connect to the database.

   ```
   [ODBC Data Sources]
       TTCS_181=TimesTen 18.1 Client Driver
   ```

3. Set a logical server name for `TTC_SERVER` and use the server DSN value for the `TTC_SERVER_DSN` entry. The value of the `TTC_SERVER_DSN` must match the database specific server DSN that exists in the database server's `sys.odbc.ini` file.

```
[TTCS_181]
TTC_SERVER=ttRemoteDBServer_TT_181
TTC_SERVER_DSN=ttDatabaseDSN
```

4. Edit the `$TIMESTEN_HOME/conf/sys.ttconnect.ini` file with the settings described in these steps.

```
vi $TIMESTEN_HOME/conf/sys.ttconnect.ini
```

5. Create an entry of the same logical server name that was used as the value of `TTC_SERVER` within the `sys.odbc.ini` file. In this example, `ttRemoteDBServer_TT_181` is the entry name. Include an optional description value, and the required `Network_Address` and `TCP_PORT` values that point to the Oracle TimesTen database server and port.

```
[ttRemoteDBServer_TT_181]
Description=TimesTen
ServerNetwork_Address=server.company.com
TCP_PORT=6625
```

6. From the Oracle GoldenGate directory on the target, verify the connection settings by running GGSCI and issuing the `DBLOGIN` command to log into the target database.

```
DBLOGIN SOURCEDB database, USERID db_user [, PASSWORD pw [encryption
options]]
```

In this example:

- `SOURCEDB` database specifies the new Data Source Name.
- `USERID db_user`, `PASSWORD pw` are the Replicat database user profile and password.
- `encryption options` is optional password encryption

Preparing Tables for Processing

This section describes the table attributes you must address in an Oracle GoldenGate environment with TimesTen.

Disabling Triggers and Cascade Constraints

Disable triggers, cascade delete constraints, and cascade update constraints on the target tables, or alter them to ignore changes made by the Oracle GoldenGate database user. Oracle GoldenGate replicates DML that results from a trigger or cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.

4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

Ensuring Row Uniqueness for Tables

Oracle GoldenGate requires some form of unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority:

1. Primary key
2. First unique key alphanumerically that does not contain a timestamp or non-materialized computed column.
3. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding those that are not supported by Oracle GoldenGate in a key or those that are excluded from the Oracle GoldenGate configuration.

> **Note:**
>
> If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

4. If a table does not have an appropriate key, or if you prefer that the existing key(s) are not used, you can define a substitute key, if the table has columns that always contain unique values. You define this substitute key by including a KEYCOLS clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. See TABLE | MAP in *Parameters and Functions Reference for Oracle GoldenGate*.

Configuring Oracle GoldenGate

Learn about the prerequisites and tasks for configuring Oracle GoldenGate Replicat for Teradata database.

Configuring Oracle GoldenGate Replicat

This section describes the Replicat parameters that are required for most target database types. For additional parameters that may be required, see the Oracle GoldenGate installation and configuration documentation for your target database and the *Parameters and Functions Reference for Oracle GoldenGate*.

Perform these steps on the target replication server or target database.

1. Configure the Manager process according to the instructions in *Administering Oracle GoldenGate*.

2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.
3. Create a Replicat checkpoint table. There are multiple options for this purpose, see *Administering Oracle GoldenGate*.

   ```
   DBLOGIN SOURCEDB myttdbcs USERIDLIAS useralias
   ADD CHECKPOINTTABLE owner.oggcheckpointtable
   ```

4. Create a Replicat. For documentation purposes, this Replicat is called `reptt`.

   ```
   ADD REPLICAT reptt, EXTTRAIL ./dirdat/remote_trail, CHECKPOINTTABLE
   owner.oggcheckpointtable
   ```

   Use the `EXTTRAIL` argument to link the Replicat to the remote trail that you specified with the Data Pump Extract on the source Oracle GoldenGate installation.
5. Use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the parameters shown below plus any others that apply to your database environment.

Example 3-3 Parameters for the Replicat Group

```
REPLICAT reptt
-- Specify database login information as needed for the database:
TARGETDB myttdbcs, USERIDALIAS useralias
-- Specify tables for delivery:
MAP owner.sourcetable, TARGET owner.targettable;
```

Additional Oracle GoldenGate Configuration Guidelines

The following are additional considerations to make once you have installed and configured your Oracle GoldenGate environment.

Performing Initial Synchronization

Perform an initial synchronization of the source and target data before using Oracle GoldenGate to transmit transactional changes for the first time to configure an initial load. See Initial Synchronization in *Administering Oracle GoldenGate*..

Understanding What's Supported for Oracle TimesTen

This chapter contains information on database and table features supported by Oracle GoldenGate.

Supported Objects and Operations for TimesTen

The following objects and operations are supported:

- Oracle GoldenGate for Oracle TimesTen supports delivery of transactional DML to user tables.
- `INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE` operations are supported.

Non-Supported TimesTen Data Types and Features

The `INTERVAL` and `ROWID` data types are not supported.

Supported TimesTen Data Types

The following data types are supported for delivery, unless specifically noted in the limitations that follow:

- Binary Data Types

  `(binary, varbinary)`

- Character Data Types

  `(char, nchar, nvarchar2, varchar2)`

- Date and Time Data Types

  `(date, time, timestamp, tt_date, tt_time, tt_timestamp)`

- Numeric Data Types

  `(binary_float, binary_double, double, float, tt_bigint, tt_integer, number, real, tt_smallint, tt_tinyint)`

- LOB

  `(blob, clob, nclob)`

Limitations and Non-supported Items for Oracle TimesTen

The limitations and non-supported items for Oracle TimesTen are:

- Capture (extraction) of DML operations is not supported.
- Capture and replication of DDL (data definition language) operations is not supported.
- The support of range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- Oracle GoldenGate supports timestamp data from 0001/01/03:00:00:00 to 9999/12/31:23:59:59. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the time zone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.
- Modifying the Primary Key Column value is not supported.
- Limitations on Automatic Heartbeat Table support are as follows:
  - Oracle GoldenGate supports only Delivery on TimesTen so no mechanism is required to populate/update the heartbeat tables using any event/job schedulers.
  - The `ALTER HEARTBEATTABLE` command is not supported and if used is ignored.
  - The `ADD HEARTBEATTABLE` command with the `FREQUENCY`, `PURGE_FREQUENCY`, or `RETENTION_TIME` option is not supported. When any of these options are specified with the `ADD HEARTBEATTABLE` command, a warning is displayed that the option is ignored.

  - Since TimesTen does not have any internal event/job schedulers, automatic purging of heartbeat history tables *cannot* occur. As such, you should explicitly drop or truncate the corresponding heartbeat objects to suit your environment.

System Requirements and Preinstallation Instructions

This chapter contains the requirements for the system and database resources that support Oracle GoldenGate.

Supported Database Architectures

Oracle GoldenGate for Oracle TimesTen supports the Classic and Scaleout architectures of the TimesTen database.

Supported Platforms and Database Versions

Oracle TimesTen supports installing Oracle GoldenGate on Linux.

For supported platform and database version information, review the certification matrix:

https://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html.

Oracle TimesTen Software Installation

The Oracle TimesTen Client needs to be installed on the server where Oracle GoldenGate is going to be installed. If Oracle GoldenGate is installed on the Oracle TimesTen database server, then the required components are already available. However, if you are installing Oracle GoldenGate on a hub server, then you must separately install the Oracle TimesTen Client.

In both cases you will need to configure the ODBC connection information.

For Linux platforms there is only one TimesTen software distribution that provides both server and client components. To download the Oracle TimesTen Software, visit:

https://www.oracle.com/database/technologies/timesten-downloads.html

Before beginning to install Oracle GoldenGate with Oracle TimesTen, you must also set the `LD_LIBRARY_PATH` variable:

1. Download the *TimesTen Scaleout and TimesTen Classic/Cache 18.x for Linux x86 (64-bit)* build.
2. Extract the Oracle TimesTen installation files to the designated location, based on the instructions provided in *Oracle TimesTen In-Memory Database Installation Guide*.
3. Set the `LD_LIBARY_PATH` system variable to include the TimesTen installation's `lib` directory. This system variable must be set to install and run Oracle GoldenGate. Example:

```
export LD_LIBRARY_PATH=/installpath/tt18.1.2.2.0/lib:$LD_LIBRARY_PATH
```

Client-only Instance Creation

For non-database server environments where you plan to install Oracle GoldenGate, after installing the Oracle TimesTen client libraries, follow the TimesTen document instructions to create a client-only instance of TimesTen.

1. Perform the following:

   ```
   [oracle@tt_installation_dir]$ ./tt18.1.2.1.0/bin/ttInstanceCreate -
   clientonly
   ```

2. Follow the instance installation prompts, taking note of where the TimesTen instance is installed. This information will be required when setting up a Replicat's ODBC connection to TimesTen.
3. Set the `TIMESTEN_HOME` system variable to the TimesTen instance path.

   Example:

   ```
   export TIMESTEN_HOME=/instancepath/tt181
   ```

Operating System Privileges

The operating system privileges for using Oracle GoldenGate for Oracle TimesTen are:

- You need read and write privileges on the Oracle GoldenGate installation directory.
- Oracle GoldenGate Replicat and Manager processes must operate as an operating system user that has privileges to read, write, and delete files and subdirectories in the Oracle GoldenGate directory. In addition, the Manager process requires privileges to control all other Oracle GoldenGate processes.
- Dedicate the Replicat and Manager operating system users to Oracle GoldenGate to avoid access to sensitive information to other users who run Oracle GoldenGate processes.

Prepare Oracle GoldenGate Classic Architecture for Data Replication

Learn about the tasks for preparing Oracle GoldenGate Classic Architecture before setting up Oracle GoldenGate processes.

Oracle GoldenGate Security Privileges

This section outlines the security privileges that Oracle GoldenGate requires on a source DB2 for i system and on a Windows or Linux target system.

Oracle GoldenGate Security Privileges

This section outlines the security privileges that Oracle GoldenGate requires on a source DB2 for i system and on a Windows or Linux target system.

The person who installs Oracle GoldenGate must have read and write privileges on the Oracle GoldenGate installation directory, because steps will be performed to create some sub-folders and run some programs. On a Windows systen, the person who installs Oracle GoldenGate must log in as Administrator.

Manager, Replicat, and Collector (program name is `server`) are active. Manager controls the other processes and interacts with Collector to receive incoming data, while Replicat applies data to the target DB2 for i database through ODBC.

Oracle GoldenGate processes must be assigned a user account that is dedicated to Oracle GoldenGate and cannot be used by any other program. One user account can be used by all of the Oracle GoldenGate processes. This account must have privileges to read, write, and delete files and directories within the Oracle GoldenGate installation directory.

If the Extract user profile does not have the required authority, Extract will log the following errors and stop.

```
[SC=-1224:SQL1224N A database agent could not be started to service a
request, or was terminated as a result of a database system shutdown or a
force command.SQL STATE 55032: The CONNECT statement is invalid, because the
database manager was stopped after this application was started]
```

The user profile must be specified with the `USERID` parameter when you configure the parameter files and in the `DBLOGIN` command prior to issuing any GGSCI commands that interact with the database.

For more information on user profiles and security privileges, see User Profiles and Security Privileges.

Oracle GoldenGate Security Privileges on a DB2 for i System

The Oracle GoldenGate processes must be assigned a user profile account that is dedicated to Oracle GoldenGate and cannot be used by any other program. One user profile can be used by all of the Oracle GoldenGate processes. This profile need only be granted permission to the objects that Oracle GoldenGate will be operating upon. If specific change data is not to be seen by Oracle GoldenGate, do not include it in any of the journals that the Oracle GoldenGate user profile is allowed to access.

The Manager process must have privileges to control all other Oracle GoldenGate processes (DB2 for i `*JOBCTL` authority).

Assign `*USE` authority to all objects on the system that the Extract user profile must have access to. Assign `*CHANGE` authority to all objects on the system that the Replicat user profile must have access to. This can be accomplished by either granting `*ALLOBJ` authority to the user, or by setting the individual authority to the objects (`FILE`, `LIBRARY` and `JOURNAL` objects) that the user must access. This includes the objects in the `QSYS2` library where the SQL catalog resides. These authorities must be granted through the native DB2 for i interface through a 5250 terminal session or through the DB2 for i Operations Navigator product available from IBM.

The Extract and Replicat database user profiles must be specified with the `USERID` parameter when you configure the parameter files and in the `DBLOGIN` command prior to issuing any GGSCI commands that interact with the database.

The Oracle GoldenGate user profile that runs the Extract process needs to have the `*USE` authority on the `QSYS/QPMLPMGT` service program.

Initializing the Transaction Logs

When you initialize a transaction log, you must ensure that all of the data is processed by Oracle GoldenGate first, and then you must delete and re-add the Extract group and its associated trail.

1. Stop the application from accessing the database. This stops more transaction data from being logged.
2. Connect the Admin Client command line using the `CONNECT` command.

3. Issue the `SEND EXTRACT` command with the `LOGEND` option for the primary Extract group. This command queries Extract to determine whether or not Extract is finished processing the records that remain in the transaction log.

```
SEND EXTRACT group LOGEND
```

4. Continue issuing the command until it returns a `YES` status, indicating that there are no more records to process.
5. On the target system, issue the `SEND REPLICAT` command with the `STATUS` option. This command queries Replicat to determine whether or not it is finished processing the data that remains in the trail.

```
SEND REPLICAT group STATUS
```

6. Continue issuing the command until it shows 0 records in the current transaction, for example:

```
Sending STATUS request to REPLICAT REPSTAB...
Current status:
  Seqno 0, Rba 9035
  0 records in current transaction.
```

7. Stop the primary Extract group, and the Replicat group.

```
STOP EXTRACT group
STOP REPLICAT group
```

8. Delete the Extract and Replicat groups.

```
DELETE EXTRACT group
DELETE REPLICAT group
```

9. Using standard operating system commands, delete the trail files.
10. Stop the database.
11. Initialize and restart the database.
12. Recreate the primary Extract group.

```
ADD EXTRACT group TRANLOG, BEGIN NOW
```

13. Recreate the local trail (if used).

```
ADD EXTTRAIL trail, EXTRACT group
```

14. Recreate the remote trail.

```
ADD RMTTRAIL trail, EXTRACT group
```

15. Recreate the Replicat group.

```
ADD REPLICAT group, EXTTRAIL trail
```

16. Start Extract and Replicat.

```
START EXTRACT group
START REPLICAT group
```

Creating a Checkpoint Table

Replicat maintains checkpoints that provide a known position in the trail from which to start after an expected or unexpected shutdown. To store a record of its checkpoints, Replicat uses a checkpoint table in the target database. This enables the Replicat checkpoint to be included within the Replicat transaction itself, to ensure that a transaction will only be applied once, even if there is a failure of the Replicat process or the database process. The checkpoint table remains small because rows are deleted when no longer needed, and it does not affect database performance. See About Checkpoints for more information about the checkpoint table.

Options for Creating the Checkpoint Table

The checkpoint table can reside in a schema of your choice. Use one that is dedicated to Oracle GoldenGate if possible.

More than one instance of Oracle GoldenGate (multiple installations) can use the same checkpoint table. Oracle GoldenGate keeps track of the checkpoints, even if Replicat group names are the same in different instances.

More than one checkpoint table can be used as needed. For example, you can use different ones for different Replicat groups.

You can install your checkpoint tables in these ways:

- You can specify a default checkpoint table in the `GLOBALS` file. New Replicat groups created with the `ADD REPLICAT` command will use this table automatically, without requiring any special instructions.
- You can provide specific checkpoint table instructions when you create any given Replicat group with the `ADD REPLICAT` command:
  - To use a specific checkpoint table for a group, use the `CHECKPOINTTABLE` argument of `ADD REPLICAT`. This checkpoint table overrides any default specification in the `GLOBALS` file. If using only one Replicat group, you can use this command and skip creating the `GLOBALS` file altogether.
  - To omit using a checkpoint table for a group, use the `NODBCHECKPOINT` argument of `ADD REPLICAT`. Without a checkpoint table, Replicat still maintains checkpoints in a checkpoint file on disk, but you introduce the risk of data inconsistency.

However you implement the checkpoint table, you must create it in the target database prior to using the `ADD REPLICAT` command.

To Add a Checkpoint Table to the Target Database

The following steps, which create the checkpoint table through GGSCI, can be bypassed by running the `chkpt_db_create.sql` script instead, where *db* is an abbreviation of the database type. By using the script, you can specify custom storage or other attributes. Do not change the names or attributes of the columns in this table.

1. From the Oracle GoldenGate directory, run GGSCI and issue the `DBLOGIN` command to log into the database. The user issuing this command must have `CREATE TABLE` permissions.

See *Parameters and Functions Reference for Oracle GoldenGate* for the correct syntax to use for your database.

2. In GGSCI, issue the following command to add the checkpoint table to the database.

   ```
   ADD CHECKPOINTTABLE container owner.table
   ```

   Where:

   `owner.table` is the owner and name of the table, *`container`* is the name of a PDB if installing into an Oracle multitenant container database. The owner and name can be omitted if you are using this table as the default checkpoint table and this table is specified with `CHECKPOINTTABLE` in the `GLOBALS` file. The name of this table must not exceed the maximum length permitted by the database for object names. The checkpoint table name cannot contain any special characters, such as quotes, backslash, pound sign, and so forth.

To Specify a Default Checkpoint Table in the GLOBALS File

This procedure specifies a global name for all checkpoint tables in the Oracle GoldenGate instance. You can override this name for any given Replicat group by specifying a different checkpoint table when you create the Replicat group.

1. Create a `GLOBALS` file (or edit the existing one, if applicable). The file name must be all capital letters on UNIX or Linux systems, without a file extension, and must reside in the root Oracle GoldenGate directory. You can use an ASCII text editor to create the file, making certain to observe the preceding naming conventions, or you can use GGSCI to create and save it with the correct name and location automatically. When using GGSCI, use the following command, typing `GLOBALS` in upper case.

   ```
   EDIT PARAMS ./GLOBALS
   ```

2. Enter the following parameter:

   ```
   CHECKPOINTTABLE container.owner.table
   ```

   Where:

   `catalog.owner.table` is the fully qualified name of the default checkpoint table, including the name of the container if the database is an Oracle multitenant container database (CDB).

3. Note the name of the table, then save and close the `GLOBALS` file. Make certain the file was created in the root Oracle GoldenGate directory. If there is a file extension, remove it.

Adjusting for Coordinated Replicat in Oracle RAC

If the Replicat for which you are creating a checkpoint table will run in an Oracle RAC configuration, it is recommended that you increase the `PCTFREE` attribute of the Replicat checkpoint table to as high a value as possible, as high as 90 if possible. This accommodates the more frequent checkpointing that is inherent in coordinated processing. This change must be made before starting the Replicat group for the first time. See Creating an Online Replicat Group for more information about coordinated Replicat.

Specifying the DB2 LUW Database in Parameter Files

For an Oracle GoldenGate process to connect to the correct DB2 LUW database, you must specify the name (not an alias) of the DB2 LUW database with the following parameters:

- Specify the DB2 source database with the Extract parameter `SOURCEDB`.

- Specify the DB2 target database name with the Replicat parameter `TARGETDB`.

For more information about these parameters, see the Reference for Oracle GoldenGate for Windows and UNIX.

4

Manage

Learn about configuring the Manager process and specify ports for local and remote network communications.

Overview of the Manager Process

To configure and run Oracle GoldenGate, a Manager process must be running on all Oracle GoldenGate source and target systems, and any intermediary systems if used in your configuration. Manager is the controller process that instantiates the Oracle GoldenGate processes, allocates port numbers, and performs file maintenance. Together, the Manager process and its child processes, and their related programs and files comprise an Oracle GoldenGate instance. The Manager process performs the following functions:

- Starts Oracle GoldenGate processes
- Starts dynamic processes
- Starts the Collector process
- Manages the port numbers for processes. (All Oracle GoldenGate ports are configurable.)
- Performs trail management
- Creates event, error, and threshold reports

There is one Manager per Oracle GoldenGate installation. One Manager can support multiple Oracle GoldenGate extraction and replication processes.

Configure Network Communications

Configuring the Manager process and ports for local and remote network communications.

Assigning Manager a Port for Local Communication

The Manager process in each Oracle GoldenGate installation requires a dedicated port for communication between itself and other local Oracle GoldenGate processes. To specify this port, use the `PORT` parameter in the Manager parameter file. Follow these guidelines:

- The default port number for Manager is 7809. You must specify either the default port number (recommended, if available) or a different one of your choice.
- The port must be unreserved and unrestricted.
- If more than one instance of Oracle GoldenGate exists on the system, then each Manager must use a different port number.

See `PORT` in *Parameters and Functions Reference for Oracle GoldenGate*for more information.

Maintaining Ports for Remote Connections through Firewalls

If a firewall is being used at an Oracle GoldenGate target location, additional ports are required on the target system to receive dynamic TCP/IP communications from remote Oracle GoldenGate processes. These ports are:

- One port for *each* Collector process that is started by the local Manager to receive propagated transaction data from remote online Extract processes. When an Extract process sends data to a target, the Manager on the target starts a dedicated Collector process.
- One port for *each* Replicat process that is started by the local Manager as part of a remote task. A remote task is used for initial loads and is specified with the `RMTTASK` parameter. This port is used to receive incoming requests from the remote Extract process. See `RMTTASK` in *Parameters and Functions Reference for Oracle GoldenGate*for more information.
- Some extra ports in case they are needed for expansion of the local Oracle GoldenGate configuration.
- Ports for the other Oracle GoldenGate products if they interact with the local Oracle GoldenGate instance, as stated in the documentation of those products.

To specify these ports, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Follow these guidelines:

- You can specify up to 5000 ports in any combination of the following formats:

  ```
  7830, 7833, 7835
  7830-7835
  7830-7835, 7839
  ```
- The ports must be unreserved and unrestricted.
- If more than one instance of Oracle GoldenGate exists on the system, then each Manager must use a different port number.

Although not a required parameter, `DYNAMICPORTLIST` is strongly recommended for best performance. The Collector process is responsible for finding and binding to an available port, and having a known list of qualified ports speeds this process. In the absence of `DYNAMICPORTLIST` (or if not enough ports are specified with it), Collector can use a port range between 7819 and 12818. If Collector runs out of ports in the `DYNAMICPORTLIST` list, the following occurs:

- Manager reports an error in its process report and in the Oracle GoldenGate `ggserr` log.
- Collector retries based on the rules in the Oracle GoldenGate `tcperrs` file. For more information about the `tcperrs` file, see Handling TCP/IP Errors.

See `DYNAMICPORTLIST` in *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Choosing an Internet Protocol

By default, Oracle GoldenGate selects a socket in the following order of priority to ensure the best chance of connection success:

- IPv6 dual-stack
- IPv4 if IPv6 dual-stack is not available

- IPv6

If your network has IPv6 network devices that do not support dual-stack mode, you can use the `USEIPV6` parameter to force Oracle GoldenGate to use IPv6 for all connections. This is a `GLOBALS` parameter that applies to all processes of an Oracle GoldenGate instance. When `USEIPV6` is used, the entire network must be IPv6 compatible to avoid connection failures. See `USEIPV6` in *Parameters and Functions Reference for Oracle GoldenGate*for more information.

Creating the Manager Parameter File

To configure Manager with required port information and optional parameters, create a parameter file by following these steps. See Using Oracle GoldenGate Parameter Files for more information about Oracle GoldenGate parameter files.

> **Note:**
>
> If Oracle GoldenGate resides in a cluster, configure the Manager process within the cluster application as directed by the vendor's documentation, so that Oracle GoldenGate fails over properly with other applications. For more information about installing Oracle GoldenGate in a cluster, see the Oracle GoldenGate documentation for your database.

1. From the Oracle GoldenGate directory, run the GGSCI program to open the Oracle GoldenGate Software Command Interface (GGSCI).
2. In GGSCI, issue the following command to edit the Manager parameter file.

   ```
   EDIT PARAMS MGR
   ```

3. Add the parameters that you want to use for the Manager process, each on one line.
4. Save, then close the file.

Example 4-1 Sample manager file on a UNIX system

```
PORT 7809
DYNAMICPORTLIST 7810-7820, 7830
AUTOSTART ER t*
AUTORESTART ER t*, RETRIES 4, WAITMINUTES 4
STARTUPVALIDATIONDELAY 5
USERIDALIAS mgr1
PURGEOLDEXTRACTS /ogg/dirdat/tt*, USECHECKPOINTS, MINKEEPHOURS 2
```

The following is a sample Manager parameter file on a UNIX system using required and recommended parameters.

Using the Recommended Manager Parameters

The following parameters are optional, but recommended, for the Manager process.

- `AUTOSTART`: Starts Extract and Replicat processes when Manager starts. This parameter is required in a cluster configuration, and is useful when Oracle GoldenGate activities must begin immediately at system startup. (Requires Manager to be part of the startup routine.) You can use multiple `AUTOSTART` statements in the same parameter file. See `AUTOSTART`in *Parameters and Functions Reference for Oracle GoldenGate*for more information.

- `AUTORESTART`: Starts Extract and Replicat processes again after abnormal termination. This parameter is required in a cluster configuration, but is also useful in any configuration to ensure continued processing. See `AUTORESTART`in *Parameters and Functions Reference for Oracle GoldenGate*for more information.
- `PURGEOLDEXTRACTS`: Purges trail files when Oracle GoldenGate is finished processing them. Without `PURGEOLDEXTRACTS`, no purging is performed and trail files can consume significant disk space. For best results, use `PURGEOLDEXTRACTS` as a Manager parameter, not as an Extract or Replicat parameter. See `PURGEOLDEXTRACTS`in *Parameters and Functions Reference for Oracle GoldenGate*for more information.
- `STARTUPVALIDATIONDELAY` | `STARTUPVALIDATIONDELAYCSECS`: Sets a delay time after which Manager validates the run status of a process. Startup validation makes Oracle GoldenGate users aware of processes that fail before they can generate an error message or process report. See `STARTUPVALIDATIONDELAYCSECS`in *Parameters and Functions Reference for Oracle GoldenGate*for more information.

Controlling Manager

Learn about the steps to start and stop the Manager process.

Starting Manager

Manager must be running before you start other Oracle GoldenGate processes. You can start Manager from:

- The command line of the operating system.
- The GGSCI command interface.
- The Services applet on a Windows system if Manager is installed as a service. See the Windows documentation or your system administrator.
- The Cluster Administrator tool if the system is part of a Windows cluster. This is the recommended way to bring the Manager resource online. See the cluster documentation or your system administrator.
- The cluster software of a UNIX or Linux cluster. Refer to the documentation provided by the cluster vendor to determine whether to start Manager from the cluster or by using GGSCI or the command line of the operating system.

> **Note:**
>
> When starting Manager from the command line or GGSCI with User Account Control enabled, you will receive a UAC prompt requesting you to allow or deny the program to run.

Starting Manager from the Command Shell of the Operating System

To start Manager from the command shell of the operating system, issue the following command.

```
mgr paramfile parameter_file [reportfile report_file]
```

The `reportfile` argument is optional and can be used to store the Manager process report in a location other than the default of the `dirrpt` directory in the Oracle GoldenGate installation location.

Starting Manager from GGSCI

To start Manager from GGSCI, run GGSCI from the Oracle GoldenGate directory, and then issue the following command.

```
START MANAGER
```

> **Note:**
>
> When starting Manager from the command line or GGSCI with User Account Control enabled, you will receive a UAC prompt requesting you to allow or deny the program to run.

Stopping Manager

Manager runs indefinitely or until stopped by a user. In general, Manager should remain running when there are synchronization activities being performed. Manager performs important monitoring and maintenance functions, and processes cannot be started unless Manager is running.

Stopping Manager on UNIX and Linux

On UNIX and Linux (including USS on z/OS), Manager must be stopped by using the `STOP MANAGER` command in GGSCI.

```
STOP MANAGER [!]
```

Where:

`!` stops Manager without user confirmation.

In a UNIX or Linux cluster, refer to the documentation provided by the cluster vendor to determine whether Manager should be stopped from the cluster or by using GGSCI.

Stopping Manager on Windows

On Windows, you can stop Manager from the Services applet (if Manager is installed as a service). See the Windows documentation or your system administrator.

If Manager is not installed as a service, you can stop it with the `STOP MANAGER` command in GGSCI.

```
STOP MANAGER [!]
```

In a Windows cluster, you must take the Manager resource offline from the Cluster Administrator. If you attempt to stop Manager from the GGSCI interface, the cluster monitor interprets it as a resource failure and attempts to bring the resource online again. Multiple start requests through GGSCI eventually will exceed the start threshold of the Manager cluster resource, and the cluster monitor will mark the Manager resource as failed.

5

Extract

Learn about different types of Extract and how to add and manage Extracts.

About Extract

Extract is a process that is configured to run against the source database or configured to run on a downstream mining database (Oracle only) with capturing data generated in the true source database located somewhere else. This process is the extraction or the data capture mechanism of Oracle GoldenGate.

You can configure an Extract for the following use cases:

- **Initial Loads**: When you set up Oracle GoldenGate for initial loads, the Extract process captures the current, static set of data directly from the source objects. See #unique_500
- **Change Synchronization**: When you set up Oracle GoldenGate to keep the source data synchronized with another set of data, the Extract process captures the DML and DDL operations performed on the configured objects after the initial synchronization has taken place. Extracts can run locally on the same server as the database or on another server using the downstream integrated Extract (in case of Oracle database) for reduced overhead. It stores these operations until it receives commit records or rollbacks for the transactions that contain them. If it receives a rollback, it discards the operations for that transaction. If it receives a commit, it persists the transaction to disk in a series of files called a trail, where it is queued for propagation to the target system. All the operations in each transaction are written to the trail as a sequentially organized transaction unit and are in the order in which they were committed to the database (commit sequence order). This design ensures both speed and data integrity.

> **Note:**
>
> Extract ignores operations on objects that are not in the Extract configuration, even though a transaction may also include operations on objects that are in the Extract configuration.

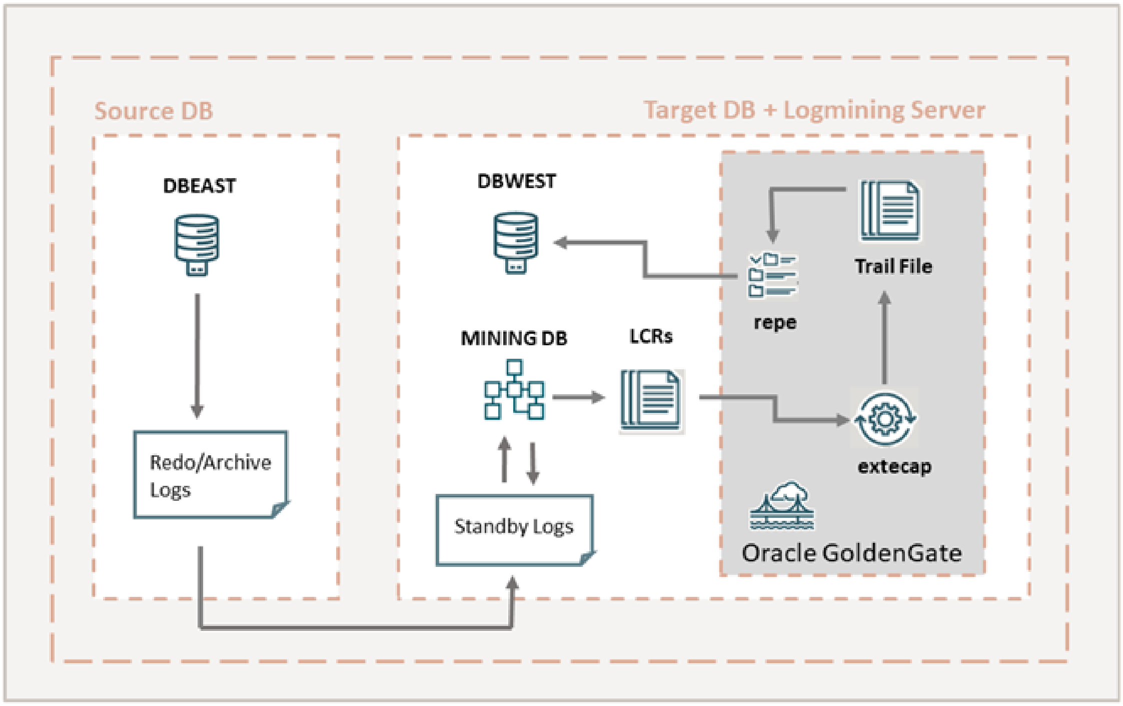
The Extract process can be configured to capture data from the following types of data sources:

- **Source tables**: This source type is used for initial loads.
- **Database recovery logs or transaction logs**: While capturing from the logs, the actual method varies depending on the database type. An example of this source type is the Oracle database redo logs, which are used for supplemental logging.

About Integrated Extract

The Oracle GoldenGate Integrated Extract process interacts directly with a database logmining server to receive data changes in the form of logical change records (LCRs).

The following diagram illustrates the configuration of Extract.



Some of the additional features of Oracle GoldenGate Extract are:

- Extract is fully integrated with the database, allowing seamless interoperability between features such as Oracle RAC, ASM, and TDE.
- Extract uses the database logmining server to access the Oracle redo stream, with the benefit of being able to automatically switch between different copies of archive logs or different mirrored versions of the online logs. Thus, capture can transparently handle the absence of a log file caused by disk corruption, hardware failure, or operator error, assuming that additional copies of the archived and online logs are available
- Extract enables faster filtering of tables.
- Extract handles point-in-time recovery and RAC integration more efficiently.
- Extract features integrated log management. The Oracle Recovery Manager (RMAN) automatically retains the archive logs that are needed by Extract.
- Extract supports capture from a multitenant container database and from per-PDB capture mode.
- Extract and Replicat (integrated) are both database objects, so the naming of the objects follow the same rules as other Oracle database objects. See Specifying Object Names in Oracle GoldenGate Input in *Oracle GoldenGate Microservices Documentation*.
- When Extract is running from a remote system, Oracle GoldenGate automatically enables cross endian interoperability. This implies that if the endian value where Extract is running is different from the endian value where the Oracle database is running, then the cross endian support is automatically enabled. For cross endian Extract to work, the compatibility parameter of the source database must be 11.2.0.4 or higher.
- Each Extract group must process objects that are suited to the processing mode, based on table data types and attributes. No objects in one Extract can have DML or DDL dependencies on objects in the other Extract.

Deciding Which Extract Method to Use

The placement of Extract depends on where the mining database is located. The mining database is the one where the logmining server is deployed.

- **Local Extract**: For a local deployment, the source database and the mining database are the same. The source database is the database for which you want to mine the redo stream to capture changes, and also where you deploy the logmining server. Because integrated capture is fully integrated with the database, this mode does not require any special database setup.
- **Downstream Extract**: In a downstream deployment, the source and mining databases are different databases. You create the logmining server at the downstream database. You configure redo transport at the source database to ship the redo logs to the downstream mining database for capture at that location. Using a downstream mining server for capture may be desirable to offload the capture overhead and any other overhead from transformation or other processing from the production server, but requires log shipping and other configuration.

  When using a downstream mining configuration, the source database and mining database must be of the same platform. For example, if the source database is running on Windows 64-bit, the downstream database must also be on a Windows 64-bit platform. See Configuring a Downstream Mining Database .
- **Downstream sourceless Extract:** In the Extract parameter file, replace the `USERID` parameter with `NOUSERID`. You must use `TRANLOGOPTIONS MININGUSER`. Extract obtains all required information from the downstream mining database. Extract is not dependent on any connection to the source database. The source database can be shutdown and restarted without affecting Extract.

  Extract will abend if it encounters redo changes that require data to be fetched from the source database.

  To capture any tables that are listed as `ID KEY` in the `dba_goldengate_support_mode` view, you need to have a `FETCHUSERID` or `FETCHUSERIDALIAS` connection to support the tables. Tables that are listed as `FULL` do not require this. We also need to state that if a customer wants to perform `SQLEXEC` operations that perform a query or execute a stored procedure they cannot use this method as it is incompatible with `NOUSERID` because `SQLEXEC` works with `USERID` or `USERIDALIAS`.

The method that you use determines how you configure the Oracle GoldenGate processes and depends on such factors as:

- the data types involved
- the database configuration
- the version of the Oracle Database

Configuring Extract

This section contains instructions for configuring the Oracle GoldenGate Extract process to extract transaction data.

When Extract is running from a remote system, Oracle GoldenGate automatically enables cross endian interoperability. This implies that if the endian value where Extract is running is different from the endian value where the Oracle database is running, then the cross endian support is automatically enabled. For cross endian Extract to work, the compatibility parameter of the source database must be 11.2.0.4 or higher.

Add the Primary Extract

The primary Extract writes to a trail. These steps add the primary Extract that captures change data.

1. If using downstream capture, set the RMAN archive log deletion policy to the following value in the source database:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON ALL STANDBY
   ```

   This must be done before you add the primary Extract.
2. Run GGSCI.
3. If using integrated capture, issue the `DBLOGIN` command.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   Where: *`alias`* specifies the alias of the database login credential that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store.
4. Issue the `ADD EXTRACT` command to add the primary Extract group.

   ```
   ADD EXTRACT group name
   {, TRANLOG | , INTEGRATED TRANLOG}
   {, BEGIN {NOW | yyyy-mm-dd[ hh:mi:[ss[.cccccc]]]} | SCN value}
   [, THREADS n]
   ```

   Where:

   - *`group name`* is the name of the Extract group.
   - `INTEGRATED TRANLOG` specifies that Extract receives logical change records through a database logmining server; *for integrated capture only*. See Example 5-1. Before issuing `ADD EXTRACT` with this option, make certain you logged in to the database with the `DBLOGIN` command and that you registered this Extract with the database. See Registering Extract with the Mining Database for more information.
   - `BEGIN` specifies to begin capturing data as of a specific *time*:
     - `NOW` starts at the first record that is time stamped at the same time that `ADD EXTRACT` is issued.
     - *`yyyy-mm-dd[ hh:mi:[ss[.cccccc]]]`* starts at an explicit timestamp. Logs from this timestamp must be available. For Extract in integrated mode, the timestamp value must be greater than the timestamp at which the Extract was registered with the database.
     - `SCN` *`value`* starts Extract at the transaction in the redo log that has the specified Oracle system change number (SCN). For Extract in integrated mode, the SCN value must be greater than the SCN at which the Extract was registered with the database. See Registering Extract with the Mining Database for more information.
   - `THREADS` *`n`* is required in classic capture mode for Oracle Real Application Cluster (RAC), to specify the number of redo log threads being used by the cluster. Extract reads and coordinates each thread to maintain transactional consistency. Not required for integrated capture.

> **Note:**
>
> Additional options are available. See *Parameters and Functions Reference for Oracle GoldenGate*.

Example 5-1 Integrated capture with timestamp start point

```
DBLOGIN USERIDALIAS myalias
ADD EXTRACT finance, INTEGRATED TRANLOG, BEGIN NOW
```

Add the Data Pump Extract Group

These steps add the data pump that reads the local trail and sends the data to the target.

In GGSCI on the source system, issue the `ADD EXTRACT` command.

```
ADD EXTRACT group name, EXTTRAILSOURCE trail name
```

Where:

- group name is the name of the Extract group.
- `EXTTRAILSOURCE` *`trail name`* is the relative or fully qualified name of the local trail.

Example 5-2

```
ADD EXTRACT financep, EXTTRAILSOURCE c:\ggs\dirdat\lt
```

Registering Extract with the Mining Database

You need to create a database logmining server to capture redo data when using a downstream database.

The creation of the logmining server captures a snapshot of the source database in the redo stream of the source database. In a source multitenant container database, you register Extract with each of the pluggable databases that you want to include for Extract.

> **WARNING:**
>
> Make certain that you know the earliest SCN of the log stream at which you want Extract to begin processing. Extract cannot have a starting SCN value that is lower than the first SCN that is specified when the underlying database capture process is created with the `REGISTER EXTRACT` command. You can use the `SCN` option

1. Log into the mining database then use the commands appropriate to your environment. The use of `DBLOGIN` always refers to the source database.

   Command for source database deployment:

   ```
   DBLOGIN USERIDALIAS ggeast
   ```

   Command for downstream mining database deployment:

   ```
   DBLOGIN USERIDALIAS ggwest
   MININGDBLOGIN USERIDALIAS dbnorth
   ```

Where: `alias` specifies the alias of the database login credential that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials. For more information about `DBLOGIN` and `MININGDBLOGIN`, see *Parameters and Functions Reference for Oracle GoldenGate*.

2. Register the Extract process with the mining database.

   ```
   REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])] [SCN
   system_change_number]
   ```

   Where:

   - `group` is the name of the Extract group.
   - `CONTAINER (container[, ...])` specifies a pluggable database (PDB) within a multitenant container database, or a list of PDBs separated with commas. The specified PDBs must exist before the `REGISTER` command is executed. Extract will capture only from the PDBs that are listed in this command. For example, the following command registers PDBs `mypdb1` and `mypdb4`. Changes from any other PDBs in the multitenant container database are ignored by Oracle GoldenGate.

     ```
     REGISTER EXTRACT exte DATABASE CONTAINER (pdbeast, pdbwest, dbnorth)
     ```

     You can add or drop pluggable databases at a later date by stopping Extract, issuing a `DBLOGIN` command, and then issuing `REGISTER EXTRACT` with the `{ADD | DROP} CONTAINER` option of `DATABASE`.

     > **Note:**
     >
     > Adding `CONTAINER`s at particular SCN on an existing Extract is not supported.

   - Registers Extract to begin capture at a specific SCN in the past. Without this option, capture begins from the time that `REGISTER EXTRACT` is issued. The specified SCN must correspond to the begin SCN of a dictionary build operation in a log file. You can issue the following query to find all valid SCN values:

     ```
     SELECT first_change#
        FROM v$archived_log
        WHERE dictionary_begin = 'YES' AND
           STANDBY_DEST = 'NO' AND
           NAME IS NOT NULL AND
           STATUS = 'A';
     ```

3. To register additional Extracts with a downstream database for the same source database, issue this `REGISTER` command.

   If you want to have more than one extract per source database, you can do that using the `SHARE` with `REGISTER EXTRACT` for better performance and metadata management. The specified `SCN` must correspond to the `SCN` where mining should begin in the archive logs.

   ```
   REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])]
   [SCN system_change_number] SHARE
   ```

> **Note:**
>
> The register process may take a few to several minutes to complete, even though the `REGISTER` command returns immediately.

Creating an Online Extract Group

To create an online Extract group, run GGSCI on the source system and issue the `ADD EXTRACT` command. Separate all command arguments with a comma. There are two syntax forms:

- Syntax to Create a Regular, Passive, or Data Pump Extract Group
- **Syntax to Create an Alias Extract Group**

Syntax to Create a Regular, Passive, or Data Pump Extract Group

```
ADD EXTRACT group
{, datasource}
{, BEGIN start_point} | {position_point}
[, PASSIVE]
[, THREADS n]
[, PARAMS pathname]
[, REPORT pathname]
[, DESC 'description']
```

Where:

- *`group`* is the name of the Extract group. A group name is required.
- *`datasource`* is required to specify the source of the data to be extracted. Use one of the following:
  - `TRANLOG` specifies the transaction log as the data source. When using this option for Oracle Enterprise Edition, you must issue the `DBLOGIN` command as the Extract database user (or a user with the same privileges) before using `ADD EXTRACT` (and also before issuing `DELETE EXTRACT` to remove an Extract group).

    Use the *`bsds`* option for DB2 running on z/OS to specify the Bootstrap Data Set file name of the transaction log.
  - `INTEGRATED TRANLOG` specifies that this Extract will operate in integrated capture mode to receive logical change records (LCR) from an Oracle Database logmining server. This parameter applies only to Oracle Databases..
  - `EXTTRAILSOURCE` *`trail name`* to specify the relative or fully qualified name of a local trail. Use to create a data pump. A data pump can be used with any Oracle GoldenGate extraction method.
- `BEGIN` *`start_point`* defines an online Extract group by establishing an initial checkpoint and start point for processing. Transactions started before this point are discarded. Use one of the following:
  - `NOW` to begin extracting changes that are timestamped at the point when the `ADD EXTRACT` command is executed to create the group or, for an Oracle Extract in integrated mode, from the time the group is registered with the `REGISTER EXTRACT`

command. Do not use `NOW` for a data pump Extract unless you want to bypass any data that was captured to the Oracle GoldenGate trail prior to the `ADD EXTRACT` statement.

`YYYY-MM-DD HH:MM[:SS[.CCCCCC]]` as the format for specifying an exact timestamp as the begin point. Use a begin point that is later than the time at which replication or logging was enabled.

- *`position_point`* specifies a specific position within a specific transaction log file at which to start processing. For the specific syntax to use for your database, see `ADD EXTRACT` in *Parameters and Functions Reference for Oracle GoldenGate*.
- `PASSIVE` indicates that the group is a passive Extract. When using `PASSIVE`, you must also use an alias Extract. This option can appear in any order among other `ADD EXTRACT` options.
- `PARAMS` *`pathname`* is required if the parameter file for this group will be stored in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.
- `REPORT` *`pathname`* is required if the process report for this group will be stored in a location other than the `dirrpt` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.
- `DESC '`*`description`*`'` specifies a description of the group.

Syntax to Create an Alias Extract Group

```
ADD EXTRACT group
, RMTHOST {host | IP address}
, {MGRPORT port} | {PORT port}
[, RMTNAME name]
[, DESC 'description']
```

Where:

- `RMTHOST` identifies this group as an alias Extract and specifies either the DNS name of the remote host or its IP address.
- `MGRPORT` specifies the port on the remote system where Manager is running. Use this option when using a dynamic Collector.
- `PORT` specifies a static Collector port. Use instead of `MGRPORT` only if running a static Collector.
- `RMTNAME` specifies the passive Extract name, if different from that of the alias Extract.
- `DESC` '*`description`*' specifies a description of the group.

Example 5-3 Adding an Extract Group for Log-based Capture

This example creates an Extract group named `finance.` Extraction starts with records generated at the time when the group was created.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW
```

Example 5-4 Adding a Data-pump Extract Group

This example creates a data-pump Extract group named `finance.` It reads from the Oracle GoldenGate trail `c:\ggs\dirdat\lt`.

```
ADD EXTRACT finance, EXTTRAILSOURCE c:\ggs\dirdat\lt
```

Example 5-5 Adding a Passive Extract Group

This example creates a passive Extract group named `finance`. Extraction starts with records generated at the time when the group was created. Because this group is marked as passive, an alias Extract on the target will initiate connections to this Extract.

```
ADD EXTRACT finance, TRANLOG, BEGIN NOW, PASSIVE
```

Example 5-6 Adding a Passive Data-pump Extract Group

This example creates a data-pump Extract group named `finance`. This is a passive data pump Extract that reads from the Oracle GoldenGate trail `c:\ggs\dirdat\lt`. Because this data pump is marked as passive, an alias Extract on the target will initiate connections to it.

```
ADD EXTRACT finance, EXTTRAILSOURCE c:\ggs\dirdat\lt, PASSIVE
```

Example 5-7 Adding an Alias Extract Group

This example creates an alias Extract group named `alias`.

```
ADD EXTRACT alias, RMTHOST sysA, MGRPORT 7800, RMTNAME finance
```

Example 5-8 Adding a Primary Extract in Integrated Mode for Oracle

This example creates an Extract in integrated capture mode for an Oracle source database and sets the start point to the time when the Extract group is registered with the Oracle database by means of the `REGISTER EXTRACT` command. Integrated capture is available only for an Oracle database.

```
ADD EXTRACT finance INTEGRATED TRANLOG, BEGIN NOW
```

Configuring the Data Pump Extract

These steps configure the data pump that reads the local trail and sends the data across the network to a remote trail on the target. The data pump is optional, but recommended.

1. In GGSCI on the source system, create the data-pump parameter file.

   ```
   EDIT PARAMS name
   ```

   Where *`name`* is the name of the data-pump Extract.

2. Enter the data-pump Extract parameters in the order shown, starting a new line for each parameter statement. Your input variables will be different.

   Basic parameters for the data-pump Extract group:

   ```
   EXTRACT extpump
   SOURCEDB mypump, USERIDALIAS myalias
   RMTHOST fin1, MGRPORT 7809 ENCRYPT AES192, KEYNAME securekey2
   RMTTRAIL /ggs/dirdat/rt
   TABLE hr.*;
   ```

Parameter	Description
`EXTRACT` *`group`*	*`group`* is the name of the data pump Extract.

`SOURCEDB` *`database`*`,`
`USERIDALIAS` *`alias`*
	Specifies the real name of the source DB2 LUW database (not an alias), plus the alias of the database login credential of the user that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store.

Parameter	Description

`RMTHOST` *`hostname`*`,`
`MGRPORT` *`portnumber`*`,`
`[, ENCRYPT` *`algorithm`*
`KEYNAME` *`keyname`*`]`
	- `RMTHOST` specifies the name or IP address of the target system.
	- `MGRPORT` specifies the port number where Manager is running on the target.
	- `ENCRYPT` specifies optional encryption of data across TCP/IP.

`RMTTRAIL` *`pathname`*	Specifies the path name of the remote trail.
`TABLE` *`schema.object;`*	Specifies a table or sequence, or multiple objects specified with a wildcard. In most cases, this listing will be the same as that in the primary Extract parameter file. - `TABLE` is a required keyword. - *`schema`* is the schema name or a wildcarded set of schemas. - *`object`* is the name of a table or a wildcarded set of tables. Only the asterisk (*) wildcard is supported for DB2 LUW. The question mark (?) wildcard is not supported for this database. Terminate the parameter statement with a semi-colon. To exclude tables from a wildcard specification, use the `TABLEEXCLUDE` parameter. For more information and for additional `TABLE` options that control data filtering, mapping, and manipulation, see `TABLE | MAP` in *Parameters and Functions Reference for Oracle GoldenGate*.

3. Enter any optional Extract parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command in GGSCI.
4. Save and close the file.

Configuring a Downstream Mining Database

Learn the details for preparing a downstream Oracle mining database to support Extract. For examples of the downstream mining configuration, see the following:

Example 1: Capturing from One Source Database in Real-time Mode.

Example 2: Capturing from Multiple Sources in Archive-log-only Mode

Example 3: Capturing from Multiple Sources with Mixed Real-time and Archive-log-only Mode

Evaluating Capture Options for a Downstream Deployment

Downstream deployment allows you to offload the source database.

A downstream mining database can accept both archived logs and online redo logs from a source database.

Multiple source databases can send their redo data to a single downstream database; however the downstream mining database can accept *online* redo logs from only one of those source databases. The rest of the source databases must ship archived logs.

When online logs are shipped to the downstream database, *real-time capture* by Extract is possible. Changes are captured as though Extract is reading from the source logs. In order to accept online redo logs from a source database, the downstream mining database must have standby redo logs configured.

When using a downstream mining configuration, the source database and mining database must be the same endian and same bitsize, which is 64 bits. For example, if the source database was on Linux 64-bit, you can have the mining database run on Windows 64-bit, because they have the same endian and bitsize.

Preparing the Source Database for Downstream Deployment

The source database ships its redo logs to a downstream database, and Extract uses the logmining server at the downstream database to mine the redo logs.

This section guides you in the process of:

Creating the Source User Account

There must be an Extract user on the source database. Extract uses the credentials of this user to do metadata queries and to fetch column values as needed from the source database.

The source user is specified by the `USERIDALIAS` parameter.

To assign the required privileges, follow the procedure in Establishing Oracle GoldenGate Credentials

Configuring Redo Transport from Source to Downstream Mining Database

To set up the transfer of redo log files from a source database to the downstream mining database, and to prepare the downstream mining database to accept these redo log files, perform the steps given in this topic.

The following summarizes the rules for supporting multiple sources sending redo to a single downstream mining database:

- Only one source database can be configured to send *online* redo to the standby redo logs at the downstream mining database. The `log_archive_dest_n` setting for this source database should *not* have a `TEMPLATE` clause.
- Source databases that are *not* sending online redo to the standby redo logs of the downstream mining database *must have* a `TEMPLATE` clause specified in the `log_archive_dest_n` parameter.
- Each of the source databases that sends redo to the downstream mining database must have a unique `DBID`. You can select the `DBID` column from the `v$database` view of these source databases to ensure that the DBIDs are unique.
- The `FAL_SERVER` value must be set to the downstream mining database. `FAL_SERVER` specifies the FAL (fetch archive log) server for a standby database. The value is a list of Oracle Net service names, which are assumed to be configured properly on the standby database system to point to the desired FAL servers. The list contains the net service name of any database that can potentially ship redo to the downstream database.
- When using redo transport, there could be a delay in processing redo due to network latency. For Extract, this latency is monitored by measuring the delay between LCRs received from source database and reporting it. If the latency exceeds a threshold, a warning message appears in the report file and a subsequent information message appears when the lag drops to normal values. The default value for the threshold is 10 seconds.

> **Note:**
>
> The archived logs shipped from the source databases are called *foreign archived logs*. From Oracle Database 12.2.0.1 onward, the archived logs sent to downstream are purged automatically in downstream database as long as it is stored on Flash Recovery Area (FRA).

These instructions take into account the requirements to ship redo from multiple sources, if required. You must configure an Extract process for each of those sources.

To Configure Redo Transport

1. Configure Oracle Net so that each source database can communicate with the mining database. For instructions, see *Oracle Database Net Services Administrator's Guide*.
2. Configure authentication at each source database and at the downstream mining database to support the transfer of redo data. Redo transport sessions are authenticated using either the Secure Sockets Layer (SSL) protocol or a remote login password file. If a source database has a remote login password file, copy it to the appropriate directory of the mining database system. The password file must be the same at all source databases, and at the mining database. For more information about authentication requirements for redo transport, see Preparing the Primary Database for Standby Database Creation in *Oracle Data Guard Concepts and Administration*.
3. At each source database, configure one `LOG_ARCHIVE_DEST_n` initialization parameter to transmit redo data to the downstream mining database. Set the attributes of this parameter as shown in one of the following examples, depending on whether real-time or archived-log-only capture mode is to be used.
   - Example for real-time capture at the downstream logmining server, where the source database sends its online redo logs to the downstream database:

     ```
     ALTER SYSTEM
     SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC NOREGISTER
     VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap'
     ```

   - Example for archived-log-only capture at the downstream logmining server:

     ```
     ALTER SYSTEM SET
     LOG_ARCHIVE_DEST_2='SERVICE=DMBSCAP.EXAMPLE.COM ASYNC NOREGISTER
     VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
     TEMPLATE=/usr/oracle/log_for_dbms1/dbms1_arch_%t_%s_%r.log
     DB_UNIQUE_NAME=dbmscap'
     ```

   > **Note:**
   >
   > When using an archived-log-only downstream mining database, you must specify a value for the `TEMPLATE` attribute. Oracle also recommends that you use the `TEMPLATE` clause in the source databases so that the log files from all remote source databases are kept separated from the local database log files, and from each other.

4. At the source database, set a value of `ENABLE` for the `LOG_ARCHIVE_DEST_STATE_n` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_n` parameter that

corresponds to the destination for the downstream mining database, as shown in the following example.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

5. At the source database, and at the downstream mining database, set the `DG_CONFIG` attribute of the `LOG_ARCHIVE_CONFIG` initialization parameter to include the `DB_UNIQUE_NAME` of the source database and the downstream database, as shown in the following example.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)'
```

Preparing the Downstream Mining Database

A downstream mining database can accept both archived logs and online redo logs from a source database.

The following sections explain how to prepare the downstream mining database:

Creating the Downstream Mining User Account

When using a downstream mining configuration, there must be an Extract mining user on the downstream database. The mining Extract process uses the credentials of this user to interact with the downstream logmining server. The downstream mining user is specified by the `TRANLOGOPTIONS` parameter with the `MININGUSERALIAS` option. See Establishing Oracle GoldenGate Credentials to assign the correct credentials for the version of your database.

Configuring the Mining Database to Archive Local Redo Log Files

This procedure configures the downstream mining database to archive redo data in its online redo logs. These are redo logs that are generated at the downstream mining database.

Archiving must be enabled at the downstream mining database if you want to run Extract in real-time integrated capture mode, but it is also recommended for archive-log-only capture. Extract in integrated capture mode writes state information in the database. Archiving and regular backups will enable you to recover this state information in case there are disk failures or corruption at the downstream mining database.

To Archive Local Redo Log Files

1. Alter the downstream mining database to be in archive log mode. You can do this by issuing the following DDL.

```
STARTUP MOUNT;
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

2. At the downstream mining database, set the first archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example:

```
ALTER SYSTEM SET
LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)'
```

Alternatively, you can use a command like this example:

```
ALTER SYSTEM SET
LOG_ARCHIVE_DEST_1='LOCATION='USE_DB_RECOVERY_FILE_DEST'
valid_for=(ONLINE_LOGFILE,PRIMARY_ROLE)'
```

> **Note:**
>
> The online redo logs generated by the downstream mining database can be archived to a recovery area. However, you must not use the recovery area of the downstream mining database to stage foreign archived logs or to archive standby redo logs. For information about configuring a fast recovery area, see the *Oracle Database Backup and Recovery User's Guide*.

3. Enable the local archive destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

For more information about these initialization parameters, see Set Primary Database Initialization Parameters in the *Oracle Data Guard Concepts and Administration* guide.

Preparing a Downstream Mining Database for Real-time Capture

This procedure is only required if you want to use real-time capture at a downstream mining database. It is not required to use archived-log-only capture mode. To use real-time capture, it is assumed that the downstream database has already been configured to archive its local redo data as shown in Configuring the Mining Database to Archive Local Redo Log Files .

Create the Standby Redo Log Files

The following steps outline the procedure for adding standby redo log files to the downstream mining database. The following summarizes the rules for creating the standby redo logs:

- Each standby redo log file must be at least as large as the largest redo log file of the redo source database. For administrative ease, Oracle recommends that all redo log files at source database and the standby redo log files at the downstream mining database be of the same size.
- The standby redo log must have at least one more redo log group than the redo log at the source database, for each redo thread at the source database.

The specific steps and SQL statements that are required to add standby redo log files depend on your environment. See Creating a Physical Standby Database for detailed instructions about adding standby redo log files to a database.

Note:

If there will be multiple source databases sending redo to a single downstream mining database, only one of those sources can send redo to the standby redo logs of the mining database. An Extract process that mines the redo from this source database can run in real-time mode. All other source databases must send only their archived logs to the downstream mining database, and the Extracts that read this data must be configured to run in archived-log-only mode.

To Create the Standby Redo Log Files

1. In SQL*Plus, connect to the source database as an administrative user.
2. Determine the size of the source log file. Make note of the results.

   ```
   SELECT BYTES FROM V$LOG;
   ```

3. Determine the number of online log file groups that are configured on the source database. Make note of the results.

   ```
   SELECT COUNT(GROUP#) FROM V$LOG;
   ```

4. Connect to the downstream mining database as an administrative user.
5. Add the standby log file groups to the mining database. The standby log file size must be at least the size of the source log file size. The number of standby log file groups must be at least one more than the number of source online log file groups. This applies to each instance (thread) in a RAC installation. So if you have **`n`** threads at the source database, each having **`m`** redo log groups, you should configure **`n*(m+1)`** redo log groups at the downstream mining database.

   The following example shows three standby log groups.

   ```
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 3
   ('/oracle/dbs/slog3a.rdo', '/oracle/dbs/slog3b.rdo') SIZE 500M;
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 4
   ('/oracle/dbs/slog4.rdo', '/oracle/dbs/slog4b.rdo') SIZE 500M;
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 5
   ('/oracle/dbs/slog5.rdo', '/oracle/dbs/slog5b.rdo') SIZE 500M;
   ```

6. Confirm that the standby log file groups were added successfully.

   ```
   SELECT GROUP#, THREAD#, SEQUENCE#, ARCHIVED, STATUS
   FROM V$STANDBY_LOG;
   ```

   The output should be similar to the following:

   ```
   GROUP#     THREAD#    SEQUENCE#  ARC STATUS
   ---------- ---------- ---------- --- ----------
            3          0          0 YES UNASSIGNED
            4          0          0 YES UNASSIGNED
            5          0          0 YES UNASSIGNED
   ```

7. Ensure that log files from the source database are appearing in the location that is specified in the `LOCATION` attribute of the local `LOG_ARCHIVE_DEST_n` that you set. You might need to switch the log file at the source database to see files in the directory.

Configure the Database to Archive Standby Redo Log Files Locally

This procedure configures the downstream mining database to archive the standby redo logs that receive redo data from the online redo logs of the source database. Keep in mind that foreign archived logs should not be archived in the recovery area of the downstream mining database.

To Archive Standby Redo Logs Locally

1. At the downstream mining database, set the second archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms1
   VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
   ```

   Oracle recommends that foreign archived logs (logs from remote source databases) be kept separate from local mining database log files, and from each other. You must not use the recovery area of the downstream mining database to stage foreign archived logs..

2. Enable the `LOG_ARCHIVE_DEST_2` parameter you set in the previous step as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
   ```

Example Downstream Mining Configuration

This section contains examples for preparing a downstream Oracle mining database to support Extract.

Example 1: Capturing from One Source Database in Real-time Mode

This example captures changes from source database DBMS1 by deploying an Extract at a downstream mining database DBMSCAP.

> **Note:**
>
> The example assumes that you created the necessary standby redo log files as shown in Preparing the Downstream Mining Database.

This assumes that the following users exist:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. This user has the alias of `ggadm1` in the Oracle GoldenGate credential store and logs in as `ggadm1@dbms1`. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the source database.
- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database DBMSCAP. This user has the alias of `ggadmcap` in the Oracle GoldenGate credential store and logs in

as `ggadmcap@dbmscap`. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the mining database.

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

   ```
   STARTUP MOUNT;
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
   VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
   ```

3. Enable `log_archive_dest_1`.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
   ```

Prepare the Mining Database to Archive Redo Received in Standby Redo Logs from the Source Database

To prepare the mining database to archive the redo received in standby redo logs from the source database:

1. At the downstream mining database, set `log_archive_dest_2` as shown in the following example.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms1
   VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
   ```

2. Enable `log_archive_dest_2` as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
   ```

3. Set `DG_CONFIG` at the downstream mining database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)'
   ```

Prepare the Source Database to Send Redo to the Mining Database

To prepare the source database to send redo to the mining database:

1. Make sure that the source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME            VALUE
   ---------       ---------------------
   compatible      11.1.0.7.0
   ```

   The minimum compatibility setting required from integrated capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at the source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)';
   ```

3. Set up redo transport at the source database.

```
ALTER SYSTEM
SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
```

4. Enable the downstream destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

Set up Extract (ext1) on DBMSCAP

To set up Extract (ext1) on DBMSCAP:

1. Register Extract with the downstream mining database. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

```
GGSCI> DBLOGIN USERIDALIAS ggadm1
GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
GGSCI> REGISTER EXTRACT ext1 DATABASE
```

2. Create Extract at the downstream mining database.

```
GGSCI> ADD EXTRACT ext1 INTEGRATED TRANLOG BEGIN NOW
```

3. Edit Extract parameter file `ext1.prm`. The following lines must be present to take advantage of real-time capture. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

```
USERIDALIAS ggadm1
TRANLOGOPTIONS MININGUSERALIAS ggadmcap
TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine Y)
```

4. Start Extract.

```
GGSCI> START EXTRACT ext1
```

Note:

You can create multiple Extracts running in real-time Extract mode in the downstream mining database, as long as they all are capturing data from the same source database, such as capturing changes for database DBMS1 in the preceding example.

Example 2: Capturing from Multiple Sources in Archive-log-only Mode

The following example captures changes from database DBMS1 and DBMS2 by deploying an Extract at a downstream mining database DBMSCAP.

It assumes the following users:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS1.
- User GGADM2 in DBMS2 whose credentials Extract will use to fetch data and metadata from DBMS2. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS2.

- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the downstream mining database DBMSCAP.

This procedure also assumes that the downstream mining database is configured in archive log mode.

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

   ```
   STARTUP MOUNT;
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
   VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
   ```

3. Enable `log_archive_dest_1`.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
   ```

Prepare the Mining Database to Archive Redo from the Source Database

Set `DG_CONFIG` at the downstream mining database.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbms2, dbmscap)'
```

Prepare the First Source Database to Send Redo to the Mining Database

To prepare the first source database to send redo to the mining database:

1. Make certain that DBMS1 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME            VALUE
   ---------       ---------------------
   compatible      11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS1 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbmscap)';
   ```

3. Set up redo transport at DBMS1 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms1/dbms1_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

Prepare the Second Source Database to Send Redo to the Mining Database

To prepare the second source database to send redo to the mining database:

1. Make sure that DBMS2 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME                    VALUE
   ---------               ---------------------
   compatible              11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS2 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms2, dbmscap)';
   ```

3. Set up redo transport at DBMS2 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms2/dbms2_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Set up Extracts at Downstream Mining Database

These steps set up Extract at the downstream database to capture from the archived logs sent by DBMS1 and DBMS2.

Example 3: Capturing from Multiple Sources with Mixed Real-time and Archive-log-only Mode

The following example captures changes from database DBMS1, DBMS2 and DBMS3 by deploying an Extract at a downstream mining database DBMSCAP.

> **Note:**
>
> This example assumes that you created the necessary standby redo log files as shown in Preparing the Downstream Mining Database.

It assumes the following users:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS1.
- User GGADM2 in DBMS2 whose credentials Extract will use to fetch data and metadata from DBMS2. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS2.

- User GGADM3 in DBMS3 whose credentials Extract will use to fetch data and metadata from DBMS3. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS3.
- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the downstream mining database DBMSCAP.

This procedure also assumes that the downstream mining database is configured in archive log mode.

In this example, the redo sent by DBMS3 will be mined in real time mode, whereas the redo data sent from DBMS1 and DBMS2 will be mined in archive-log-only mode.

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

```
STARTUP MOUNT;
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

```
ALTER SYSTEM SETLOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/
localVALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
```

3. Enable `log_archive_dest_1`.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

Prepare the Mining Database to Accept Redo from the Source Databases

Because redo data is being accepted in the standby redo logs of the downstream mining database, the appropriate number of correctly sized standby redo logs must exist. If you did not configure the standby logs, see Configuring a Downstream Mining Database .

1. At the downstream mining database, set the second archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example. This is needed to handle archive standby redo logs.

```
ALTER SYSTEM SET
LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms3
VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
```

2. Enable the `LOG_ARCHIVE_DEST_STATE_2` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_2` parameter as shown in the following example.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

3. Set `DG_CONFIG` at the downstream mining database to accept redo data from all of the source databases.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbms2, dbms3, dbmscap)'
```

Prepare the First Source Database to Send Redo to the Mining Database

To prepare the first source database to send redo to the mining database:

1. Make certain that DBMS1 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME             VALUE
   ---------        ---------------------
   compatible       11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.
2. Set `DG_CONFIG` at DBMS1 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbmscap)';
   ```
3. Set up redo transport at DBMS1 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms1/dbms1_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```
4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Prepare the Second Source Database to Send Redo to the Mining Database

To prepare the second source database to send redo to the mining database:

1. Make sure that DBMS2 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME                     VALUE
   ---------                ---------------------
   compatible               11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.
2. Set `DG_CONFIG` at DBMS2 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms2, dbmscap)';
   ```
3. Set up redo transport at DBMS2 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms2/dbms2_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```
4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Prepare the Third Source Database to Send Redo to the Mining Database

To prepare the third source database to send redo to the mining database:

1. Make sure that DBMS3 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';
   ```

```
NAME                    VALUE
---------               ---------------------
compatible              11.1.0.0.0
```

The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS3 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms3, dbmscap)';
   ```

3. Set up redo transport at DBMS3 source database. Because DBMS3 is the source that will send its online redo logs to the standby redo logs at the downstream mining database, do not specify a `TEMPLATE` clause.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Set up Extracts at Downstream Mining Database

These steps set up Extract at the downstream database to capture from the archived logs sent by DBMS1 and DBMS2.

Set up Extract (ext1) to Capture Changes from Archived Logs Sent by DBMS1

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with DBMSCAP for the DBMS1 source database. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   GGSCI> DBLOGIN USERIDALIAS ggadm1
   GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
   GGSCI> REGISTER EXTRACT ext1 DATABASE
   ```

2. Add Extract at the mining database DBMSCAP.

   ```
   GGSCI> ADD EXTRACT ext1 INTEGRATED TRANLOG BEGIN NOW
   ```

3. Edit the Extract parameter file `ext1.prm`. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm1
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine N)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext1
   ```

Set up Extract (ext2) to Capture Changes from Archived Logs Sent by DBMS2

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with the mining database for source database DBMS2. In the credential store, the alias name of `ggadm2` is linked to a user connect string of `ggadm2@dbms2`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

```
GGSCI> DBLOGIN USERIDALIAS ggadm2
GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
GGSCI> REGISTER EXTRACT ext2 DATABASE
```

2. Create Extract at the mining database.

   ```
   GGSCI> ADD EXTRACT ext2 INTEGRATED TRANLOG, BEGIN NOW
   ```

3. Edit the Extract parameter file `ext2.prm`. In the credential store, the alias name of `ggadm2` is linked to a user connect string of `ggadm2@dbms2`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm2
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine N)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext2
   ```

Set up Extract (ext3) to Capture Changes in Real-time Mode from Online Logs Sent by DBMS3

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with the mining database for source database DBMS3. In the credential store, the alias name of `ggadm3` is linked to a user connect string of `ggadm3@dbms3`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   GGSCI> DBLOGIN USERID ggadm3
   GGSCI> MININGDBLOGIN USERID ggadmcap
   GGSCI> REGISTER EXTRACT ext3 DATABASE
   ```

2. Create Extract at the mining database.

   ```
   GGSCI> ADD EXTRACT ext3 INTEGRATED TRANLOG, BEGIN NOW
   ```

3. Edit the Extract parameter file `ext3.prm`. To enable real-time mining, you must specify `downstream_real_time_mine`. In the credential store, the alias name of `ggadm3` is linked to a user connect string of `ggadm3@dbms3`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm3
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine Y)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext3
   ```

> **Note:**
>
> You can create multiple Extracts running in real-time integrated capture mode in the downstream mining database, as long as they all are capturing data from the same source database, such as all capturing for database DBMS3 in the preceding example.

Positioning Extract to a Specific Start Point for MySQL

You can position the Extract to a specific start point in the transaction logs using the `ADD/ALTER EXTRACT` commands:

```
{ADD | ALTER EXTRACT} group, LOGNUM log_num, LOGPOS log_pos
```

- *`group`* is the name of the Oracle GoldenGate Extract group for which the start position is required.
- `LOGNUM` is the log file number. For example, if the required log file name is `test.000034`, the `LOGNUM` value is 34. Extract will search for this log file.
- *`LOGPOS`* is an event offset value within the log file that identifies a specific transaction record. Event offset values are stored in the header section of a log record. To position at the beginning of a `binlog` file, set the `LOGPOS` as 0.

In MySQL logs, an event offset value can be unique only within a given binary file. The combination of the position value and a log number will uniquely identify a transaction record. Maximum Log number length is 8 bytes unsigned integer and Maximum Log offset length is 8 bytes unsigned integer. Log number and Log offset are separated by a pipe (‘|’) delimiter. Transactional records available after this position within the specified log will be captured by Extract. In addition, you can position an Extract using a timestamp.

Additional Parameter Options for Extract

This section contains additional parameters that may be required for your Extract configuration.

Extract uses a database logmining server in the mining database to mine the redo stream of the source database. You can set parameters that are specific to the logmining server by using the `TRANLOGOPTIONS` parameter with the `INTEGRATEDPARAMS` option in the Extract parameter file.

> **Note:**
>
> For detailed information and usage guidance for these parameters, see the "`DBMS_CAPTURE_ADM`" section in PL/SQL Packages and Types Reference.

The following parameters can be set with `INTEGRATEDPARAMS`:

- `CAPTURE_IDKEY_OBJECTS`: Controls the capture of objects that can be supported by `FETCH`. The default for Oracle GoldenGate is `Y` (capture ID key logical change records).
- `DOWNSTREAM_REAL_TIME_MINE`: Controls whether the logmining server operates as a real-time downstream capture process or as an archived-log downstream capture process. The default is `N` (archived-log mode). Specify this parameter to use real-time capture in a downstream logmining server configuration. For more information on establishing a downstream mining configuration, see Configuring a Downstream Mining Database .
- `INLINE_LOB_OPTIMIZATION`: Controls whether LOBs that can be processed inline (such as small LOBs) are included in the LCR directly, rather than sending LOB chunk LCRs. The default for Oracle GoldenGate is `Y` (Yes).
- `MAX_SGA_SIZE`: Controls the amount of shared memory used by the logmining server. The shared memory is obtained from the streams pool of the SGA. The default is 1 GB.

- `PARALLELISM`: Controls the number of processes used by the logmining server. The default is 2. For Oracle Standard Edition, this must be set to `1`.
- `TRACE_LEVEL`: Controls the level of tracing for the Extract logmining server. For use only with guidance from Oracle Support. The default for Oracle GoldenGate is `0` (no tracing).
- `WRITE_ALERT_LOG`: Controls whether the Extract logmining server writes messages to the Oracle alert log. The default for Oracle GoldenGate is `Y` (Yes).

See Managing Server Resources.

6

Replicat

Learn about the Replicat process, its types, and steps to add a replicat, and other tasks associated with Replicat.

About Replicat

Replicat is a process that delivers data to a target system. It reads the trail file on the target database, reconstructs the DML or DDL operations, and applies them to the target database.

The Replicat process uses SQL to compile a SQL statement once and then executes it many times with different bind variables. You can configure the Replicat process so that it waits a specific amount of time before applying the replicated operations to the target database. For example, a delay may be desirable to prevent the propagation of errant SQL, to control data arrival across different time zones, or to allow time for other planned events to occur.

For the following two common uses cases of Oracle GoldenGate, the function of the Replicat process is as follows:

- **Initial Loads**: When you set up Oracle GoldenGate for initial loads, the Replicat process applies a static data copy to target objects or routes the data to a high-speed bulk-load utility.
- **Change Synchronization**: When you set up Oracle GoldenGate to keep the target database synchronized with the source database, the Replicat process applies the source operations to the target objects using a native database interface or ODBC, depending on the database type.

You can configure multiple Replicat processes with one or more Extract processes to increase the throughput. To preserve data integrity, each set of processes handles a different set of objects. To differentiate among Replicat processes, you assign each one a group name.

Deciding Which Replicat Method to Use

The Replicat process applies replicated data to an Oracle target database.

For an Oracle target database, you can run Replicat in parallel, non-integrated or integrated mode. Oracle recommends that you use the parallel Replicat unless a specific feature requires a different type of Replicat.

The following table lists the features supported by the respective Replicats.

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
Batch Processing	Yes	Yes	Yes	Yes
Barrier Transactions	Yes	Yes	Yes	No
Dependency Computation	Yes	Yes	No	No

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
Auto-parallelism	Yes	Yes	No	No
	Note: Auto-parallelism is disabled, by default. Only fou			

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	r threads are used in the default settings. If you want to change Re			

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	plicat to use `MIN_PARALLELISM` and `MAX_PARALLELISM` then auto-			

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	parallelism is used.			
DML Handler	Yes, Integrated mode	Yes	No	No
Procedural Replication	Yes, used for integrated Parallel Replicat (iPR)	Yes	No	No
Auto CDR	Yes, used by iPR only	Yes	No	No
Dependency-aware Transaction Split	Yes	No	No	No
Cross-RAC-node Processing	Yes	No	Yes	No
`ALLOWDUPTARGETMAP` See `ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP`	No. Oracle Database with iPR	No, Oracle Database	Yes	Yes

About Parallel Replicat

Parallel Replicat is another variant of Replicat that applies transactions in parallel to improve performance.

It takes into account dependencies between transactions, similar to Integrated Replicat. The dependency computation, parallelism of the mapping and apply is performed outside the database so can be off-loaded to another server. The transaction integrity is maintained in this process. In addition, parallel Replicat supports the parallel apply of large transactions by splitting a large transaction into chunks and applying them in parallel.

> **Note:**
>
> For best performance for an OLTP workload, parallel Replicat in non-integrated mode is recommended.

Only Oracle database supports parallel Replicat and integrated parallel Replicat. However, parallel Replicat supports all databases when using the non-integrated option.

To use parallel Replicat, you need to ensure that you have the following values, which are also the default values:

- Metadata in the trail (which means you can't use parallel Replicat if your trails are formatted below 12.1.
- Scheduling columns in the trail file.
- `UPDATERCORDFORMAT COMPACT` parameter.

With integrated parallel Replicat, the Replicat sends the LCRs to the inbound server, which applies the data to the target database, and in regular parallel Replicat, Oracle GoldenGate applies the LCR as a SQL statement directly to the database, similar to how the other non-integrated Replicats work.

The components of parallel Replicat are:

- Mappers operate in parallel to read the trail, map trail records, convert the mapped records to the Integrated Replicat LCR format, and send the LCRs to the Merger for further processing. While one Mapper maps one set of transactions, the next Mapper maps the next set of transactions. The the trail information is split and the trail file is untouched because it orders trail information in order.
- Master processes have two threads, Collater and Scheduler. The Collater receives mapped transactions from the Mappers and puts them back into trail order for dependency calculation. The Scheduler calculates dependencies between transactions, groups transactions into independent batches, and sends the batches to the Appliers to be applied to the target database.
- Appliers reorder records within a batch for array execution. It applies the batch to the target database and performs error handling. It also tracks applied transactions in checkpoint tables.

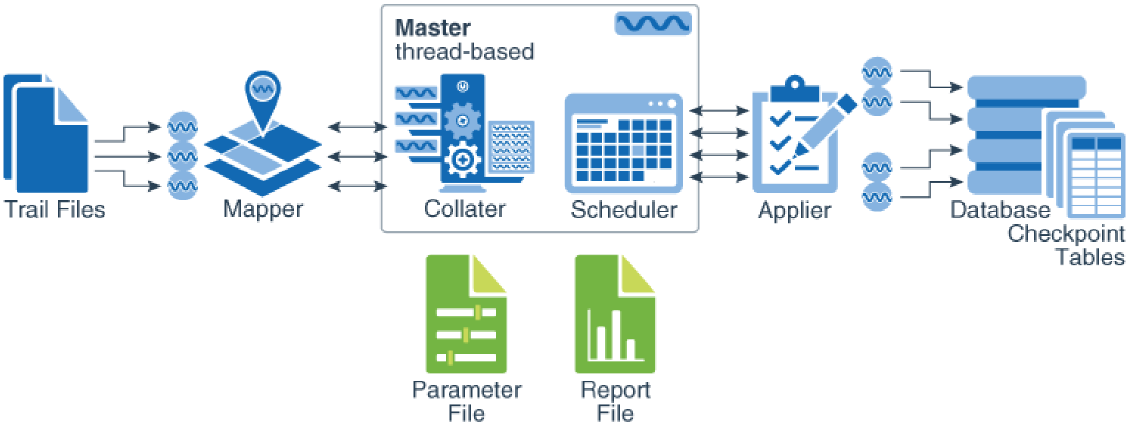
> **Note:**
>
> Parallel Replicat requires that any foreign key columns are indexed.

Parallel Replication Architecture

Parallel replication processes leverage the apply processing functionality that is available within the Oracle Database in integrated mode.

Within a single Replicat configuration, multiple inbound server child processes, known as apply servers, apply transactions in parallel while preserving the original transaction atomicity.

The following architecture diagram depicts the flow of change records through the various processes of a parallel replication from the trail files to the target database, for a non-integrated parallel Replicat.



The Mappers read the trail file and map records, forward the mapped records to the Master. The batches are sent to the Appliers where they are applied to the target database.

The Master process consists of two separate threads, Collater and Scheduler. The Collater is responsible for managing and communicating with the Mappers, along with receiving the mapped transactions and reordering them into a single in-order stream. The Scheduler is responsible for managing and communicating with the Appliers, along with reading transactions from the Collater, batching them, and scheduling them to Appliers.

The Scheduler controller communicates with the Scheduler to gather any necessary information (such as, the current low watermark position). The Scheduler controller is required for CDB mode for Oracle Database because it is responsible for aggregating information pertaining to the different target PDBs and reporting a unified picture. The Scheduler controller is created for simplicity and uniformity of implementation, even when not in CDB mode. Every process reads the parameter file and shares a single checkpoint file.

Basic Parameters for Parallel Replicat

The following table lists the basic parallel Replicat parameters and their description.

Parameter	Description
`MAP_PARALLELISM`	Configures number of mappers. This controls the number of threads used to read the trail file. The minimum value is `1`, maximum value is `100` and the default value is `2`.
`APPLY_PARALLELISM`	Configures number of appliers. This controls the number of connections in the target database used to apply the changes. The default value is four.
`MIN_APPLY_PARALLELISM` `MAX_APPLY_PARALLELISM`	The Apply parallelism is auto-tuned. You can set a minimum and maximum value to define the ranges in which the Replicat automatically adjusts its parallelism. There are no defaults. Do *not* use with `APPLY_PARALLELISM` at same time.
`SPLIT_TRANS_REC`	Specifies that large transactions should be broken into pieces of specified size and applied in parallel. Dependencies between pieces are still honored. Disabled by default.

Parameter	Description
`COMMIT_SERIALIZATION`	Enables commit `FULL` serialization mode, which forces transactions to be committed in trail order.
Advanced Parameters	
`LOOK_AHEAD_TRANSACTIONS`	Controls how far ahead the Scheduler looks when batching transactions. The default value is 10000.
`CHUNK_SIZE`	Controls how large a transaction must be for parallel Replicat to consider it as large. When parallel Replicat encounters a transaction larger than this size, it will serialize it, resulting in decreased performance. However, increasing this value will also increase the amount of memory consumed by parallel Replicat.

Example Parameter File

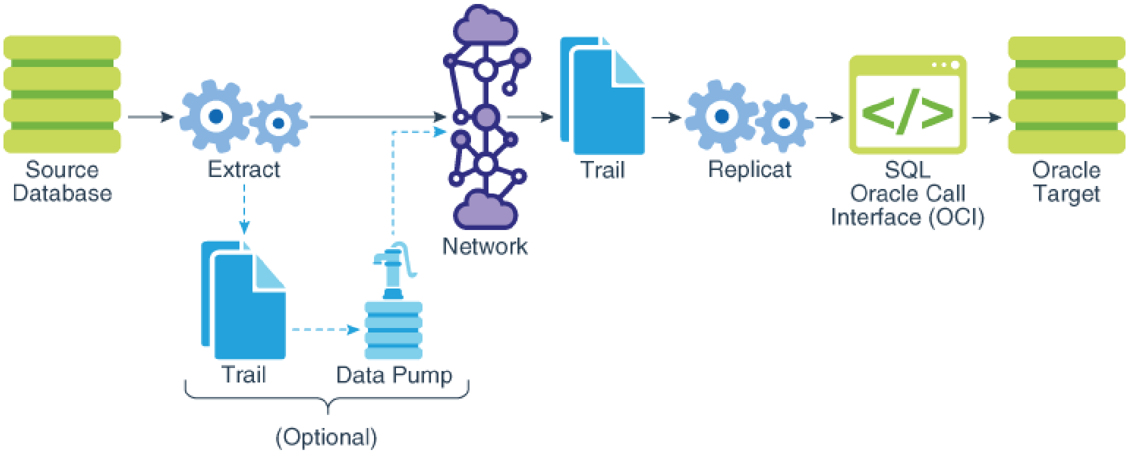
```
Replicat repe
USERID ggadmin, PASSWORD ***
MAP_PARALLELISM 3
MIN_APPLY_PARALLELISM 2
MAX_APPLY_PARALLELISM 10
SPLIT_TRANS_RECS 60000
MAP *.*, TARGET *.*;
```

About Non-integrated Replicat

In non-integrated mode, the Replicat process uses standard SQL to apply data directly to the target tables. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Constructs SQL statements that represent source database DML or DDL transactions (in committed order).
- Applies the SQL to the target through Oracle Call Interface (OCI).

The following diagram illustrates the configuration of Replicat in non-integrated mode.



Use non-integrated Replicat when you want to make heavy use of features that are not supported in integrated Replicat mode.

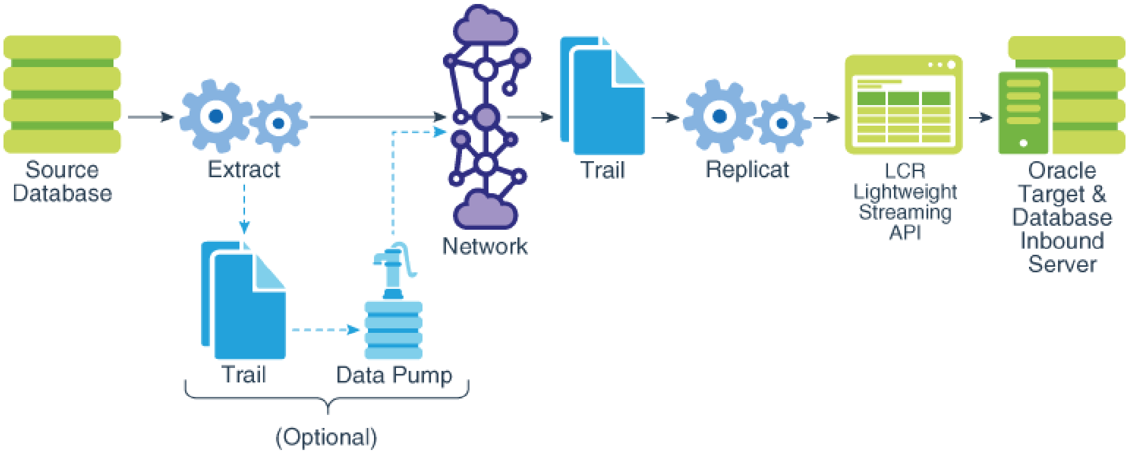
You can apply transactions in parallel with a non-integrated Replicat by using a coordinated Replicat configuration.

About Integrated Replicat

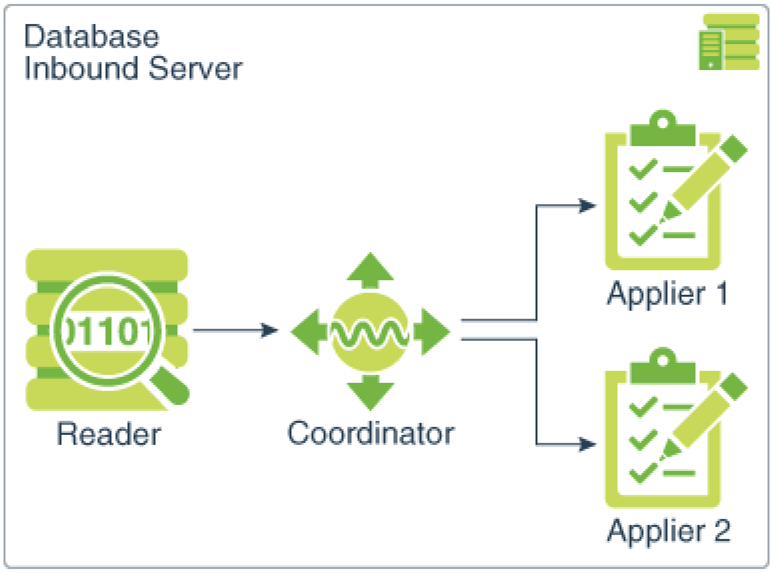
In integrated mode, the Replicat process leverages the apply processing functionality that is available within the Oracle Database. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Constructs logical change records (LCR) that represent source database DML transactions (in committed order). DDL is applied directly by Replicat.
- Attaches to a background process in the target database known as a *database inbound server* by means of a lightweight streaming interface.
- Transmits the LCRs to the inbound server, which applies the data to the target database.

The following figure illustrates the configuration of Replicat in integrated mode.


Source Database
Extract
Network
Trail
Replicat
LCR Lightweight Streaming API
Oracle Target & Database Inbound Server
Trail
Data Pump
(Optional)

Within a single Replicat configuration, multiple inbound server child processes known as *apply servers* apply transactions in parallel while preserving the original transaction atomicity. You can increase this parallelism as much as your target system will support when you configure the Replicat process or dynamically as needed. The following diagram illustrates integrated Replicat configured with two parallel apply servers.



Integrated Replicat applies transactions asynchronously. Transactions that do not have interdependencies can be safely executed and committed out of order to achieve fast throughput. Transactions with dependencies are guaranteed to be applied in the same order as on the source.

A reader process in the inbound server computes the dependencies among the transactions in the workload based on the constraints defined at the target database (primary key, unique, foreign key). Barrier transactions and DDL operations are managed automatically, as well. A coordinator process coordinates multiple transactions and maintains order among the apply servers.

If the inbound server does not support a configured feature or column type, Replicat disengages from the inbound server, waits for the inbound server to complete transactions in its queue, and then applies the transaction to the database in *direct apply* mode through OCI. Replicat resumes processing in integrated mode after applying the direct transaction.

The following features are applied in direct mode by Replicat:

- DDL operations
- Sequence operations
- `SQLEXEC` parameter within a `TABLE` or `MAP` parameter
- `EVENTACTIONS` processing
- `UDT`

> **Note:**
>
> By default, UDT's are applied with the inbound server. Only if `NOUSENATIVEOBJSUPPORT` is in place, then Extract handling is done by Replicat directly.

Because transactions are applied serially in direct apply mode, heavy use of such operations may reduce the performance of the integrated Replicat mode. Integrated Replicat performs best when most of the apply processing can be performed in integrated mode, see Monitoring and Controlling Processing After the Instantiation in .

Note:

User exits are executed in integrated mode. However, user exit may produce unexpected results, if the exit code depends on data in the replication stream.

Note:

Integrated Replicat requires that any foreign key columns are indexed.

Benefits of Integrated Replicat

The following are the benefits of using integrated Replicat versus nonintegrated Replicat.

- Integrated Replicat enables heavy workloads to be partitioned automatically among parallel apply processes that apply multiple transactions concurrently, while preserving the integrity and atomicity of the source transaction. Both a minimum and maximum number of apply processes can be configured with the `PARALLELISM` and `MAX_PARALLELISM` parameters. Replicat automatically adds additional servers when the workload increases, and then adjusts downward again when the workload lightens.
- Integrated Replicat requires minimal work to configure. All work is configured within one Replicat parameter file, without configuring range partitions.
- High-performance apply streaming is enabled for integrated Replicat by means of a lightweight application programming interface (API) between Replicat and the inbound server.
- Barrier transactions are coordinated by integrated Replicat among multiple server apply processes.
- DDL operations are processed as direct transactions that force a barrier by waiting for server processing to complete before the DDL execution.
- Transient duplicate primary key updates are handled by integrated Replicat in a seamless manner.

Integrated Replicat Requirements

To use integrated Replicat, the following must be true.

- Supplemental logging must be enabled on the source database to support the computation of dependencies among tables and scheduling of concurrent transactions on the target. Instructions for enabling the required logging are in Configuring Logging Properties. This logging can be enabled at any time up to, but before, you start the Oracle GoldenGate processes.
- Integrated Parallel Replicat is supported on Oracle Database 12.2.0.1 and greater.

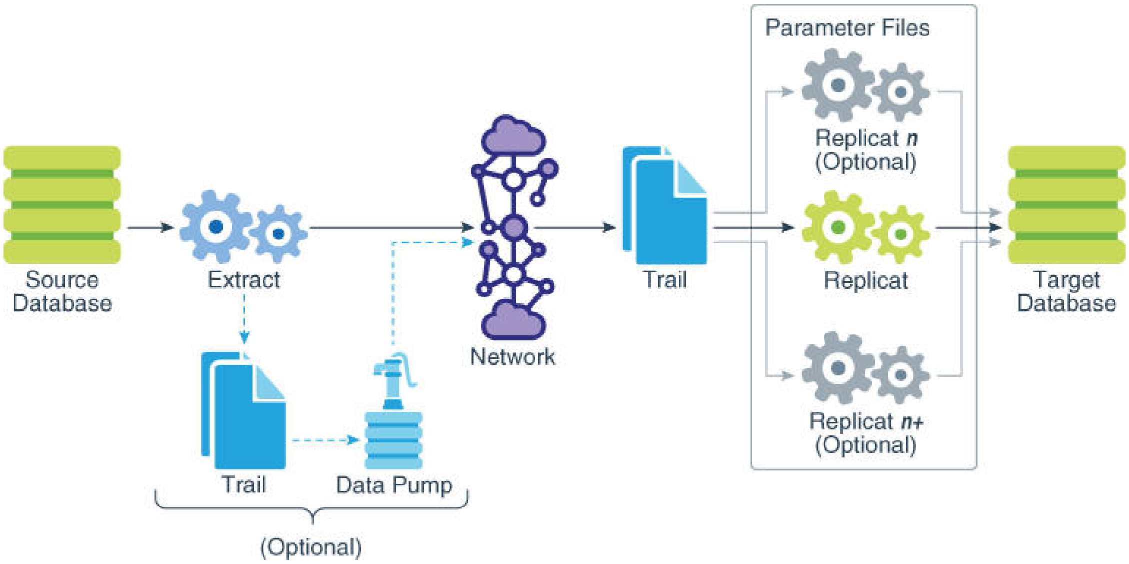
About Classic Replicat Mode

In classic mode, Replicat is a single-threaded process that uses standard SQL to apply data to the target tables. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.

- Performs data filtering, mapping, and conversion.
- Constructs SQL statements that represent source database DML or DDL transactions (in committed order).
- Applies the SQL to the target through the SQL interface that is supported for the given target database, such as ODBC or the native database interface.

Figure 6-1 Classic Replicat



As shown in Figure 6-1, you can apply transactions in parallel with a classic Replicat, but only by partitioning the workload across multiple Replicat processes. A parameter file must be created for each Replicat.

To determine whether to use classic mode for any objects, you must determine whether the objects in one Replicat group will ever have dependencies on objects in any other Replicat group, transactional or otherwise. Not all workloads can be partitioned across multiple Replicat groups and still preserve the original transaction atomicity. For example, tables for which the workload routinely updates the primary key cannot easily be partitioned in this manner. DDL replication (if supported for the database) is not viable in this mode, nor is the use of some `SQLEXEC` or `EVENTACTIONS` features that base their actions on a specific record.

If your tables do not have any foreign- key dependencies or updates to primary keys, classic mode may be suitable. Classic mode requires less overhead than coordinated mode.

For more information about using parallel Replicat groups, see About Parallel Replicat.

About Coordinated Replicat Mode

In coordinated mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Processes operations sent to each thread in a committed order.
- Applies the SQL to the target through the SQL interface that is supported for the given target database, such as ODBC or the native database interface.

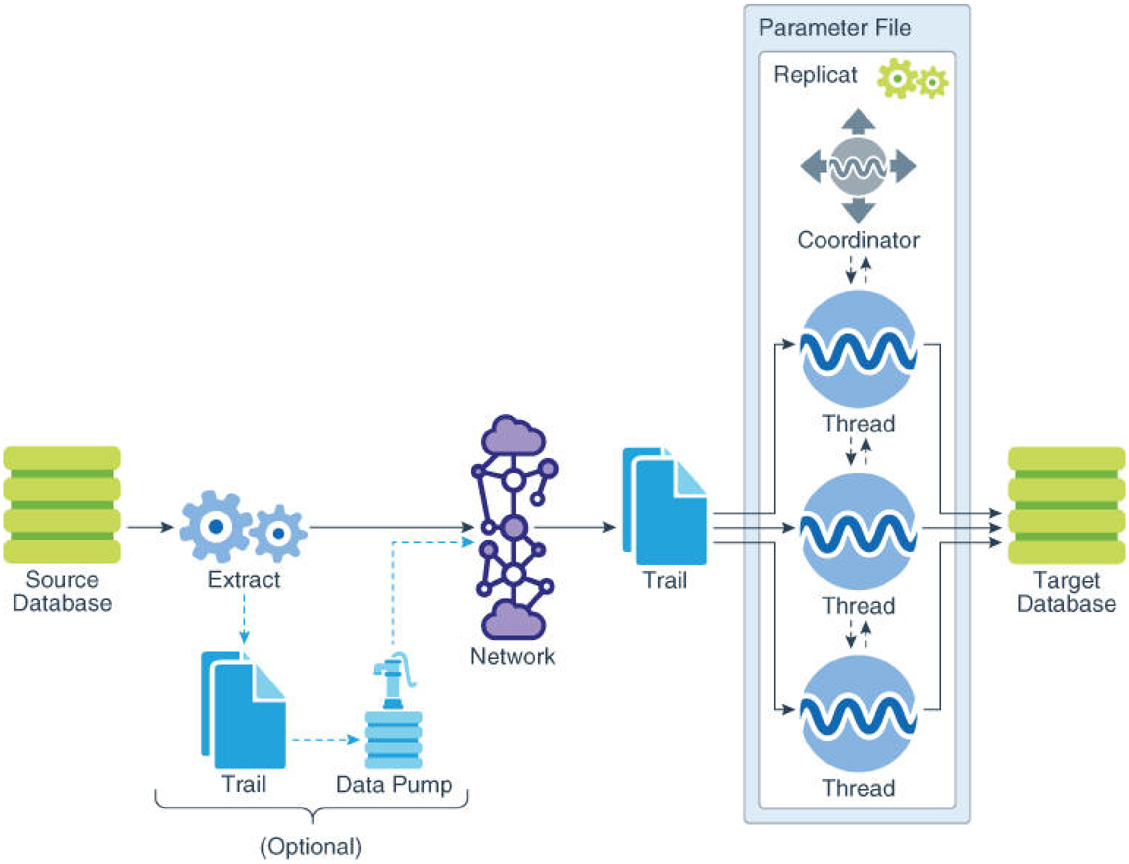
The difference between classic mode and coordinated mode is that Replicat is multi-threaded in coordinated mode. Within a single Replicat instance, multiple threads read the trail independently and apply transactions in parallel. Each thread handles the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A *coordinator* thread coordinates the transactions across threads to account for dependencies among the threads.

The source transactions could be split across CR processes such that the integrity of the total source transaction is not maintained. The portion of the transaction processed by a CR process is done in committed order but the whole transaction across all CR processes is not.

Coordinated Replicat allows for user-defined partitioning of the workload so as to apply high volume transactions concurrently. In addition, it automatically coordinates the execution of transactions that require coordination, such as DDL, and primary key updates with `THREADRANGE` partitioning. Such a transaction is executed as one transaction in the target with full synchronization: it waits until all prior transactions are applied first, and all transactions after this barrier transaction have to wait until this barrier transaction is applied.

Only one parameter file is required for a coordinated Replicat, regardless of the number of threads. You use the `THREAD` or `THREADRANGE` option in the `MAP` statement to specify which threads process the transactions for those objects, and you specify the maximum number of threads when you create the Replicat group.

Figure 6-2   Coordinated Replicat



About Barrier Transactions

Barrier transactions are managed automatically in a coordinated Replicat configuration. Barrier transactions are transactions that require coordination across threads. Examples include DDL statements, transactions that include updates to primary keys, and certain `EVENTACTIONS` actions.

Optionally, you can force other transactions to be treated like a barrier transaction through the use of the `COORDINATED` keyword in a `MAP` statement. One use case for this would be force a `SQLEXEC` to be executed in a manner similar to a serial execution. This could be beneficial if the results can become ambiguous unless the state of the target is consistent across all transactions.

> **Note:**
>
> Coordinated Replicat doesn't do dependency calculations for non-barrier transactions when a mapped table is partitioned based on `THNREADRANGE`. It relies on specified `THREADRANGE` columns to compute a hash value. It partitions the incoming data based on the hash value and sends all the records that match this hash value to same thread.

How Barrier Transactions are Processed

All threads converge and wait at the start of a barrier transaction. The barrier transaction is suspended until the other threads reach its start position. If any threads were already processing part of the barrier transaction, those threads perform a rollback. Grouped transactions, such as those controlled by the `BATCHSQL` or `GROUPTRANSOPS` parameters, are also rolled back and then reapplied until they reach the start of the barrier transaction.

All of the threads converge and wait at the start of the next transaction after the barrier transaction as well. The two synchronization points, before and after the barrier transaction, ensure that metadata operations and `EVENTACTIONS` actions all occur in the proper order relevant to the data operations.

Once the threads are synchronized at the start of the barrier transaction, the barrier transaction is processed serially by the thread that has the lowest thread ID among all of the threads specified in the `MAP` statements, and then parallel processing across threads is resumed. You can force barrier transactions to be processed through a specific thread, which is always thread 0, by specifying the `USEDEDICATEDCOORDINATIONTHREAD` parameter in the Replicat parameter file.

Using Different Replicat Modes

The recommended Oracle GoldenGate configuration, when supported by the Oracle version, is to use one Extract on an Oracle source and one parallel Replicat per source database on an Oracle target.

One integrated Replicat configuration supports all Oracle data types either through the inbound server or by switching to direct apply when necessary, and it preserves source transaction integrity. You can adjust the parallelism settings to the desired apply performance level as needed.

Each Replicat group must process objects that are suited to the processing mode, based on table data types and attributes. No objects in one Replicat can have DML or DDL dependencies on objects in the other Replicat.

If the target database is an Oracle version that does not support integrated Replicat, or if it is a non-Oracle database, you can use a coordinated or parallel Replicat configuration.

Add the Replicat Group

These steps add the Replicat group that reads the remote trail and applies the data changes to the target Oracle Database.

1. Connect to the deployment from the Admin Client using the CONNECT command.
2. If using integrated Replicat, issue the `DBLOGIN` command to log into the database.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   Where: *`alias`* specifies the alias of the database login credential that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials.
3. Issue the `ADD REPLICAT` command with the following syntax.

   ```
   ADD REPLICAT group name, [INTEGRATED,] EXTTRAIL pathname
   ```

   Where:

   - *`group name`* is the name of the Replicat group.
   - `INTEGRATED` creates an integrated Replicat group.
   - `EXTTRAIL` *`pathname`* is the relative or fully qualified name of the remote trail, including the two-character name.

     For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Example 6-1 Adds a Nonintegrated Replicat

```
ADD REPLICAT repe, EXTTRAIL east/ea
```

Example 6-2 Adds an Integrated Replicat

```
ADD REPLICAT repn, INTEGRATED, EXTTRAIL north/em
```

Creating a Parallel Replicat

You can create a parallel replication using the graphical user interface or the command line interfaces GGSCI and the Admin Client.

A parallel Replicat requires a checkpoint table so both the Administration Service UI and Admin Client issue an error when the parallel Replicat does not include a checkpoint table.

> **Note:**
>
> Parallel replication does not support `COMMIT_SERIALIZATION` in Integrated Mode. To use this apply process, use Integrated Replicat.

Creating a Non-Integrated Parallel Replicat with the Administration Service

1. Open a browser and connect to the Service Manager that you created with the Configuration Assistant:

   ```
   https://server_name:service_manger_port/
   ```

   For Example, https://localhost:9000/. In an non secured environment, use http instead of https.

   The Oracle GoldenGate Service Manager is displayed.
2. Enter the username and password you created and click **Sign In**.

   In the Service Manager, you can see servers that are running.
3. In the Services section, click **Administration Service**, and then log in.
4. Click the Application Navigation icon to the left of the page title to expand the navigation panel.
5. Create the checkpoint table by clicking **Configuration** in the right navigation panel.
6. Ensure that you have a valid credential and log in to the database by clicking the ‘log in database’ icon under **Action**.
7. Click the **+** sign to add a checkpoint table.
8. Enter the *`schema`*`.`*`name`* of the checkpoint table that you would like to create, and then click **Submit**.
9. Validate that the table was created correctly by logging out of the Credential Alias using the log out database icon, and then log back in.

   Once the log in is complete, your new checkpoint table is listed.
10. Click **Overview** to return to the main Administration Service page.
11. Click the **+** sign next to **Replicats**.
12. Select **Parallel Replicat** and then select either **Integrated** or **Nonintegrated Replicat** then click **Next**. In this document, **Nonintegrated** option is selected.
13. Enter the required information making sure that you complete the Credential Domain and Credential Alias fields before completing the Checkpoint Table field, and then select your newly created Checkpoint Table from the list.
14. Click **Next**, and then click **Create and Run** to complete the Replicat creation.

Creating a Non-Integrated Parallel Replicat with the Admin Client

1. Go the `bin` directory of your Oracle GoldenGate installation directory.

   ```
   cd $OGG_HOME/bin
   ```
2. Start the Admin Client.

   ```
   ./adminclient
   ```

The Admin Client command prompt is displayed.

```
OGG (not connected) 12>
```

3. Connect to the Service Manager deployment source:

   ```
   connect http://localhost:9500 deployment Target1 as oggadmin password welcome1
   ```

   You must use http or https in the connection string; this example is a non-SSL connection.

4. Add the Parallel Replicat, which may take a few minutes to complete:

   ```
   add replicat R1, parallel, exttrail bb checkpointtable ggadmin.ggcheckpoint
   ```

   You could use just the two character trail name as part of the `ADD REPLICAT` or you can use the full path, such as `/u01/oggdeployments/target1/var/lib/data/bb`.

5. Verify that the Replicat is running:

   ```
   info replicat R1
   ```

   Messages similar to the following are displayed:

   ```
   REPLICAT   R1        Initialized   2016-12-20 13:56   Status RUNNING
   Parallel
   Checkpoint Lag       00:00:00 (updated 00:00:22 ago)
   Process ID           30007
   Log Read
   Checkpoint  File ./ra000000000First Record  RBA 0
   ```

Configuring Oracle GoldenGate Replicat

This chapter contains instructions for configuring the Replicat apply process in either nonintegrated or integrated mode.

Prerequisites for Configuring Replicat

This topic provides the best practices for configuring Replicat.

The guidelines to follow before configuring Replicat are:

1. Preparing the Database for Oracle GoldenGate.
2. Establishing Oracle GoldenGate Credentials.
3. Using Different Replicat Modes.
4. Create the Oracle GoldenGate instance on the target system by configuring the Manager process.

Also see, Quickstart Bidirectional Replication to learn about using the active-active replication process in case of bidirectional replication.

What to Expect from these Instructions

These instructions show you how to configure a basic Replicat parameter (configuration) file.

Your business requirements probably will require a more complex topology, but this procedure forms a basis for the rest of your configuration steps.

By performing these steps, you can:

- get the basic configuration file established.
- build upon it later by adding more parameters as you make decisions about features or requirements that apply to your environment.
- use copies of it to make the creation of additional Replicat parameter files faster than starting from scratch.

> **Note:**
>
> These instructions do not configure Replicat to apply DDL to the target. To support DDL, create the basic Replicat parameter file and then see Configuring DDL Support for configuration instructions.

About Checkpoint Table

The checkpoint table is a required component of Replicat.

A Replicat maintains its recovery checkpoints in the checkpoint table, which is stored in the target database. Checkpoints are written to the checkpoint table within the Replicat transaction. Because a checkpoint either succeeds or fails with the transaction, Replicat ensures that a transaction is only applied once, even if there is a failure of the process or the database. See Before You Add a Replicat in the *Oracle GoldenGate Microservices Documentation* to learn to create checkpoint tables from the Microservices web interface.

> **Note:**
>
> Oracle recommends using checkpoint tables. Multiple classic or coordinated Replicats can share the same checkpoint table, but that may not result in the best performance. With high volume environments, you must ensure that the checkpoint tables do not reside on different drives to become a point of conflict.
>
> See #unique_565 for more information.

Adding the Checkpoint Table to the Target Database

1. From the Oracle GoldenGate directory on the target, run GGSCI and issue the `DBLOGIN` command to log into the target database.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   Where:

   - *`alias`* specifies the alias of the database login credential of a user that can create tables in a schema that is accessible to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials.

2. In GGSCI or Admin Client, create the checkpoint table in a schema of your choice (ideally dedicated to Oracle GoldenGate).

   ```
   ADD CHECKPOINTTABLE [container.]schema.table
   ```

   Where:

- `container` is the name of the container if `schema.table` is in a multitenant container database. This container can be the root container or a pluggable database that contains the table.
- `schema.table` are the schema and name of the table. See *Administering Oracle GoldenGate* for instructions for specifying object names.

Include the Checkpoint Table in the GLOBALS File

To specify the checkpoint table in the Oracle GoldenGate configuration:

1. Create a `GLOBALS` file (or edit the existing one).

   ```
   EDIT PARAMS ./GLOBALS
   ```

   > **Note:**
   >
   > `EDIT PARAMS` creates a simple text file. When you save the file after `EDIT PARAMS`, it is saved with the name `GLOBALS` in upper case, without a file extension. It must remain as such, and the file must remain in the root Oracle GoldenGate directory.

2. In the `GLOBALS` file, enter the `CHECKPOINTTABLE` parameter.

   ```
   CHECKPOINTTABLE [container.]schema.table
   ```

3. Save and close the `GLOBALS` file.

Disabling Default Asynchronous COMMIT to Checkpoint Table

When a nonintegrated Replicat uses a checkpoint table, it uses an asynchronous `COMMIT` with the `NOWAIT` option to improve performance. Replicat can continue processing immediately after applying this `COMMIT`, while the database logs the transaction in the background. You can disable the asynchronous `COMMIT` with `NOWAIT` by using the `DBOPTIONS` parameter with the `DISABLECOMMITNOWAIT` option in the Replicat parameter file.

> **Note:**
>
> When the configuration of a nonintegrated Replicat group does not include a checkpoint table, the checkpoints are maintained in a file on disk. In this case, Replicat uses `COMMIT` with `WAIT` to prevent inconsistencies in the event of a database failure that causes the state of the transaction, as in the checkpoint file, to be different than its state after the recovery.

Configuring Replicat

Configure a Replicat process to configure Replicat for a pluggable database (PDB). Replicat can operate in any of the available modes within an Oracle multitenant container database.

To add a Replicat from the command line interface, see the `ADD REPLICAT` from GGSCI.

Use the following steps to configure the parameters for different Replicat modes.

1. On the target system, create the Replicat parameter file using GGSCI command line interface.

   ```
   EDIT PARAMS name
   ```

   Where: *name* is the name of the Replicat group.

2. Enter the Replicat parameters in the order shown, starting a new line for each parameter statement.

   Basic parameters for the Replicat group in nonintegrated mode:

   ```
   REPLICAT repe
   USERIDALIAS ggeast
   ASSUMETARGETDEFS
   MAP hr.*, TARGET hr2.*;
   ```

   Basic parameters for the Replicat group in integrated Replicat mode:

   ```
   REPLICAT repw
   DBOPTIONS INTEGRATEDPARAMS(parallelism 6)
   USERIDALIAS ggwest
   ASSUMETARGETDEFS
   MAP hr.*, TARGET hr2.*;
   ```

Parameter	Description
`REPLICAT` *`group`*	*`group`* is the name of the Replicat group.
`DBOPTIONS DEFERREFCONST`	Applies to Replicat in nonintegrated mode. `DEFERREFCONST` sets constraints to `DEFERRABLE` to delay the enforcement of cascade constraints by the target database until the Replicat transaction is committed. See `DBOPTIONS` for additional important information.
`DBOPTIONS INTEGRATEDPARAMS (`*`parameter`*`[, ...])`	This parameter specification applies to Replicat in integrated mode. It specifies optional parameters for the inbound server. See Additional Parameter Options for Integrated Replicatfor additional important information about these `DBOPTIONS` options.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential of the user that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials

Parameter	Description
`MAP [container.]schema.object, TARGET schema.object;`	Specifies the relationship between a source table or sequence, or multiple objects, and the corresponding target object or objects. • `MAP` specifies the source table or sequence, or a wildcarded set of objects. • `TARGET` specifies the target table or sequence or a wildcarded set of objects. • *`container`* is the name of a container, if the source database is a multitenant container database. • *`schema`* is the schema name or a wildcarded set of schemas. • *`object`* is the name of a table or sequence, or a wildcarded set of objects. Terminate this parameter statement with a semi-colon. To exclude objects from a wildcard specification, use the `MAPEXCLUDE` parameter. For more information and for additional options that control data filtering, mapping, and manipulation, see `MAP` in *Parameters and Functions Reference for Oracle GoldenGate*.

Basic parameters for the Replicat group in parallel Replicat mode:

```
REPLICAT repe
USERID ggadmin, PASSWORD ***
MAP_PARALLELISM 3
MIN_APPLY_PARALLELISM 2
MAX_APPLY_PARALLELISM 10
SPLIT_TRANS_RECS 60000
MAP *.*, TARGET *.*;
```

Parameter	Description
`MAP_PARALLELISM`	Configures number of mappers. This controls the number of threads used to read the trail file. The minimum value is `1`, maximum value is `100` and the default value is `2`.
`APPLY_PARALLELISM`	Configures number of appliers. This controls the number of connections in the target database used to apply the changes. The default value is four.
`MIN_APPLY_PARALLELISM` `MAX_APPLY_PARALLELISM`	The Apply parallelism is auto-tuned. You can set a minimum and maximum value to define the ranges in which the Replicat automatically adjusts its parallelism. There are no defaults. Do *not* use with `APPLY_PARALLELISM` at same time.
`SPLIT_TRANS_REC`	Specifies that large transactions should be broken into pieces of specified size and applied in parallel. Dependencies between pieces are still honored. Disabled by default.
`COMMIT_SERIALIZATION`	Enables commit `FULL` serialization mode, which forces transactions to be committed in trail order.
Advanced Parameters	
`LOOK_AHEAD_TRANSACTIONS`	Controls how far ahead the Scheduler looks when batching transactions. The default value is 10000.

Parameter	Description
`CHUNK_SIZE`	Controls how large a transaction must be for parallel Replicat to consider it as large. When parallel Replicat encounters a transaction larger than this size, it will serialize it, resulting in decreased performance. However, increasing this value will also increase the amount of memory consumed by parallel Replicat.

3. If using integrated Replicat or parallel Replicat in integrated mode, add the following parameters to the Extract parameter file:
   - `LOGALLSUPCOLS`: This parameter ensures the capture of the supplementally logged columns in the before image. It's the default parameter and shouldn't be turned off or disabled. It is valid for any source database that is supported by Oracle GoldenGate. For Extract versions older than 12c, you can use `GETUPDATEBEFORES` and `NOCOMPRESSDELETES` parameters to satisfy the same requirement. The database must be configured to log the before and after values of the primary key, unique indexes, and foreign keys.
   - The `UPDATERECORDFORMAT` parameter set to COMPACT: This setting causes Extract to combine the before and after images of an `UPDATE` operation into a single record in the trail. This is the default option and it is recommended that you don't change the default setting.
4. Enter any optional Replicat parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command. See Additional Parameter Options for Integrated Replicat for additional configuration considerations.
5. Save and close the file.

Additional Parameter Options for Integrated Replicat

You can set these parameters by using the `DBOPTIONS` parameter with the `INTEGRATEDPARAMS` option or dynamically by issuing the `SEND REPLICAT` command with the `INTEGRATEDPARAMS` option.

The default Replicat configuration as directed in Configuring Oracle GoldenGate Replicat should be sufficient. However, if needed, you can set the following inbound server parameters to support specific requirements.

> **Note:**
>
> See *Parameters and Functions Reference for Oracle GoldenGate* for more information about the `DBOPTIONS` parameter.

- `COMMIT_SERIALIZATION`: Controls the order in which applied transactions are committed and has 2 modes, `DEPENDENT_TRANSACTIONS` and `FULL`. The default mode for Oracle GoldenGate is `DEPENDENT_TRANSACTIONS` where dependent transactions are applied in the correct order though may not necessarily be applied in source commit order. In `FULL` mode, the source commit order is enforced when applying transactions.
- `BATCHSQL_MODE`: Controls the batch execution scheduling mode including pending dependencies. A pending dependency is a dependency on another transaction that has

already been scheduled, but not completely executed. The default is `DEPENDENT`. You can use following three modes:

`DEPENDENT`

Dependency aware scheduling without an early start. Batched transactions are scheduled when there are no pending dependencies.

`DEPENDENT_EAGER`

Dependency aware batching with early start. Batched transactions are scheduled irrespective of pending dependencies.

`SEQUENTIAL`

Sequential batching. Transactions are batched by grouping the transactions sequentially based on the original commit order.

- `DISABLE_ON_ERROR`: Determines whether the apply server is disabled or continues on an unresolved error. The default for Oracle GoldenGate is `N` (continue on errors), however, you can set the option to Y if you need to disable the apply server when an error occurs.
- `EAGER_SIZE`: Sets a threshold for the size of a transaction (in number of LCRs) after which Oracle GoldenGate starts applying data before the commit record is received. The default for Oracle GoldenGate is `15100`.
- `ENABLE_XSTREAM_TABLE_STATS`: Controls whether statistics on applied transactions are recorded in the `V$GOLDENGATE_TABLE_STATS` view or not collected at all. The default for Oracle GoldenGate is `Y` (collect statistics).
- `MAX_PARALLELISM`: Limits the number of apply servers that can be used when the load is heavy. This number is reduced again when the workload subsides. The automatic tuning of the number of apply servers is effective only if `PARALLELISM` is greater than 1 and `MAX_PARALLELISM` is greater than `PARALLELISM`. If `PARALLELISM` is equal to `MAX_PARALLELISM`, the number of apply servers remains constant during the workload. The default for Oracle GoldenGate is 50.
- `MAX_SGA_SIZE`: Controls the amount of shared memory used by the inbound server. The shared memory is obtained from the streams pool of the SGA. The default for Oracle GoldenGate is `INFINITE`.
- `MESSAGE_TRACKING_FREQUENCY`: Controls how often LCRs are marked for high-level LCR tracing through the apply processing. The default value is `2000000`, meaning that every 2 millionth LCR is traced. A value of zero (`0`) disables LCR tracing.
- `PARALLELISM`: Sets a minimum number of apply servers that can be used under normal conditions. Setting `PARALLELISM` to 1 disables apply parallelism, and transactions are applied with a single apply server process. The default for Oracle GoldenGate is `4`. For Oracle Standard Edition, this must be set to `1`.
- `PARALLELISM_INTERVAL`: Sets the interval in seconds at which the current workload activity is computed. Replicat calculates the mean throughput every 5 X `PARALLELISM_INTERVAL` seconds. After each calculation, the apply component can increase or decrease the number of apply servers to try to improve throughput. If throughput is improved, the apply component keeps the new number of apply servers. The parallelism interval is used only if `PARALLELISM` is set to a value greater than one and the `MAX_PARALLELISM` value is greater than the `PARALLELISM` value. The default is 5 seconds.
- `PRESERVE_ENCRYPTION`: Controls whether to preserve encryption for columns encrypted using Transparent Data Encryption. The default for Oracle GoldenGate is `N` (do not apply the data in encrypted form).

- `TRACE_LEVEL`: Controls the level of tracing for the Replicat inbound server. For use only with guidance from Oracle Support. The default for Oracle GoldenGate is `0` (no tracing).
- `WRITE_ALERT_LOG`: Controls whether the Replicat inbound server writes messages to the Oracle alert log. The default for Oracle GoldenGate is `Y` (yes).

Next Steps After Configuring Replicat

After you have created a basic parameter file for Replicat, see the following for additional configuration steps.

Configuring Extract if you have not configured Extract yet.

Creating a Parallel Replicat if you want to create integrated or nonintegrated parallel Replicats for your deployment.
Additional Configuration Steps For Using Nonintegrated Replicat (if using nonintegrated Replicat)

Additional Oracle GoldenGate Configuration for Your Database

Configuring DDL Support (to use Oracle GoldenGate DDL support)

#unique_565

Additional Configuration Steps For Using Nonintegrated Replicat

This chapter contains instructions that are specific only to Replicat when operating in *nonintegrated* mode.
When Replicat operates in nonintegrated mode, triggers, cascade constraints, and unique identifiers must be properly configured in an Oracle GoldenGate environment.

Disabling Triggers and Referential Cascade Constraints on Target Tables

Triggers and cascade constraints must be disabled on Oracle target tables when Replicat is in nonintegrated mode.

Constraints must be disabled in nonintegrated Replicat mode because Oracle GoldenGate replicates DML that results from the firing of a trigger or a cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

Understanding Replicat Processing in Relation to Parameter Changes

Changes to the object specifications in the Replicat configuration cannot be made to affect transactions that are already applied, but only for those not yet applied. This is an important consideration when using coordinated or integrated Replicat.

For a Replicat in classic mode, the boundary between applied and non-applied transactions is a clean one, because transactions are applied serially. For a coordinated or integrated Replicat, however, there is no single point in the trail that marks applied and unapplied transactions, because transactions are being applied asynchronously in parallel.

In coordinated or integrated modes, there are a low watermark, below which all transactions were applied, and a high watermark above which no transactions were applied. In between those boundaries there may be transactions that may or may not have been applied, depending on the progress of individual threads. As a result, if Replicat is forced changes to object specifications in the Replicat configuration may be reflected unevenly in the target after Replicat is restarted. Examples of parameter changes for which this applies are changes to `MAP` mappings, `FILTER` clauses, and `EXCLUDE` parameters.

Changes to the Replicat configuration should not be made after Replicat abends or is forcibly terminated. Replicat should be allowed to recover to its last checkpoint after startup. For coordinated Replicat, you can follow the administrative procedures in Administering a Coordinated Replicat Configuration. Once the recovery is complete, Replicat can be shut down gracefully with the `STOP REPLICAT` command, and then you can make the changes to the object specifications.

Controlling Extract and Replicat

Here are basic directions for controlling Extract and Replicat processes.

To Start Extract or Replicat

```
START {EXTRACT | REPLICAT} group_name
```

Where:

`group_name` is the name of the Extract or Replicat group or a wildcard set of groups (for example, `*` or `fin*`).

To Stop Extract or Replicat Gracefully

```
STOP {EXTRACT | REPLICAT} group_name
```

Where:

`group_name` is the name of the Extract or Replicat group or a wildcard set of groups (for example, `*` or `fin*`).

To Stop Replicat Forcefully

```
STOP REPLICAT group_name !
```

The current transaction is aborted and the process stops immediately. You cannot stop Extract forcefully.

To End a Process that STOP Cannot Stop

```
KILL {EXTRACT | REPLICAT} group_name
```

Ending a process does not shut it down gracefully, and checkpoint information can be lost.

To Control Multiple Processes at Once

```
command ER wildcard specification
```

Where:

- *`command`* `is: KILL, START, or STOP`
- *`wildcard specification`* is a wildcard specification for the names of the process groups that you want to affect with the command. The command affects every Extract and Replicat group that satisfies the wildcard. Oracle GoldenGate supports up to 100,000 wildcard entries.

Deleting Extract and Replicat

This section contains basic directions for deleting Extract and Replicat processes. See *Parameters and Functions Reference for Oracle GoldenGate* for additional command options.

To Delete an Extract Group

1. Run GGSCI.
2. Issue the `DBLOGIN` command as the Extract database user (or a user with the same privileges). You can use either of the following commands, depending on whether a local credential store exists.

   ```
   DBLOGIN [SOURCEDB dsn] {USERID user, PASSWORD password [encryption_options] |
   USERIDALIAS alias [DOMAIN domain]}
   ```
3. Stop the Extract process.

   ```
   STOP EXTRACT group_name
   ```
4. Issue the following command.

   ```
   DELETE EXTRACT group_name
   ```
5. (Oracle) Unregister the Extract group from the database.

   ```
   UNREGISTER EXTRACT group_name,database_name
   ```

To Delete a Replicat Group

1. Stop the Replicat process.

   ```
   STOP REPLICAT group_name
   ```
2. Issue one of the following commands from GGSCI to log into the database.

   ```
   DBLOGIN [SOURCEDB dsn] {USERID user, PASSWORD password [encryption_options] |
   USERIDALIAS alias [DOMAIN domain]}
   ```

Where:

- `SOURCEDB dsn` supplies the data source name, if required as part of the connection information.
- `USERID user`, `PASSWORD password` specifies an explicit database login credential.
- `USERIDALIAS alias [DOMAIN domain]` specifies an alias and optional domain of a credential that is stored in a local credential store.
- `encryption_options` is one of the options that encrypt the password.

3. Issue the following command to delete the group.

```
DELETE REPLICAT group_name
```

Deleting a Replicat group preserves the checkpoints in the checkpoint table (if being used). Deleting a process group also preserves the parameter file. You can create the same group again, using the same parameter file, or you can delete the parameter file to remove the group's configuration permanently.

About the Global Watermark

A clean shutdown of a Replicat ensures that all threads stop at the same transaction boundary in the trail, known as the *global watermark*. This is defined as the synchronized point where all records before this position were either committed or ignored by all of their respective threads. If a clean shutdown is not possible, you can use the `SYNCHRONIZE REPLICAT` command to return all of the threads to the position of the thread that made the most recent checkpoint. This command is valid for coordinated, integrated, and parallel Replicats. See Synchronizing Threads After an Unclean Stop for more information about recovering a coordinated Replicat group.

> **Note:**
>
> Coordinated Replicat is an online process only. Do not use it to perform initial loads.

7

Instantiate

Learn about instantiation prerequisites and steps for Oracle GoldenGate.

Instantiating Oracle GoldenGate Using Initial Load

You can use Oracle GoldenGate to:

- Perform a standalone batch load to populate database tables for migration or other purposes.
- Load data into database tables as part of an initial synchronization run in preparation for change synchronization with Oracle GoldenGate.

If you are working with an Oracle to Oracle replication, there are optimized methods because the instantiation has the highest precision based on the SCN value. In this case, the `HANDLECOLLISIONS` parameter isn't required.

See `HANDLECOLLISIONS | NOHANDLECOLLISIONS`.

For non-Oracle environments, traditional methods are used for the initial load.

Prerequisites for Initial Load

Verify that you meet the prerequisites for executing an initial load that are described in the following sections.

Disable DDL Processing

Before executing an initial load, disable DDL extraction and replication. DDL processing is controlled by the `DDL` parameter in the Extract and Replicat parameter files.

Prepare the Target Tables

The following are suggestions that can make the load go faster and help you to avoid errors.

- **Data:** Make certain that the target tables are empty. Otherwise, there may be duplicate-row errors or conflicts between existing rows and rows that are being loaded.
- **Constraints:** Disable foreign-key constraints and check constraints. Foreign-key constraints can cause errors, and check constraints can slow down the loading process. Constraints can be reactivated after the load concludes successfully.
- **Indexes:** Remove indexes from the target tables. Indexes are not necessary for inserts. They will slow down the loading process significantly. For each row that is inserted into a table, the database will update every index on that table. You can add back the indexes after the load is finished.

> **Note:**
>
> A primary index is required for all applications that access DB2 for z/OS target tables. You can delete all other indexes from the target tables, except for the primary index.

- **Keys:** For Oracle GoldenGate to reconcile the replicated incremental data changes with the results of the load, each target table must have a primary or unique key. If you cannot create a key through your application, use the `KEYCOLS` option of the `TABLE` and `MAP` parameters to specify columns as a substitute key for Oracle GoldenGate's purposes. A key helps identify which row to process. If you cannot create keys, the source database must be quiesced for the load.

Configure the Manager Process

On the source and target systems, configure and start a Manager process. One Manager can be used for the initial-load processes and the change-synchronization processes. For enhanced security, the target manager parameter file should have the following parameter for `RMTTASK` to access Replicat on target:

```
ACCESSRULE, PROG *, IPADDR *, ALLOW
```

`RMTTASK` is only allowed to be used once in the Extract parameter file.

Create a Data-definitions File

A data-definitions file is required if the source and target databases have dissimilar definitions. Oracle GoldenGate uses this file to convert the data to the format required by the target database.

Create Change-synchronization Groups

To prepare for the capture and replication of transactional changes during the initial load, create online Extract and Replicat groups. You will start these groups during the load procedure. See Configuring Online Change Synchronization for more information.

> **Note:**
>
> If the load is performed from a quiet source database and *will not* be followed by continuous change synchronization, you can omit these groups.

Do not start the Extract or Replicat groups until instructed to do so in the initial-load instructions. Change synchronization keeps track of transactional changes while the load is being applied, and then the target tables are reconciled with those changes.

> **Note:**
>
> The first time that Extract starts in a new Oracle GoldenGate configuration, any open transactions will be skipped. Only transactions that begin after Extract starts are captured.

Sharing Parameters between Process Groups

Some of the parameters that you use in a change-synchronization parameter file also are required in an initial-load Extract and initial-load Replicat parameter file. You can copy those parameters from one parameter file to another, or you can store them in a central file and use the `OBEY` parameter in each parameter file to retrieve them. Alternatively, you can create an Oracle GoldenGate macro for the shared parameters and then call the macro from each parameter file with the `MACRO` parameter.

See Getting Started with the Oracle GoldenGate Process Interfaces for more information about using `OBEY` and using macros.

Instantiation Requirements for DB2 LUW

During the initialization of the Oracle GoldenGate environment, you will be doing an initial data synchronization and starting the Oracle GoldenGate processes for the first time. In conjunction with those procedures, you will be creating process groups. To create an Extract group, an initial start position must be established in the transaction log. This initial read position is on a transaction boundary that is based on one of the following:

- End of the transaction file
- A specific LRI value

The start point is specified with the `BEGIN` option of the `ADD EXTRACT` command.

When the Extract process starts for the first time, it captures all the transaction data that it encounters after the specified start point, but none of the data that occurred *before* that point. This can cause partial transactions to be captured if open transactions span the start point.

To ensure initial transactional consistency:

To avoid the capture of partial transactions, initialize the Extract process at a point in time when the database is in a paused state. DB2 LUW provides a `QUIESCE` command for such a purpose. This is the only way to ensure transactional consistency.

> **Note:**
>
> After the Extract is past the initialization, subsequent restarts of the Extract do not extract partial transactions, because the process uses recovery checkpoints to mark its last read position.

To view open transactions:

IBM provides a utility called `db2pd` for monitoring DB2 databases and instances. You can use it to view information about open transactions and to determine if any of them span the start point. However, because DB2 LUW log records lack timestamps, it might not be possible to make an accurate assessment. If possible, quiesce the database prior to initialization of Oracle GoldenGate.

Improving the Performance of an Initial Load

For all initial load methods except those performed with a database utility, you can load large databases more quickly by using parallel Oracle GoldenGate processes. To use parallel processing, take the following steps.

1. Follow the directions in this chapter for creating an initial-load Extract and an initial-load Replicat for each set of parallel processes that you want to use.
2. With the `TABLE` and `MAP` parameters, specify a different set of tables for each pair of Extract-Replicat processes, or you can use the `SQLPREDICATE` option of `TABLE` to partition the rows of large tables among the different Extract processes.

For all initial load methods, testing has shown that using the `TCPBUFSIZE` option in the `RMTHOST` parameter produced three times faster throughput than loads performed without it. Do not use this parameter if the target system is NonStop.

Loading Data with Oracle Data Pump

This method uses the Oracle Data Pump utility to establish the target data. After you apply the copy to the target, you record the SCN at which the copy stopped. Transactions that were included in the copy are skipped to avoid collisions from integrity violations. With the data pump method, Replicat has the information about the consistent SCN from the export of each table. Replicat will ignore changes that belongs to transactions up to this SCN. Transactions after this SCN will be applied. No initial-load Oracle GoldenGate processes are required for these methods.

Using Automatic Per Table Instantiation

You can automatically instantiate per table CSN filtering for Oracle Database with Oracle data pump, which avoids having all of your tables at the same SCN.

On the Source Database

1. Use `ADD TRANDATA` and `ADD SCHEMATRANDATA`. `ADD TRANDATA/SCHEMATRANDATA.PREPARECSN` automatically prepares the tables at the source so the Oracle data pump export dump file includes instantiation CSNs. Replicat uses the per table instantiation CSN set by the Oracle data pump (on import) to filter out trail records.

   Use `INFO TRANDATA` to make sure that your table is prepared for instantiation and at what point it was done. Here's a sample of the report file:

   ```
   2016-09-29 15:30:00 INFO OGG-10154 Schema level PREPARECSN set to mode
   NOWAIT on schema
         SCOTT
   ```

2. Stop Replicat on the target database.
3. Start Extract with the correct `TABLE` statement.

The `EXPORT datapump` option `FLASHBACK_SCN` is not needed as the tables have been prepared earlier.

On the Target Database

1. Import your exported tables using Oracle data pump, which populates system tables and views with instantiation SCNs, as well as the specified table data.
2. Start Replicat using one of the following:

   Set the `DBOPTIONS ENABLE_INSTANTIATION_FILTERING` parameter in the Replicat parameter file to enable table-level instantiation filtering.

   You can remove this parameter when replicat has processed all transactions beyond the instantiation SCN.

   For all other Replicats, set the `DBOPTIONS` *`source_dbase_name global_name`* parameter in the Replicat parameter file where *`global_name`* is the global name of the Oracle source database that the trail is coming from.

   > **Note:**
   >
   > When the source has no `DOMAIN`, do not specify a `DOMAIN` for the downstream database.

   Replicat queries the instantiation SCN on any new mapping and filter records accordingly. For example, see the following report file output:

   ```
   2015-06-29 17:12:39 INFO OGG-10155 Oracle GoldenGate Delivery for Oracle,
   r1.prm:
   Instantiation CSN filtering is enabled on table SCOTT.EMP at CSN 1,851,797.
   ```

You can use other methods for instantiation instead of using the data pump to export and import tables also. One such method is using the `create table` as a `select` command or RMAN. It's steps are:

1. Use `create table` with an `at SCN of` parameter, using the following command:

   `SET_INSTANTIATION_CSN` *`SCN`* `for` *`object`* `from` *`global_name`*

   For example:

   ```
   SET_INSTANTIATION_CSN 1 FOR u1.t1 FROM DBS1.REGRESS.RDBMS.DEV.US.ORACLE.COM
   ```

2. If you want to remove the manual setting of the instantiation CSN later, you can use the following command:

   ```
   CLEAR_INSTANTIATION_CSN for object from global_name
   ```

Using Oracle Data Pump Table Instantiation

To perform instantiation with Oracle Data Pump, see My Oracle Support document 1276058.1. To obtain this document, do the following:

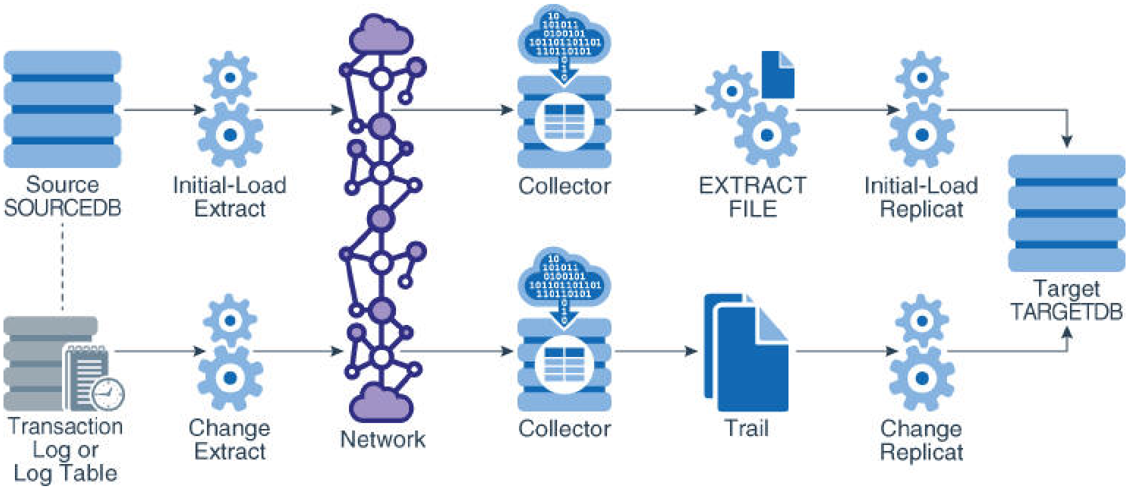
1. Go to http://support.oracle.com.
2. Under Sign In, select your language and then log in with your Oracle Single Sign-On (SSO).

3. On the Dashboard, expand the Knowledge Base heading.
4. Under Enter Search Terms, paste or type the document ID of `1276058.1` and then click **Search**.
5. In the search results, select **Oracle GoldenGate Best Practices: Instantiation from an Oracle Source Database [Article ID 1276058.1]**.
6. Click the link under Attachments to open the article.

Loading Data from File to Replicat

To use Replicat to establish the target data, you use an initial-load Extract to extract source records from the source tables and write them to an extract file in canonical format. From the file, an initial-load Replicat loads the data using the database interface. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.

During the load, the records are applied to the target database one record at a time, so this method is considerably slower than any of the other initial load methods. This method permits data transformation to be done on either the source or target system.



You can also use the Microservices Architecture to load data from file to Replicat. See Instantiating Oracle GoldenGate Using Initial Load.

To Load Data From File to Replicat

1. Make certain that you have addressed the requirements in Prerequisites for Initial Load.
2. On the source and target systems, run GGSCI and start Manager.

```
START MANAGER
```

> **Note:**
>
> In a Windows cluster, start the Manager resource from the Cluster Administrator.

3. On the source system, issue the following command to create an initial-load Extract parameter file.

```
EDIT PARAMS initial-load_Extract
```

4. Enter the parameters in the same order as shown in the following example, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for loading data from file to Replicat.

```
SOURCEISTABLE
SOURCEDB mydb, USERIDALIAS ogg
RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
ENCRYPTTRAIL AES192
RMTFILE /ggs/dirdat/initld, MEGABYTES 2, PURGE
TABLE hr.*;
TABLE sales.*;
```

Parameter	Description
`SOURCEISTABLE`	Designates Extract as an initial load process extracting records directly from the source tables.
`SOURCEDB` *`dsn`* `[, USERIDALIAS` *`alias, options`* `| , USERID` *`user, options`*`]`	Specifies database connection information. `SOURCEDB` specifies the source data source name (DSN). `USERID` and `USERIDALIAS` specify database credentials if required.

`RMTHOSTOPTIONS` *`hostname`*`,`
`MGRPORT` *`portnumber`*
`[, ENCRYPT` *`algorithm`* `KEYNAME` *`keyname`*`]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`ENCRYPTTRAIL` *`algorithm`*	Encrypts the data in the remote file.

`RMTFILE` *`path`*`,`
`[MEGABYTES` *`n`*`]`
• *`path`* is the relative or fully qualified name of the file.
• `MEGABYTES` designates the size of each file.
	Specifies the extract file to which the load data will be written. Oracle GoldenGate creates this file during the load. Checkpoints are not maintained with `RMTFILE`.
	Note that the size of an extract file cannot exceed 2GB.

`TABLE` *`container`*`.owner.`*`object`*`;`	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container or catalog unless `SOURCECATALOG` is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`TABLEEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated `TABLE` statement.

5. Enter any appropriate optional Extract parameters listed in the *Parameters and Functions Reference for Oracle GoldenGate*.

6. Save and close the parameter file.

7. On the target system, issue the following command to create an initial-load Replicat parameter file.

```
EDIT PARAMS initial-load_Replicat
```

8. Enter the parameters listed in Table 7-1 in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for loading data from file to Replicat.

```
SPECIALRUN
END RUNTIME
TARGETDB mydb, USERIDALIAS ogg
EXTFILE /ggs/dirdat/initld
SOURCEDEFS /ggs/dirdef/source_defs
MAP hr.*, TARGET hr.*;
MAP sales.*, TARGET hr.*;
```

Table 7-1 Initial-load Replicat parameters

Parameter	Description
`SPECIALRUN`	Implements the initial-load Replicat as a one-time run that does not use checkpoints.
`END RUNTIME`	Directs the initial-load Replicat to terminate when the load is finished.

`TARGETDB` *`dsn`*
`[, USERIDALIAS` *`alias, options`* `|`
`, USERID` *`user, options`*`]`
	Specifies database connection information.
	`TARGETDB` specifies the target data source name (DSN).
	`USERID` and `USERIDALIAS` specify database credentials if required.

`EXTFILE` *`path`* - *`path`* is the relative or fully qualified name of the file.	Specifies the input extract file specified with the Extract parameter `RMTFILE`.

`{SOURCEDEFS` *`file`*`} |`
`ASSUMETARGETDEFS`
- Use `SOURCEDEFS` if the source and target tables have different definitions. Specify the relative or fully qualified name of the source-definitions file generated by DEFGEN.
- Use `ASSUMETARGETDEFS` if the source and target tables have the same definitions.
	Specifies how to interpret data definitions.

`SOURCECATALOG`	Specifies a default source Oracle container. Enables the use of two-part names (*`schema.object`*) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of `MAP` parameters.

Table 7-1 (Cont.) Initial-load Replicat parameters

Parameter	Description

`MAP container.owner.object,`
`TARGET owner.object[, DEF template]`
`;`
	Specifies a relationship between a source object or objects and a target object or objects. `MAP` specifies the source object, and `TARGET` specifies the target object.
	For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container or catalog unless `SOURCECATALOG` is used.
	For the target object, specify only the *`owner.object`* components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to load data.
	See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
	The `DEF` option specifies a definitions template.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`MAPEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated `MAP` statement..

9. Enter any appropriate optional Replicat parameters listed in the *Parameters and Functions Reference for Oracle GoldenGate*.
10. Save and close the file.
11. View the Replicat parameter file to make certain that the `HANDLECOLLISIONS` parameter is listed. If not, add the parameter to the file.
12. On the source system, start change extraction.

    ```
    START EXTRACT group
    ```

13. (Oracle, if replicating sequences) Issue the `DBLOGIN` command as the user who has `EXECUTE` privilege on `update.Sequence`.

    ```
    GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
    ```

14. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).

    ```
    FLUSH SEQUENCE owner.sequence
    ```

15. From the directory where Oracle GoldenGate is installed on the source system, start the initial-load Extract.

    UNIX and Linux:

    ```
    $ /GGS directory/extract paramfile dirprm/initial-load_Extract.prm reportfile
    path
    ```

    Windows:

    ```
    C:\> GGS directory\extract paramfile dirprm\initial-load_Extract.prm reportfile
    path
    ```

Where:

`initial-load_Extract` is the name of the initial-load Extract that you used when creating the parameter file, and *`path`* is the relative or fully qualified name of the Extract report file.

16. Verify the progress and results of the initial extraction by viewing the Extract report file using the operating system's standard method for viewing files.
17. Wait until the initial extraction is finished.
18. On the target system, start the initial-load Replicat.

    UNIX and Linux:

    ```
    $ /GGS directory/replicat paramfile dirprm/initial-load_Replicat.prm reportfile
    path
    ```

    Windows:

    ```
    C:\> GGS directory\replicat paramfile dirprm\initial-load_Replicat.prm reportfile
    path
    ```

    Where:

    `initial-load_Replicat` is the name of the initial-load Replicat that you used when creating the parameter file, and *`path`* is the relative or fully qualified name of the Replicat report file.

19. When the initial-load Replicat is finished running, verify the results by viewing the Replicat report file using the operating system's standard method for viewing files.
20. On the target system, start change replication.

    ```
    START REPLICAT group
    ```

21. On the target system, issue the following command to verify the status of change replication.

    ```
    INFO REPLICAT group
    ```

22. Continue to issue the `INFO REPLICAT` command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.
23. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

    ```
    SEND REPLICAT group, NOHANDLECOLLISIONS
    ```

24. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

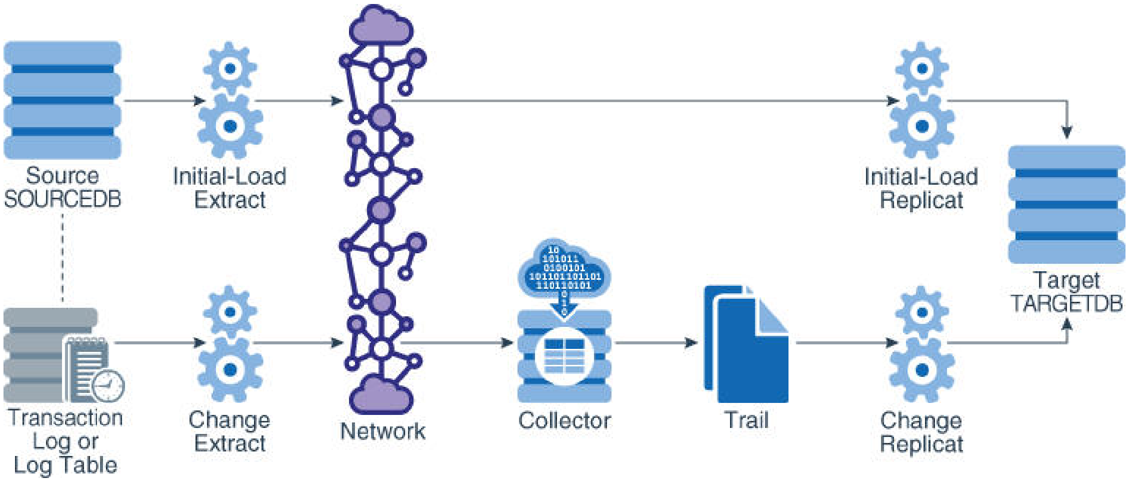
    > **Caution:**
    >
    > Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

25. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Loading Data with an Oracle GoldenGate Direct Load

To use an Oracle GoldenGate direct load, you run an Oracle GoldenGate initial-load Extract to extract the source records and send them directly to an initial-load Replicat *task*. A task is started dynamically by the Manager process and does not require the use of a Collector process or file. The initial-load Replicat task delivers the load in large blocks to the target database. Transformation and mapping can be done by Extract, Replicat, or both. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.



To control which port is used by Replicat, and to speed up the search and bind process, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Manager passes the list of port numbers that are specified with this parameter to the Replicat task process. Replicat first searches for a port from this list, and only if no ports are available from the list does Replicat begin scanning in ascending order from the default Manager port number until it finds an available port.

This method supports standard character, numeric, and datetime data types, as well as `CLOB`, `NCLOB`, `BLOB`, `LONG`, `XML`, and user-defined datatypes (`UDT`) embedded with the following attributes: `CHAR`, `NCHAR`, `VARCHAR`, `NVARCHAR`, `RAW`, `NUMBER`, `DATE`, `FLOAT`, `TIMESTAMP`, `CLOB`, `BLOB`, `XML`, and `UDT`. Character sets are converted between source and target where applicable.

This method supports Oracle internal tables, but does not convert between the source and target character sets during the load.

To Load Data with an Oracle GoldenGate Direct Load

1. Make certain to satisfy "Prerequisites for Initial Load".
2. On the source and target systems, run GGSCI and start Manager.

   ```
   START MANAGER
   ```

> **Note:**
>
> In a Windows cluster, start the Manager resource from the Cluster Administrator.

3. On the source, issue the following command to create the initial-load Extract.

   ```
   ADD EXTRACT initial-load_Extract, SOURCEISTABLE
   ```

   Where:

   - `initial-load_Extract` is the name of the initial-load Extract, up to eight characters.
   - `SOURCEISTABLE` designates Extract as an initial-load process that reads complete records directly from the source tables. Do not use any of the other `ADD EXTRACT` service options or datasource arguments.

4. On the source system, issue the following command to create an initial-load Extract parameter file.

   ```
   EDIT PARAMS initial-load_Extract
   ```

5. Enter the parameters listed in Table 7-2 in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for an Oracle GoldenGate direct load.

   ```
   EXTRACT initext
   SOURCEDB mydb, USERIDALIAS ogg
   RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
   RMTTASK REPLICAT, GROUP initrep
   TABLE hr.*;
   TABLE sales.*;
   ```

Table 7-2 Initial-load Extract Parameters for Oracle GoldenGate Direct Load

Parameter	Description
`EXTRACT initial-load_Extract`	Specifies the initial-load Extract.

`SOURCEDB dsn`
`[, USERIDALIAS alias, options |`
`, USERID user, options]`
	Specifies database connection information.
	`SOURCEDB` specifies the source datasource name (DSN). See *Parameters and Functions Reference for Oracle GoldenGate* for more information.
	`USERID` and `USERIDALIAS` specify database credentials if required.

`RMTHOSTOPTIONS hostname,`
`MGRPORT portnumber`
`[, ENCRYPT algorithm KEYNAME keyname]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`RMTTASK replicat,`
`GROUP initial-load_Replicat`
• `initial-load_Replicat` is the name of the initial-load Replicat group
	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.

`TABLE container.owner.object;`	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant database, the object name must include the name of the container or catalog unless `SOURCECATALOG` is used.

Table 7-2 (Cont.) Initial-load Extract Parameters for Oracle GoldenGate Direct Load

Parameter	Description

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`TABLEEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated `TABLE` statement. See *Parameters and Functions Reference for Oracle GoldenGate* for details.

6. Enter any appropriate optional Extract parameters listed in *Parameters and Functions Reference for Oracle GoldenGate*.
7. Save and close the file.
8. On the target system, issue the following command to create the initial-load Replicat task.

   ```
   ADD REPLICAT initial-load_Replicat, SPECIALRUN
   ```

   Where:

   - *`initial-load_Replicat`* is the name of the initial-load Replicat task.
   - `SPECIALRUN` identifies the initial-load Replicat as a one-time run, not a continuous process.
9. On the target system, issue the following command to create an initial-load Replicat parameter file.

   ```
   EDIT PARAMS initial-load_Replicat
   ```
10. Enter the parameters listed in Table 7-3 in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for an Oracle GoldenGate direct load.

    ```
    REPLICAT initrep
    TARGETDB mydb, USERIDALIAS ogg
    SOURCEDEFS /ggs/dirdef/source_defs
    MAP hr.*, TARGET hr.*;
    MAP sales.*, TARGET hr.*;
    ```

Table 7-3 Initial-load Replicat parameters for Oracle GoldenGate Direct Load

Parameter	Description
`REPLICAT` *`initial-load_Replicat`*	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat.

`[TARGETDB` *`dsn`* `|` *`container`*`]`
`[, USERIDALIAS` *`alias, options`* `|`
`, USERID` *`user, options`*`]`
	Specifies database connection information.
	`TARGETDB` specifies the target datasource name (DSN) or Oracle container. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.
	`USERID` and `USERIDALIAS` specify database credentials if required.

Table 7-3 (Cont.) Initial-load Replicat parameters for Oracle GoldenGate Direct Load

Parameter	Description
`{SOURCEDEFS` *`full_pathname`*`} | ASSUMETARGETDEFS` - Use `SOURCEDEFS` if the source and target tables have different definitions. Specify the source-definitions file generated by DEFGEN. - Use `ASSUMETARGETDEFS` if the source and target tables have the same definitions.	Specifies how to interpret data definitions.
`SOURCECATALOG`	Specifies a default source Oracle container . Enables the use of two-part names (*`schema.object`*) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of `MAP` parameters.

`MAP` *`container.owner.object`*`,`
`TARGET` *`owner.object`*`[, DEF` *`template`*`]`
`;`
	Specifies a relationship between a source object or objects and a target object or objects. `MAP` specifies the source object, and `TARGET` specifies the target object.
	For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container or catalog unless `SOURCECATALOG` is used.
	For the target object, specify only the *`owner.object`* components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to load data.
	See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
	The `DEF` option specifies a definitions template.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`MAPEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated `MAP` statement. See *Parameters and Functions Reference for Oracle GoldenGate* for details.

11. Enter any appropriate optional Replicat parameters listed in the *Parameters and Functions Reference for Oracle GoldenGate*.
12. Save and close the parameter file.
13. On the source system, start change extraction.

    ```
    START EXTRACT group
    ```

14. View the Replicat parameter file to make certain that the `HANDLECOLLISIONS` parameter is listed. If not, add the parameter to the file.
15. (Oracle, if replicating sequences) Issue the `DBLOGIN` command as the user who has `EXECUTE` privilege on `update.Sequence`.

    ```
    GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
    ```

16. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of

the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).

```
FLUSH SEQUENCE owner.sequence
```

17. On the source system, start the initial-load Extract.

    ```
    START EXTRACT initial-load_Extract
    ```

    > **Note:**
    >
    > Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

18. On the target system, issue the following command to find out if the load is finished. Wait until the load is finished before going to the next step.

    ```
    VIEW REPORT initial-load_Replicat
    ```

19. On the target system, start change replication.

    ```
    START REPLICAT group
    ```

20. On the target system, issue the following command to verify the status of change replication.

    ```
    INFO REPLICAT group
    ```

21. Continue to issue the `INFO REPLICAT` command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.
22. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

    ```
    SEND REPLICAT group, NOHANDLECOLLISIONS
    ```

23. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

    > **Caution:**
    >
    > Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

24. Save and close the parameter file. From this point forward, Oracle GoldenGate continues to synchronize data changes.

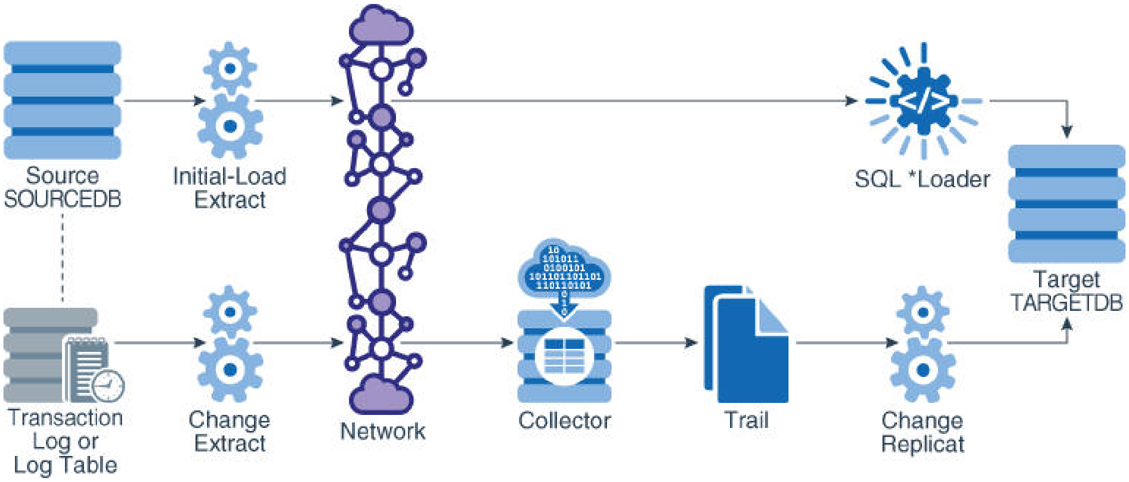
Loading Data with a Direct Bulk Load to SQL*Loader

To use Oracle's SQL*Loader utility to establish the target data, you run an Oracle GoldenGate initial-load Extract to extract the source records and send them directly to an initial-load Replicat *task*. A task is a process that is started dynamically by the Manager process and does not require the use of a Collector process or file. The initial-load Replicat task interfaces with the API of SQL*Loader to load data as a direct-path bulk load. Data mapping and transformation can be done by either the initial-load Extract or initial-load Replicat, or both. During the load, the change-synchronization groups extract and replicate incremental changes, which are then reconciled with the results of the load.

To control which port is used by Replicat, and to speed up the search and bind process, use the `DYNAMICPORTLIST` parameter in the Manager parameter file. Manager passes the list of port numbers that are specified with this parameter to the Replicat task process. Replicat first searches for a port from this list, and only if no ports are available from the list does Replicat begin scanning in ascending order from the default Manager port number until it finds an available port.

This method supports standard character, numeric, and datetime data types, as well as `CLOB`, `NCLOB`, `BLOB`, `LONG`, `XML`, and user-defined datatypes (`UDT`) embedded with the following attributes: `CHAR`, `NCHAR`, `VARCHAR`, `NVARCHAR`, `RAW`, `NUMBER`, `DATE`, `FLOAT`, `TIMESTAMP`, `CLOB`, `BLOB`, `XML`, and `UDT`. `VARRAYS` are not supported. Character sets are converted between source and target where applicable.

This method supports Oracle internal tables, but does not convert between the source and target character sets during the load.



To Load Data With a Direct Bulk Load to SQL*Loader

1. Make certain that you have addressed the requirements in "Prerequisites for Initial Load".
2. Grant `LOCK ANY TABLE` to the Replicat database user on the target Oracle database.
3. On the source and target systems, run GGSCI and start Manager.

   ```
   START MANAGER
   ```

4. On the source system, issue the following command to create the initial-load Extract.

   ```
   ADD EXTRACT initial-load_Extract, SOURCEISTABLE
   ```

   Where:

   - *`initial-load_Extract`* is the name of the initial-load Extract, up to eight characters.
   - `SOURCEISTABLE` designates Extract as an initial-load process that reads complete records directly from the source tables. Do not use any of the other `ADD EXTRACT` service options or datasource arguments.

5. On the source system, issue the following command to create an initial-load Extract parameter file.

   ```
   EDIT PARAMS initial-load_Extract
   ```

6. Enter the parameters listed in Table 7-4 in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Extract parameter file for a direct bulk load to SQL*Loader.

   ```
   EXTRACT initext
   SOURCEDB mydb, USERIDALIAS ogg
   RMTHOSTOPTIONS ny4387, MGRPORT 7888, ENCRYPT AES 192 KEYNAME mykey
   RMTTASK REPLICAT, GROUP initrep
   TABLE hr.*;
   TABLE sales.*;
   ```

Table 7-4 Initial-load Extract Parameters for a Direct Bulk Load to SQL*Loader

Parameter	Description
`EXTRACT` *`initial-load_Extract`*	Specifies the initial-load Extract.

`[, USERIDALIAS` *`alias, options`* `|`
`, USERID` *`user, options`*`]`
	Specifies database connection information.
	`USERID` and `USERIDALIAS` specify database credentials if required.

`RMTHOSTOPTIONS` *`hostname`*`,`
`MGRPORT` *`portnumber`*
`[, ENCRYPT` *`algorithm`* `KEYNAME` *`keyname`*`]`
	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

`RMTTASK replicat,`
`GROUP` *`initial-load_Replicat`*
• *`initial-load_Replicat`* is the name of the initial-load Replicat group.
	Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.

`TABLE [`*`container.`*`]`*`owner.object`*`;`	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container unless `SOURCECATALOG` is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`TABLEEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated `TABLE` statement. See *Parameters and Functions Reference for Oracle GoldenGate* for details.

7. Enter any appropriate optional parameters.
8. Save and close the file.
9. On the target system, issue the following command to create the initial-load Replicat.

   ```
   ADD REPLICAT initial-load_Replicat, SPECIALRUN
   ```

   Where:

   - *`initial-load_Replicat`* is the name of the initial-load Replicat task.

   - `SPECIALRUN` identifies the initial-load Replicat as a one-time task, not a continuous process.

10. On the target system, issue the following command to create an initial-load Replicat parameter file.

    ```
    EDIT PARAMS initial-load_Replicat
    ```

11. Enter the parameters listed in Table 7-5 in the order shown, starting a new line for each parameter statement. The following is a sample initial-load Replicat parameter file for a direct load to SQL*Loader.

    ```
    REPLICAT initrep
    USERIDALIAS ogg
    BULKLOAD
    SOURCEDEFS /ggs/dirdef/source_defs
    MAP hr.*, TARGET hr.*;
    MAP sales.*, TARGET hr.*;
    ```

Table 7-5   Initial-load Replicat Parameters for Direct Load to SQL*Loader

Parameter	Description
`REPLICAT` *`initial-load_Replicat`*	Specifies the initial-load Replicat task to be started by Manager. Use the name that you specified when you created the initial-load Replicat.

`[TARGETDB` *`container`*`]`
`[, USERIDALIAS` *`alias, options`* `|`
`, USERID` *`user, options`*`]`
	Specifies database connection information.
	`TARGETDB` specifies the target Oracle container. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.
	`USERID` and `USERIDALIAS` specify database credentials if required.

`BULKLOAD`	Directs Replicat to interface directly with the Oracle SQL*Loader interface. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

`{SOURCEDEFS` *`full_pathname`*`} |`
`ASSUMETARGETDEFS`
• Use `SOURCEDEFS` if the source and target tables have different definitions. Specify the source-definitions file generated by DEFGEN.
• Use `ASSUMETARGETDEFS` if the source and target tables have the same definitions.
	Specifies how to interpret data definitions.

`SOURCECATALOG`	Specifies a default source Oracle container for subsequent `MAP` statements. Enables the use of two-part names (*`schema.object`*) where three-part names otherwise would be required. You can use multiple instances of this parameter to specify different default containers for different sets of `MAP` parameters.

Table 7-5 (Cont.) Initial-load Replicat Parameters for Direct Load to SQL*Loader

Parameter	Description

`MAP [`*`container.`*`]`*`owner.object`*`,`
`TARGET` *`owner.object`*`[, DEF` *`template`*`]`
`;`
	Specifies a relationship between a source object or objects and a target object or objects. `MAP` specifies the source object, and `TARGET` specifies the target object.
	For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database, the source object name must include the name of the container unless `SOURCECATALOG` is used.
	For the target object, specify only the *`owner.object`* components of the name, regardless of the database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container to which you want to load data.
	See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
	The `DEF` option specifies a definitions template.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`MAPEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated `MAP` statement. See *Parameters and Functions Reference for Oracle GoldenGate* for details.

12. Enter any appropriate optional Replicat parameters listed in *Parameters and Functions Reference for Oracle GoldenGate*.
13. Save and close the parameter file.
14. On the source system, start change extraction.

    ```
    START EXTRACT group
    ```

15. View the Replicat parameter file to make certain that the `HANDLECOLLISIONS` parameter is listed. If not, add the parameter to the file.
16. (Oracle, if replicating sequences) Issue the `DBLOGIN` command as the user who has `EXECUTE` privilege on `update.Sequence`.

    ```
    GGSCI> DBLOGIN USERID DBLOGINuser, PASSWORD password [encryption_options]
    ```

17. (Oracle, if replicating sequences) Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target. You can use an asterisk wildcard for any or all characters in the name of a sequence (but not the owner).

    ```
    FLUSH SEQUENCE owner.sequence
    ```

18. On the source system, start the initial-load Extract.

    ```
    START EXTRACT initial-load_Extract
    ```

    > **Caution:**
    >
    > Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

19. On the target system, issue the following command to determine when the load is finished. Wait until the load is finished before proceeding to the next step.

```
VIEW REPORT initial-load_Extract
```

20. On the target system, start change replication.

    ```
    START REPLICAT group
    ```

21. On the target system, issue the following command to verify the status of change replication.

    ```
    INFO REPLICAT group
    ```

22. Continue to issue the `INFO REPLICAT` command until you have verified that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that point.
23. On the target system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

    ```
    SEND REPLICAT group, NOHANDLECOLLISIONS
    ```

24. On the target system, edit the Replicat parameter file to remove the `HANDLECOLLISIONS` parameter. This prevents `HANDLECOLLISIONS` from being enabled again the next time Replicat starts.

    > **Caution:**
    >
    > Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

25. Save and close the parameter file.

    From this point forward, Oracle GoldenGate continues to synchronize data changes.

Precise Instantiation for MySQL to MySQL Replication Using the Dump Utility

For MySQL, the precise instantiation is only valid from MySQL to MySQL and not on non-MySQL database sources and targets.

The precise instantiation in MySQL can be achieved using the dump utility of MySQL shell, and not using the OGG initial load. MySQL shell provides utilities to dump an instance, schema or tables. See the following link for details about the dump utility:

https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-utilities-dump-instance-schema.html

Use MySQL Shell's instance `dump utility util.dumpInstance()` to dump or export the source MySQL database.

> **Note:**
>
> The `dump utility util.dumpInstance()` is different from `mysqldump`.

The dump utility creates multiple files. The position of the last committed record is found in the `@.json file`. This file contains both `lognumber`/`offset` and the `gtid` set for the last committed transaction.

An example position in the `@.json` file looks similar to the following:

```
"binlogFile": "binlog.000005",
    "binlogPosition": 1289,
    "gtidExecuted": "1174b383-3441-11e8-b90a-
c80aa9429920:1-9,\n1174b383-3441-11e8-b90a-c80aa9429921:1-9"
```

Precise Instantiation for Oracle GoldenGate Extract for MySQL

The following steps describe precise instantiation for Extract in Oracle GoldenGate Classic Architecture:

1. To test the precise instantiation on non-gtid-based capture, make sure either the `gtid_mode` is not enabled in the database server or `_DISABLEGTIDRECOVERY true` is specified in the Extract parameter file.
2. After completing the initial load, use the instance dump.
3. Read the `@.json` file and get the values of `binlogFile` and `binlogPosition` from the `@.json` file. Use these values as `lognum` and `logpos`.
4. Add Extract using `lognum` and `logpos` in Oracle GoldenGate GGSCI, as shown in the following example:

   ```
   ADD EXTRACT extpos tranlog lognum 5 logpos 1289
   ```

5. Perform the DML operations and verify that there are no duplicate or missing transactions on the target side. Check the Extract report file for the initial position using `lognum` and `logpos`.

Backing up the Oracle GoldenGate Environment

After you start Oracle GoldenGate processing, an effective backup routine is critical to preserving the state of processing in the event of a failure. Unless the Oracle GoldenGate working files can be restored, the entire replication environment must be re-instantiated, complete with new initial loads.

As a best practice, include the entire Oracle GoldenGate home installation in your backup routines. There are too many critical sub-directories, as well as files and programs at the root of the directory, to keep track of separately. In any event, the most critical files are those that consume the vast majority of backup space, and therefore it makes sense just to back up the entire installation directory for fast, simple recovery.

Monitoring and Controlling Processing After the Instantiation

After the target is instantiated and replication is in effect, you can control processes and view the overall health of the replication environment.

If you configured Replicat in integrated mode, you can use the `STATS REPLICAT` command to view statistics on the number of transactions that are applied in integrated mode as compared to those that are applied in direct apply mode.

```
STATS REPLICAT group
```

The output of this command shows the number of transactions applied, the number of transactions that were redirected to direct apply, and the direct transaction ratio, among other statistics. The statistics help you determine whether integrated Replicat is performing as intended. If the environment is satisfactory and there is a high ratio of direct apply operations, consider using nonintegrated Replicat. You can configure parallelism with nonintegrated Replicat.

> **Note:**
>
> To ensure realistic statistics, view apply statistics only after you are certain that the Oracle GoldenGate environment is well established, that configuration errors are resolved, and that any anticipated processing errors are being handled properly.

You can also view runtime statistics for integrated Replicat in the `V$`views for each of the inbound server components.

- The reader statistics are recorded in `V$GG_APPLY_READER` and include statistics on number of messages read, memory used, and dependency counts.
- The apply coordinator statistics are recorded in `V$GG_APPLY_COORDINATOR` and record statistics at the transaction level.
- The apply server statistics are recorded in `V$GG_APPLY_SERVER`. This view records information for each of the apply server processes (controlled by `parallelism` and `max_parallelism` parameters) as separate rows. The statistics for each apply server are identified by the `SERVER_ID` column. If a `SERVER_ID` of `0` exists, this represents an aggregate of any apply servers that exited because the workload was reduced.
- Statistics about the number of messages received by the database from Replicat are recorded in the `V$GG_APPLY_RECEIVER` table.

To control processes, see Controlling Oracle GoldenGate Processes in *Administering Oracle GoldenGate*.

To ensure that all processes are running properly and that errors are being handled according to your error handling rules, see Handling Processing Errors in *Administering Oracle GoldenGate*. Oracle GoldenGate provides commands and logs to view process status, lag, warnings, and other information.

To know more about querying the following views, see Oracle Database Reference.

- `V$GOLDENGATE_TABLE_STATS` to see statistics for DML and collisions that occurred for each replicated table that the inbound server processed.
- `V$GOLDENGATE_TRANSACTION` to see information about transactions that are being processed by Oracle GoldenGate inbound servers.

Verifying Synchronization

To verify that the source and target data are synchronized, you can use the Oracle GoldenGate Veridata product or use your own scripts to select and compare source and target data.

8

Administer

Learn about installation prerequisites for Oracle GoldenGate, steps to install Oracle GoldenGate for different databases, post-installation tasks, installing patches, and uninstalling Oracle GoldenGate.

Data Management

Learn about various aspects of data management in Oracle GoldenGate, including DDL and DML replication, requirements and steps for configuring procedural replication, using SQLEXEC, Event Actions, and User Exits.

Creating a Data Definitions File

When replicating data from one table to another, an important consideration is whether the column structures (metadata) of the source and target tables are identical. Oracle GoldenGate looks up metadata for the following purposes:

> **Note:**
>
> This is only required when writing trails for Oracle GoldenGate 11.2 or earlier.

- On the source, to supply complete information about captured operations to the Replicat process.
- On the target, to determine the structures of the target tables, so that the replicated data is correctly mapped and converted (if needed) by Replicat.

When source and target table definitions are dissimilar, Oracle GoldenGate must perform a conversion from one format to the other. To perform conversions, both sets of definitions must be known to Oracle GoldenGate. Oracle GoldenGate can query the local database to get one set of definitions, but it must rely on a *data-definitions file* to get definitions from the remote database. The data-definitions file contains information about the metadata of the data that is being replicated.

To create a definitions file, you configure and run the `DEFGEN` utility and then transfer the definitions file to the target system. This file must be in place on the target system before you start the Oracle GoldenGate processes for the first time.

Using DDL Replication

Learn how to install, use, configure, and remove DDL replication.
Data Definition Language (DDL) statements (operations) are used to define MySQL database structures or schema. You can use these DDL statements for data replication between MySQL source and target databases. MySQL DDL specifics are found in the MySQL documentation at https://dev.mysql.com/doc/.

Oracle GoldenGate 21c for MySQL has introduced a transaction log based replication solution for MySQL 8.0, which has improved performance and usability when compared to the plug-in based DDL replication approach. However, plug-in based DDL replication approach is supported in older releases.

Plug-in Based DDL Configuration Prerequisites and Considerations

This is an older approach to performing DDL Replication. The prerequisites for configuring DDL replication are as follows:

- DDL replication is supported for MySQL 5.7.
- Remote capture for MySQL 5.7 doesn't support DDL replication.
- Oracle GoldenGate DDL replication uses two plug-ins as a shared library, `ddl_rewriter` and `ddl_metadata`, which must be installed on your MySQL server before Oracle GoldenGate replication starts.
- The standalone application, Oracle GoldenGate `metadata_server`, must be running to capture the DDL metadata.
- The `history` table under the new `oggddl` database (`oggddl.history`). This metadata history table is used to store and retrieve the DDL metadata history. The history table records must be ignored from being logged into the binary log so you must specify `binlog-ignore-db=oggddl` in the `my.cnf` file.
- You should not manually drop the `oggddl` database or the `history` table because all DDL statements that run after this event will be lost.
- You should not stop the `metadata_server` during DDL capture as all the DDL statements that run after this event will be lost.
- You should not manually remove the `ddl_rewriter` and the `ddl_metadata` plugins during DDL capture because all DDL statements that run after this event will be lost.
- DDL executed within the stored procedure is *not* supported. For example , a DDL executed as in the following is *not* supported.

```
CREATE PROCEDURE atssrc.generate_data()
BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 800 DO
SET i = i + 1;
IF (i = 100) then
alter table atssrc.`ddl6` add col2 DATE after id;
ELSEIF (i = 200) then
alter table atssrc.`ddl6` add col3 DATETIME after datetime;
ELSEIF (i = 300) then
alter table atssrc.`ddl6` add `col4` timestamp NULL DEFAULT NULL after
channel;
ELSEIF (i = 400) then
alter table atssrc.`ddl6` add col5 YEAR after value;
END IF;
END WHILE;
END$$
DELIMITER ;
call atssrc.generate_data();
```

- By design, the heartbeat table DDLs are ignored by the capture and you should create the heartbeat tables manually at the target.

Installing DDL Replication

To install DDL replication, you run the installation script that is provided with Oracle GoldenGate as the replication user. This user must have `Create`, `Insert`,`Select`, `Delete`, `Drop`, and `Truncate` database privileges. Additionally, this user must have write permission to copy the Oracle GoldenGate plugin in the MySQL plugin directory. For example, the MySQL plugin are typically in `/usr/lib64/mysql/plugin/`.

The installation script options are `install`, `uninstall`, `start`, `stop`, and `restart`.

The command to install DDL replication uses the install option, user id, password, and port number respectively:

```
bash-3.2$ ./ddl_install.sh install-option user-id password port-number
```

For example:

```
bash-3.2$ ./ddl_install.sh install root welcome 3306
```

The DDL replication installation script completes the following tasks:

1. Ensures that you have a supported MySQL server version installed. DDL replication is supported for MySQL 5.7.10 and greater.
2. Locates the MySQL plugin directory.
3. Ensures that the `ddl_rewriter`, `ddl_metadata` plugins and the `metadata_server` files exist. If these files are not found, then an error message appears and the installation exits.
4. Ensures that the plugins are already installed. If installed, the script exits with a message requesting you to uninstall first and then reinstall.
5. Stops the `metadata_server` if it is running.
6. Deletes the `oggddl.history` table if it exists.
7. Starts the `metadata_server` as a daemon process.
8. Installs the `ddl_rewriter` and `ddl_metadata` plugins.

Using the Metadata Server

You can use the following options with the metadata server:

- You must have the Oracle GoldenGate `metadata_server` running to capture the DDL metadata.
- Run the install script with `start` option to start the metadata server.
- Run the install script with `stop` option to stop the metadata server.
- Run the install script with `restart` option to stop the running metadata server and start again.
- Oracle GoldenGate DDL replication uses two plugins as a shared library, `ddl_rewriter` and `ddl_metadata`, both of which must be installed on your MySQL server before Oracle GoldenGate replication starts.

- The `oggddl.history` metadata history table is used to store and retrieve the DDL metadata history.

There is a single history table and metadata server for each MySQL server. If you want to issue and capture DDLs from multiple instances of an Extract process on the same database server at the same time, there is a possibility of conflict between accessing and populating the metadata history table. Oracle recommends that you do not run and capture DDLs using multiple Extract instances on the same MySQL server.

Using DDL Filtering for Replication

The following options are supported for MySQL DDL replication:

Option	Description
`DDL INCLUDE OPTYPE CREATE OBJTYPE TABLE;`	Include create table.
`DDL INCLUDE OBJNAME ggvam.*`	Include tables under the `ggvam`database.
`DDL INCLUDE OBJNAME atssrc.ggvam EXCLUDE OBJNAME atssrc.emp`	Exclude all the tables under the `ggvam` database with the `emp` wildcard. **Note:** The `EXCLUDE` option needs to be added together with the `INCLUDE` option in this parameter.
`DDL INCLUDE INSTR ‘XYZ’`	Include DDL that contains this string.
`DDL EXCLUDE INSTR ‘WHY’`	Excludes DDL that contains this string.
`DDL INCLUDE MAPPED`	MySQL DDL uses this option and should be used as the default for Oracle GoldenGate MySQL DDL replication. `DDL INCLUDE ALL` and `DDL` are not supported.
`DDL EXCLUDE ALL`	Default option.

For a full list of options, see DDL in *Parameters and Functions Reference for Oracle GoldenGate*.

Using DDL Statements and Options

- `INCLUDE` (default) means include all objects that fit the rest of the description. `EXCLUDE` means to omit items that fit the description. Exclude rules take precedence over include rules.
- `OPTYPE` specifies the types of operations to be included or excluded. You can use `CREATE` and `ALTER`. Multiple `OPTYPE` can be specified using parentheses. For example, `optype (create, alter)`. The asterisk (*) wildcard can be specified to indicate all operation types, and this is the default.
- `OBJTYPE` specifies the `TABLE` operations to include or exclude. The wildcard can be specified to indicate all object types, and this is the default.

- `OBJNAME` specifies the actual object names to include or exclude. For example, `eric.*`. Wildcards are specified as in other cases where multiple tables are specified. The default is `*`.
- `String` indicates that the rule is true if any of the strings in `stringspec` are present (or false if `excludestring` is specified and the `stringspec` is present). If multiple `string` entries are made, at least one entry in each `stringspec` must be present to make the rule evaluate true.

  For example:

  ```
  ddlops string (“a”, “b”), string (“c”) evaluates true if string “a” OR “b”
  is present, AND string “c” is present
  ```

- `local` is specified if you want the rule to apply only to the current Extract trail (the Extract trail to which the rule applies must precede this `ddlops` specification).
- The semicolon is required to terminate the parameter entry.

  For example:

  ```
  ddl optype (create, drop), objname (eric.*);
  ddl exclude objname (eric.tab*);
  exttrail a;
  exttrail b;
  ddl optype (create), objname (joe.*), string (“abc”, “xyz”) local;
  ddl optype (alter), objtype (index);
  ```

  In this preceding example, the `exttrail a` gets creates and drops for all objects that belong to `eric`, except for objects that start with `tab`, `exttrail a` also gets all alter index statements, unless the index name begins with `tab` (the rule is global even though it’s included in `exttrail b`). `exttrail b` gets the same objects as `a`, and it also gets all creates for objects that belong to `joe` when the string `abc`or `xyz` is present in the DDL text. The `ddlops.c` module stores all DDL operation parameters and executes related rules.

Additionally, you can use the `DDLOPTIONS` parameter to configure aspects of DDL processing other than filtering and string substitution. You can use multiple `DDLOPTIONS` statements and Oracle recommends using one. If you are using multiple `DDLOPTIONS` statements, then make each of them unique so that one does not override the other. Multiple `DDLOPTIONS` statements are executed in the order listed in the parameter file.

See DDL and DDLOPTIONS.

Troubleshooting Plug-in Based DDL Replication

Plug-in based DDL replication relies on a metadata history table and the metadata plugin and server. To troubleshoot when DDL replication is enabled, the history table contents and the metadata plugin server logs are required.
You can use the `mysqldump` command to generate the history table dump using one of the following examples:

```
mysqldump [options] database [tables]
mysqldump [options] --databases [options] DB1 [DB2 DB3...]
mysqldump [options] --all-databases [options]
```

For example, `bash-3.2$ mysqldump -uroot -pwelcome oggddl history > outfile`

The metadata plugins and server logs are located in the MySQL and Oracle GoldenGate installation directories respectively.

If you find an error in the log files, you need to ensure that the metadata server is running.

Uninstalling Plug-In Based DDL Replication

If you no longer want to capture the DDL events, then you can use the same install script and select the `uninstall` option to disable the DDL setup. Also, any Extract with DDL parameters should be removed or disabled. If you want to capture the DDL again, you can run the install script again. You should take care when multiple instances of the capture process is running on the same instance of your MySQL server. The DDL setup should *not* be disturbed or uninstalled when multiple capture processes are running and when at most one capture is designed to capture the DDL statement.

Use the installation script with the `uninstall` option to uninstall DDL Replication. For example:

```
bash-3.2$ ./ddl_install.sh uninstall root welcome 3306
```

The script performs the following tasks:

1. Uninstalls the `ddl_rewriter` and `ddl_metadata` plugins.
2. Deletes the `oggddl.history` table if exists.
3. Removes the plugins from MySQL plugin directory.
4. Stops the `metadata_server` if it is running.

Oracle: DDL Replication

Learn about DDL replication in Oracle.

Extract supports the DDL capture method for Oracle 11.2.0.4 or later. An Extract can capture DDL operations from a source Oracle database natively through the Oracle logmining server.

Managing the DDL Replication Environment

This chapter contains instructions for making changes to the database environment or the Oracle GoldenGate environment when the Oracle GoldenGate DDL trigger is being used to support DDL replication.
For instructions on configuring Oracle GoldenGate DDL support, see Configuring DDL Support .

> **Note:**
>
> This chapter is only relevant for classic capture mode or integrated capture mode in which trigger-based DDL capture is being used.

Disabling DDL Processing Temporarily

You must disable DDL activities before performing an instantiation or other tasks, if directed.

You can resume DDL processing after the task is finished.

1. Disable user DDL operations on the source database.
2. If there are previous DDL replication processes that are still active, make certain that the last executed DDL operation was applied to the target before stopping those processes, so that the load data is applied to objects that have the correct metadata.
3. Comment out the `DDL` parameter in the Extract and Replicat parameter files that you configured for the new Oracle GoldenGate environment. Comment out any other parameters that support DDL.
4. Disable the Oracle GoldenGate DDL trigger, if one is in use. See Enabling and Disabling the DDL Trigger.

Enabling and Disabling the DDL Trigger

You can enable and disable the trigger that captures DDL operations without making any configuration changes within Oracle GoldenGate.

The following scripts control the DDL trigger.

- `ddl_disable`: Disables the trigger. No further DDL operations are captured or replicated after you disable the trigger.
- `ddl_enable`: Enables the trigger. When you enable the trigger, Oracle GoldenGate starts capturing current DDL changes, but does not capture DDL that was generated while the trigger was disabled.

Before running these scripts, disable all sessions that ever issued DDL, including those of the Oracle GoldenGate processes, SQL*Plus, business applications, and any other software that uses Oracle. Otherwise the database might generate an ORA-04021 error. Do not use these scripts if you intend to maintain consistent DDL on the source and target systems.

Maintaining the DDL Marker Table

You can purge rows from the marker table at any time. It does not keep DDL history.

To purge the marker table, use the Manager parameter `PURGEMARKERHISTORY`. Manager gets the name of the marker table from one of the following:

1. The name given with the `MARKERTABLE` parameter in the `GLOBALS` file, if specified.
2. The default name of `GGS_MARKER`.

`PURGEMARKERHISTORY` provides options to specify maximum and minimum lengths of time to keep a row, based on the last modification date.

Deleting the DDL Marker Table

Do not delete the DDL marker table unless you want to discontinue synchronizing DDL.

The marker table and the DDL trigger are interdependent. An attempt to drop the marker table fails if the DDL trigger is enabled. This is a safety measure to prevent the trigger from becoming invalid and missing DDL operations. If you remove the marker table, the following error is generated:

```
ORA-04098: trigger 'SYS.GGS_DDL_TRIGGER_BEFORE' is invalid and failed re-validation
```

The proper way to remove an Oracle GoldenGate DDL object depends on your plans for the rest of the DDL environment. To choose the correct procedure, see one of the following:

- Restoring an Existing DDL Environment to a Clean State

- Removing the DDL Objects from the System

Maintaining the DDL History Table

You can purge the DDL history table to control its size, but do so carefully.

The DDL history table maintains the integrity of the DDL synchronization environment. Purges to this table cannot be recovered through the Oracle GoldenGate interface.

1. To prevent any possibility of DDL history loss, make regular full backups of the history table.
2. To ensure that purged DDL can be recovered, enable Oracle Flashback for the history table. Set the flashback retention time well past the point where it could be needed. For example, if your full backups are at most one week old, retain two weeks of flashback. Oracle GoldenGate can be positioned backward into the flashback for reprocessing.
3. If possible, purge the DDL history table manually to ensure that essential rows are not purged accidentally. If you require an automated purging mechanism, use the `PURGEDDLHISTORY` parameter in the Manager parameter file. You can specify maximum and minimum lengths of time to keep a row.

> **Note:**
>
> Temporary tables created by Oracle GoldenGate to increase performance might be purged at the same time as the DDL history table, according to the same rules. The names of these tables are derived from the name of the history table, and their purging is reported in the Manager report file. This is normal behavior.

Deleting the DDL History Table

The history table and the DDL trigger are interdependent. An attempt to drop the history table fails if the DDL trigger is enabled. This is a safety measure to prevent the trigger from becoming invalid and missing DDL operations.

Do not delete the DDL history table unless you want to discontinue synchronizing DDL. The history table contains a record of DDL operations that were issued. Once an Extract switches from using the DDL trigger to not using the trigger, as when source database redo compatibility is advanced to 11.2.0.4 or greater, these objects can be deleted though *not immediately*. It is imperative that all mining of the redo generated before the compatibility change be complete and that this redo not need to be mined again.

If you remove the history table, the following error is generated:

```
ORA-04098: trigger 'SYS.GGS_DDL_TRIGGER_BEFORE' is invalid and failed re-validation
```

The proper way to remove an Oracle GoldenGate DDL object depends on your plans for the rest of the DDL environment. To choose the correct procedure, see one of the following:

- Restoring an Existing DDL Environment to a Clean State
- Removing the DDL Objects from the System

Purging the DDL Trace File

To prevent the DDL trace file from consuming excessive disk space, run the `ddl_cleartrace` script on a regular basis.

This script deletes the trace file, but Oracle GoldenGate will create it again.

The default name of the DDL trace file is `ggs_ddl_trace.log`. It is in the `USER_DUMP_DEST` directory of Oracle. The `ddl_cleartrace` script is in the Oracle GoldenGate directory.

Applying Database Patches and Upgrades when DDL Support is Enabled

Database patches and upgrades usually invalidate the Oracle GoldenGate DDL trigger and other Oracle GoldenGate DDL objects.

Before applying a database patch, do the following.

1. Log in to SQL*Plus as a user that has `SYSDBA` privileges.
2. Disable the Oracle GoldenGate DDL trigger by running the `ddl_disable` script in SQL*Plus.
3. Apply the patch.
4. Enable the DDL trigger by running the `ddl_enable` script in SQL*Plus.

> **Note:**
>
> Database upgrades and patches generally operate on Oracle objects. Because Oracle GoldenGate filters out those objects automatically, DDL from those procedures is not replicated when replication starts again.

To avoid recompile errors after the patch or upgrade, which are caused if the trigger is not disabled before the procedure, consider adding calls to `@ddl_disable` and `@ddl_enable` at the appropriate locations within your scripts.

Apply Oracle GoldenGate Patches and Upgrades when DDL support is Enabled

Use the following steps to apply a patch or upgrade to the DDL objects. This section explains how to apply Oracle GoldenGate patches and upgrades when DDL support is enabled.

> **Note:**
>
> If the release notes or upgrade documentation for your Oracle GoldenGate release contain instructions similar to those provided in this section, follow those instructions instead the ones in this section. Do not use this procedure for an upgrade from an Oracle GoldenGate version that does not support DDL statements that are larger than 30K (pre-version 10.4). To upgrade in that case, follow the instructions in Restoring an Existing DDL Environment to a Clean State.

This procedure may or may not preserve the current DDL synchronization configuration, depending on whether the new build requires a clean installation.

1. Run GGSCI. Keep the session open for the duration of this procedure.
2. Stop Extract to stop DDL capture.

   ```
   STOP EXTRACT group
   ```

3. Stop Replicat to stop DDL replication.

   ```
   STOP REPLICAT group
   ```

4. Download or extract the patch or upgrade files according to the instructions provided by Oracle GoldenGate.
5. Change directories to the Oracle GoldenGate installation directory.
6. Log in to SQL*Plus as a user that has `SYSDBA` privileges.
7. Disconnect all sessions that ever issued DDL, including those of Oracle GoldenGate processes, SQL*Plus, business applications, and any other software that uses Oracle. Otherwise the database might generate an ORA-04021 error.
8. Run the `ddl_disable` script to disable the DDL trigger.
9. Run the `ddl_setup` script. You are prompted for the name of the Oracle GoldenGate DDL schema. If you changed the schema name, use the new one.
10. Run the `ddl_enable.sql` script to enable the DDL trigger.
11. In GGSCI, start Extract to resume DDL capture.

    ```
    START EXTRACT group
    ```

12. Start Replicat to start DDL replication.

    ```
    START REPLICAT group
    ```

Restoring an Existing DDL Environment to a Clean State

Follow these steps to completely remove, and then reinstall, the Oracle GoldenGate DDL objects.

This procedure creates a new DDL environment and removes any current DDL history.

> **Note:**
>
> Due to object interdependencies, all objects must be removed and reinstalled in this procedure.

1. If you are performing this procedure in conjunction with the installation of a new Oracle GoldenGate version, download and install the Oracle GoldenGate files, and create or update process groups and parameter files as necessary.
2. (Optional) To preserve the continuity of source and target structures, stop DDL activities and then make certain that Replicat finished processing all of the DDL and DML data in the trail. To determine when Replicat is finished, issue the following command until you see a message that there is no more data to process.

   ```
   INFO REPLICAT group
   ```

> **Note:**
>
> Instead of using `INFO REPLICAT`, you can use the `EVENTACTIONS` option of `TABLE` and `MAP` to stop the Extract and Replicat processes after the DDL and DML has been processed.

3. Run GGSCI.
4. Stop Extract to stop DDL capture.

   ```
   STOP EXTRACT group
   ```

5. Stop Replicat to stop DDL replication.

   ```
   STOP REPLICAT group
   ```

6. Change directories to the Oracle GoldenGate installation directory.
7. Log in to SQL*Plus as a user that has `SYSDBA` privileges.
8. Disconnect all sessions that ever issued DDL, including those of Oracle GoldenGate processes, SQL*Plus, business applications, and any other software that uses Oracle. Otherwise the database might generate an ORA-04021 error.
9. Run the `ddl_disable` script to disable the DDL trigger.
10. Run the `ddl_remove` script to remove the Oracle GoldenGate DDL trigger, the DDL history and marker tables, and other associated objects. This script produces a `ddl_remove_spool.txt` file that logs the script output and a `ddl_remove_set.txt` file that logs environment settings in case they are needed for debugging.
11. Run the `marker_remove` script to remove the Oracle GoldenGate marker support system. This script produces a `marker_remove_spool.txt` file that logs the script output and a `marker_remove_set.txt` file that logs environment settings in case they are needed for debugging.
12. If you are changing the DDL schema for this installation, grant the following permission to the Oracle GoldenGate schema.

    ```
    GRANT EXECUTE ON utl_file TO schema;
    ```

13. If you are changing the DDL schema for this installation, the schema's default tablespace must be dedicated to that schema; do not allow any other schema to share it. `AUTOEXTEND` must be set to `ON` for this tablespace, and the tablespace must be sized to accommodate the growth of the `GGS_DDL_HIST` and `GGS_MARKER` tables. The `GGS_DDL_HIST` table, in particular, will grow in proportion to overall DDL activity.

> **Note:**
>
> If the DDL tablespace fills up, Extract stops capturing DDL. To cause user DDL activity to fail when that happens, edit the `params.sql` script and set the `ddl_fire_error_in_trigger` parameter to `TRUE`. Stopping user DDL gives you time to extend the tablespace size and prevent the loss of DDL capture. Managing tablespace sizing this way, however, requires frequent monitoring of the business applications and Extract to avoid business disruptions. Instead, Oracle recommends that you size the tablespace appropriately and set `AUTOEXTEND` to `ON` so that the tablespace does not fill up.

> **WARNING:**
>
> Do not edit any other parameters in `params.sql` except if you need to follow documented instructions to change certain object names.

14. If you are changing the DDL schema for this installation, edit the `GLOBALS` file and specify the new schema name with the following parameter.

    ```
    GGSCHEMA schema_name
    ```

15. Run the `marker_setup` script to reinstall the Oracle GoldenGate marker support system. You are prompted for the name of the Oracle GoldenGate schema.
16. Run the `ddl_setup` script. You are prompted for the name of the Oracle GoldenGate DDL schema.
17. Run the `role_setup` script to recreate the Oracle GoldenGate DDL role.
18. Grant the role to all Oracle GoldenGate users under which the following Oracle GoldenGate processes run: Extract, Replicat, GGSCI, and Manager. You might need to make multiple grants if the processes have different user names.
19. Run the `ddl_enable.sql` script to enable the DDL trigger.

Removing the DDL Objects from the System

This procedure removes the DDL environment and removes the history that maintains continuity between source and target DDL operations.

> **Note:**
>
> Due to object interdependencies, all objects must be removed.

1. Run GGSCI.
2. Stop Extract to stop DDL capture.

   ```
   STOP EXTRACT group
   ```

3. Stop Replicat to stop DDL replication.

   ```
   STOP REPLICAT group
   ```

4. Change directories to the Oracle GoldenGate installation directory.
5. Run SQL*Plus and log in as a user that has `SYSDBA` privileges.
6. Disconnect all sessions that ever issued DDL, including those of Oracle GoldenGate processes, SQL*Plus, business applications, and any other software that uses Oracle. Otherwise the database might generate an ORA-04021 error.
7. Run the `ddl_disable` script to disable the DDL trigger.
8. Run the `ddl_remove` script to remove the Oracle GoldenGate DDL trigger, the DDL history and marker tables, and the associated objects. This script produces a `ddl_remove_spool.txt` file that logs the script output and a `ddl_remove_set.txt` file that logs current user environment settings in case they are needed for debugging.

9. Run the `marker_remove` script to remove the Oracle GoldenGate marker support system. This script produces a `marker_remove_spool.txt` file that logs the script output and a `marker_remove_set.txt` file that logs environment settings in case they are needed for debugging.

Configuring DDL Support

This chapter contains information to help you understand and configure DDL support in Oracle GoldenGate.

Prerequisites for Configuring DDL

Extract can capture DDL operations from a source Oracle database natively through the Oracle logmining server.

Support for DDL Capture with Extract

Extract supports the DDL capture method for **Oracle 11.2.0.4 or later**.

Oracle databases that have the `COMPATIBLE` parameter set to 11.2.0.4 or higher support DDL capture through the database logmining server. This method is known as *native DDL capture*. Native DDL capture is the only supported method for capturing DDL from a multitenant container database.

For downstream mining, the source database must also have database `COMPATIBLE` set to 11.2.0.4 or higher to support DDL capture through the database logmining server.

Configuration Guidelines for DDL Support

The following are guidelines to take into account when configuring Oracle GoldenGate processes to support DDL replication.

Database Privileges

For database privileges that are required for Oracle GoldenGate to support DDL capture and replication, see Establishing Oracle GoldenGate Credentials .

Parallel Processing

If using parallel Extract and/or Replicat processes, keep related DDL and DML together in the same process stream to ensure data integrity. Configure the processes so that:

- all DDL and DML for any given object are processed by the same Extract group and by the same Replicat group.
- all objects that are relational to one another are processed by the same process group.

For example, if `ReplicatA` processes DML for `Table1`, then it should also process the DDL for `Table1`. If `Table2` has a foreign key to `Table1`, then its DML and DDL operations also should be processed by `ReplicatA`.

If an Extract group writes to multiple trails that are read by different Replicat groups, Extract sends all of the DDL to all of the trails. Use each Replicat group to filter the DDL by using the filter options of the `DDL` parameter in the Replicat parameter file.

Object Names

Oracle GoldenGate preserves the database-defined object name, case, and character set. This support preserves single-byte and multibyte names, symbols, and accent characters at all levels of the database hierarchy.

Object names must be fully qualified with their two-part or three-part names when supplied as input to any parameters that support DDL synchronization. You can use the question mark (**?**) and asterisk (*****) wildcards to specify object names in configuration parameters that support DDL synchronization, but the wildcard specification also must be fully qualified as a two-part or three-part name. To process wildcards correctly, the `WILDCARDRESOLVE` parameter is set to `DYNAMIC` by default. If `WILDCARDRESOLVE` is set to anything else, the Oracle GoldenGate process that is processing DDL operations will abend and write the error to the process report.

Data Definitions

Because DDL support requires a like-to-like configuration, the `ASSUMETARGETDEFS` parameter must be used in the Replicat parameter file. Replicat will abend if objects are configured for DDL support and the `SOURCEDEFS` parameter is being used. For more information about `ASSUMETARGETDEFS`, see *Parameters and Functions Reference for Oracle GoldenGate*.

For more information about using a definitions file, see *Reference for Oracle GoldenGate*.

Truncates

`TRUNCATE` statements can be supported as follows:

- As part of the Oracle GoldenGate full DDL support, which supports `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL. This is controlled by the DDL parameter (see Enabling DDL Support.)
- As standalone `TRUNCATE` support. This support enables you to replicate `TRUNCATE TABLE`, but no other DDL. The `GETTRUNCATES` parameter controls the standalone `TRUNCATE` feature. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

To avoid errors from duplicate operations, only one of these features can be active at the same time.

Initial Synchronization

To configure DDL replication, start with a target database that is synchronized with the source database. DDL support is compatible with the Replicat initial load method.

Before executing an initial load, disable DDL extraction and replication. DDL processing is controlled by the DDL parameter in the Extract and Replicat parameter files.

After initial synchronization of the source and target data, use all of the source sequence values at least once with `NEXTVAL` before you run the source applications. You can use a script that selects `NEXTVAL` from every sequence in the system. This must be done while Extract is running.

Data Continuity After CREATE or RENAME

To replicate DML operations on new Oracle tables resulting from a `CREATE` or `RENAME` operation, the names of the new tables must be specified in `TABLE` and `MAP` statements in the parameter files. You can use wildcards to make certain that they are included.

To create a new user with `CREATE USER` and then move new or renamed tables into that schema, the new user name must be specified in `TABLE` and `MAP` statements. To create a new user `fin2` and move new or renamed tables into that schema, the parameter statements could look as follows, depending on whether you want the `fin2` objects mapped to the same, or different, schema on the target:

Extract:

```
TABLE fin2.*;
```

Replicat:

```
MAP fin2.*, TARGET different_schema.*;
```

Overview of DDL Synchronization

Oracle GoldenGate supports the synchronization of DDL operations from one database to another.

DDL synchronization can be active when:

- business applications are actively accessing and updating the source and target objects.
- Oracle GoldenGate transactional data synchronization is active.

The components that support the replication of DDL and the replication of transactional data changes (DML) are independent of each other. Therefore, you can synchronize:

- just DDL changes
- just DML changes
- both DDL and DML

For a list of supported objects and operations for DDL support for Oracle, see Supported Objects and Operations in Oracle DDL.

Limitations of Oracle GoldenGate DDL Support

This topic contains some limitations of the DDL feature.

For any additional limitations that were found after this documentation was published, see the *Release Notes for Oracle GoldenGate*.

DDL Statement Length

Oracle GoldenGate measures the length of a DDL statement in bytes, not in characters. The supported length is approximately 4 MB, allowing for some internal overhead that can vary in size depending on the name of the affected object and its DDL type, among other characteristics. If the DDL is longer than the supported size, Extract will issue a warning and ignore the DDL operation.

Supported Topologies

Oracle GoldenGate supports DDL synchronization only in a like-to-like configuration. The source and target object definitions must be identical.

DDL replication is only supported for Oracle to Oracle replication. It is not supported between different databases, like Oracle to Teradata, or SQL Server to Oracle.

Oracle GoldenGatedoes not support DDL on a standby database.

Oracle GoldenGate supports DDL replication in all supported unidirectional configurations, and in bidirectional configurations between two, and only two, systems. For special considerations in an Oracle active-active configuration, see Propagating DDL in Active-Active (Bidirectional) Configurations.

Filtering, Mapping, and Transformation

DDL operations cannot be transformed by any Oracle GoldenGate process. However, source DDL can be mapped and filtered to a different target object by a primary Extract or a Replicat process. Mapping or filtering of DDL by a data-pump Extract is not permitted, and the DDL is passed as it was received from the primary Extract.

For example, `ALTER TABLE TableA` is processed by a data pump as `ALTER TABLE TableA`. It cannot be mapped by that process as `ALTER TABLE TableB`, regardless of any `TABLE` statements that specify otherwise.

Renames

`RENAME` operations on tables are converted to the equivalent `ALTER TABLE RENAME` so that a schema name can be included in the target DDL statement. For example `RENAME tab1 TO tab2` could be changed to `ALTER TABLE schema.tab1 RENAME TO schema.tab2`. The conversion is reported in the Replicat process report file.

Interactions Between Fetches from a Table and DDL

Oracle GoldenGate supports some data types by identifying the modified row from the redo stream and then querying the underlying table to fetch the changed columns. For instance, partial updates on LOBs are supported by identifying the modified row and the LOB column from the redo log, and then querying for the LOB column value for the row from the base table. A similar technique is employed to support UDT.

> **Note:**
>
> Extract only requires fetch for UDT when *not* using native object support.

Such fetch-based support is implemented by issuing a flashback query to the database based on the SCN (System Change Number) at which the transaction committed. The flashback query feature has certain limitations. Certain DDL operations act as barriers such that flashback queries to get data prior to these DDLs do not succeed. Examples of such DDL are `ALTER TABLE MODIFY COLUMN` and `ALTER TABLE DROP COLUMN`.

Thus, in cases where there is Extract capture lag, an intervening DDL may cause fetch requests for data prior to the DDL to fail. In such cases, Extract falls back and fetches the current snapshot of the data for the modified column. There are several limitations to this approach: First, the DDL could have modified the column that Extract needs to fetch (for example, suppose the intervening DDL added a new attribute to the UDT that is being captured). Second, the DDL could have modified one of the columns that Extract uses as a logical row identifier. Third, the table could have been renamed before Extract had a chance to fetch the data.

To prevent fetch-related inconsistencies such as these, take the following precautions while modifying columns.

1. Pause all DML to the table.

2. Wait for Extract to finish capturing all remaining redo, and wait for Replicat to finish processing the captured data from trail. To determine whether Replicat is finished, issue the following command in GGSCI until you see a message that there is no more data to process.

   ```
   INFO REPLICAT group
   ```

3. Execute the DDL on the source.
4. Resume source DML operations.

Comments in SQL

If a source DDL statement contains a comment in the middle of an object name, that comment will appear at the end of the object name in the target DDL statement. For example:

Source:

```
CREATE TABLE hr./*comment*/emp ...
```

Target:

```
CREATE TABLE hr.emp /*comment*/ ...
```

This does not affect the integrity of DDL synchronization. Comments in any other area of a DDL statement remain in place when replicated.

Compilation Errors

If a `CREATE` operation on a trigger, procedure, function, or package results in compilation errors, Oracle GoldenGate executes the DDL operation on the target anyway. Technically, the DDL operations themselves completed successfully and should be propagated to allow dependencies to be executed on the target, for example in recursive procedures.

Interval Partitioning

DDL replication is unaffected by interval partitioning, because the DDL is implicit. However, this is system generated name so Replicat cannot convert this to the target.I believe this is expected behavior. You must drop the partition on the source. For example:

```
alter table t2 drop partition for (20);
```

DML or DDL Performed Inside a DDL Trigger

DML or DDL operations performed from within a DDL trigger are not captured.

LogMiner Data Dictionary Maintenance

Oracle recommends that you gather dictionary statistics *after* the Extract is registered (logminer session) and the logminer dictionary is loaded, or after any significant DDL activity on the database.

Understanding DDL Scopes

Database objects are classified into scopes. A scope is a category that defines how DDL operations on an object are handled by Oracle GoldenGate.

The scopes are:

- `MAPPED`

- `UNMAPPED`
- `OTHER`

The use of scopes enables granular control over the filtering of DDL operations, string substitutions, and error handling.

Mapped Scope

Objects that are specified in `TABLE` and `MAP` statements are of `MAPPED` scope. Extraction and replication instructions in those statements apply to both data (DML) and DDL on the specified objects, unless override rules are applied.

For objects in `TABLE` and `MAP` statements, the DDL operations listed in the following table are supported.

Operations	On any of these Objects[1]

`CREATE`
`ALTER`
`DROP`
`RENAME`
`COMMENT ON`[2]
	`TABLE`[3]
	`INDEX`
	`TRIGGER`
	`SEQUENCE`
	`MATERIALIZED VIEW`
	`VIEW`
	`FUNCTION`
	`PACKAGE`
	`PROCEDURE`
	`SYNONYM`
	`PUBLIC SYNONYM`[4]

`GRANT`
`REVOKE`
	`TABLE`
	`SEQUENCE`
	`MATERIALIZED VIEW`

`ANALYZE`	`TABLE`
	`INDEX`
	`CLUSTER`

[1] `TABLE` and `MAP` do not support some special characters that could be used in an object name affected by these operations. Objects with non-supported special characters are supported by the scopes of `UNMAPPED` and `OTHER`.

[2] Applies to `COMMENT ON TABLE`, `COMMENT ON COLUMN`

[3] Includes `AS SELECT`

[4] Table name must be qualified with schema name.

For Extract, `MAPPED` scope marks an object for DDL capture according to the instructions in the `TABLE` statement. For Replicat, `MAPPED` scope marks DDL for replication and maps it to the object specified by the schema and name in the `TARGET` clause of the `MAP` statement. To perform this mapping, Replicat issues `ALTER SESSION` to set the schema of the Replicat session to the schema that is specified in the `TARGET` clause. If the DDL contains unqualified objects, the schema that is assigned on the target depends on circumstances described in Understanding DDL Scopes.

Assume the following `TABLE` and `MAP` statements:

Extract (source)

```
TABLE fin.expen;
TABLE hr.tab*;
```

Replicat (target)

```
MAP fin.expen, TARGET fin2.expen2;
MAP hr.tab*, TARGET hrBackup.bak_*;
```

Also assume a source DDL statement of:

```
ALTER TABLE fin.expen ADD notes varchar2(100);
```

In this example, because the source table `fin.expen` is in a `MAP` statement with a `TARGET` clause that maps to a different schema and table name, the target DDL statement becomes:

```
ALTER TABLE fin2.expen2 ADD notes varchar2(100);
```

Likewise, the following source and target DDL statements are possible for the second set of `TABLE` and `MAP` statements in the example:

Source:

```
CREATE TABLE hr.tabPayables ... ;
```

Target:

```
CREATE TABLE hrBackup.bak_tabPayables ...;
```

When objects are of `MAPPED` scope, you can omit their names from the DDL configuration parameters, unless you want to refine their DDL support further. If you ever need to change the object names in `TABLE` and `MAP` statements, the changes will apply automatically to the DDL on those objects.

If you include an object in a `TABLE` statement, but not in a `MAP` statement, the DDL for that object is `MAPPED` in scope on the source but `UNMAPPED` in scope on the target.

Unmapped Scope

If a DDL operation is supported for use in a `TABLE` or `MAP` statement, but its base object name is not included in one of those parameters, it is of `UNMAPPED` scope.

An object name can be of `UNMAPPED` scope on the source (not in an Extract `TABLE` statement), but of `MAPPED` scope on the target (in a Replicat `MAP` statement), or the other way around. When Oracle DDL is of `UNMAPPED` scope in the Replicat configuration, Replicat will by default do the following:

1. Set the current schema of the Replicat session to the schema of the source DDL object.
2. Execute the DDL as that schema.
3. Restore Replicat as the current schema of the Replicat session.

Other Scope

DDL operations that cannot be mapped are of `OTHER` scope. When DDL is of `OTHER` scope in the Replicat configuration, it is applied to the target with the same schema and object name as in the source DDL.

An example of `OTHER` scope is a DDL operation that makes a system-specific reference, such as DDL that operates on data file names.

Some other examples of `OTHER` scope:

```
CREATE USER joe IDENTIFIED by joe;
CREATE ROLE ggs_gguser_role IDENTIFIED GLOBALLY;
ALTER TABLESPACE gg_user TABLESPACE GROUP gg_grp_user;
```

Correctly Identifying Unqualified Object Names in DDL

Extract captures the current schema (also called session schema) that is in effect when a DDL operation is executed. The current container is also captured if the source is a multitenant container database.

The container and schema are used to resolve unqualified object names in the DDL.

Consider the following example:

```
CONNECT SCOTT/TIGER
CREATE TABLE TAB1 (X NUMBER);
CREATE TABLE SRC1.TAB2(X NUMBER) AS SELECT * FROM TAB1;
```

In both of those DDL statements, the unqualified table `TAB1` is resolved as `SCOTT.TAB1` based on the current schema `SCOTT` that is in effect during the DDL execution.

There is another way of setting the current schema, which is to set the `current_schema` for the session, as in the following example:

```
CONNECT SCOTT/TIGER
ALTER SESSION SET CURRENT_SCHEMA=SRC;
CREATE TABLE TAB1 (X NUMBER);
CREATE TABLE SRC1.TAB2(X NUMBER) AS SELECT * FROM TAB1;
```

In both of those DDL statements, the unqualified table `TAB1` is resolved as `SRC.TAB1` based on the current schema `SRC` that is in effect during the DDL execution.

In both classic and integrated capture modes, Extract captures the current schema that is in effect during DDL execution, and it resolves the unqualified object names (if any) by using the current schema. As a result, `MAP` statements specified for Replicat work correctly for DDL with unqualified object names.

You can also map a source session schema to a different target session schema, if that is required for the DDL to succeed on the target. This mapping is global and overrides any other mappings that involve the same schema names. To map session schemas, use the `DDLOPTIONS` parameter with the `MAPSESSIONSCHEMA` option.

If the default or mapped session schema mapping fails, you can handle the error with the following `DDLERROR` parameter statement, where error 1435 means that the schema does not exist.

```
DDLERROR 1435 IGNORE INCLUDE OPTYPE ALTER OBJTYPE SESSION
```

Enabling DDL Support

Data Definition Language (DDL) is useful in dynamic environments which change constantly.

By default, the status of DDL replication support is as follows:

- On the source, Oracle GoldenGate DDL support is disabled by default. You must configure Extract to capture DDL by using the `DDL` parameter.
- On the target, DDL support is enabled by default, to maintain the integrity of transactional data that is replicated. By default, Replicat will process all DDL operations that the trail contains. If needed, you can use the `DDL` parameter to configure Replicat to ignore or filter DDL operations.

Filtering DDL Replication

By default, all DDL is passed to Extract.

You can use the filtering with DDL parameter method to filter DDL operations so that specific (or all) DDL is applied to the target database according to your requirements. Valid for native DDL capture. This is the preferred method of filtering and is performed within Oracle GoldenGate, and both Extract and Replicat can execute filter criteria. Extract can perform filtering, or it can send the entire DDL to a trail, and then Replicat can perform the filtering. Alternatively, you can filter in a combination of different locations. The `DDL` parameter gives you control over where the filtering is performed, and it also offers more filtering options, including the ability to filter collectively based on the DDL scope (for example, include all `MAPPED` scope).

> **Note:**
>
> If a DDL operation fails in the middle of a `TRANSACTION`, it forces a commit, which means that the transaction spanning the DDL is split into two. The first half is committed and the second half can be restarted. If a recovery occurs, the second half of the transaction cannot be filtered since the information contained in the header of the transaction is no longer there.

Filtering with PL/SQL Code

This method is only valid for trigger-based capture.

You can write PL/SQL code to pass information about the DDL to a function that computes whether or not the DDL is passed to Extract. By sending fewer DDL operations to Extract, you can improve capture performance.

1. Copy the `ddl_filter.sql` file that is in the Oracle GoldenGate installation directory to a test machine where you can test the code that you will be writing.
2. Open the file for editing. It contains a PL/SQL function named `filterDDL`, which you can modify to specify `if/then` filter criteria. The information that is passed to this function includes:
   - `ora_owner`: the schema of the DDL object
   - `ora_name`: the defined name of the object
   - `ora_objtype`: the type of object, such as `TABLE` or `INDEX`
   - `ora_optype`: the operation type, such as `CREATE` or `ALTER`
   - `ora_login_user`: The user that executed the DDL
   - `retVal`: can be either `INCLUDE` to include the DDL, or `EXCLUDE` to exclude the DDL from Extract processing.

   In the location after the `'compute retVal here'` comment, write filter code for each type of DDL that you want to be filtered. The following is an example:

   ```
   if ora_owner='SYS' then
   retVal:='EXCLUDE';
   end if;
   if ora_objtype='USER' and ora_optype ='DROP' then
   retVal:='EXCLUDE';
   end if;
   ```

```
if ora_owner='JOE' and ora_name like 'TEMP%' then
retVal:='EXCLUDE';
end if;
```

In this example, the following DDL is excluded from being processed by the DDL trigger:

- DDL for objects owned by `SYS`
- any `DROP USER`
- any DDL on `JOE.TEMP%`

3. (Optional) To trace the filtering, you can add the following syntax to each `if/then` statement in the PL/SQL:

```
if ora_owner='JOE' and ora_name like 'TEMP%' then
retVal:='EXCLUDE';
if "&gg_user" .DDLReplication.trace_level >= 1 then
"&gg_user" .trace_put_line ('DDLFILTER', 'excluded JOE.TEMP%');
end if;
```

Where:

- `&gg_user` is the schema of the Oracle GoldenGate DDL support objects.
- `.DDLReplication.trace_level` is the level of DDL tracing. To use trigger tracing, the `TRACE` or `TRACE2` parameter must be used with the `DDL` or `DDLONLY` option in the Extract parameter file. The `.DDLReplication.trace_level` parameter must be set to `>=1`.
- `trace_put_line` is a user-defined text string that Extract writes to the trace file that represents the type of DDL that was filtered.

4. Save the code.
5. Stop DDL activity on the test system.
6. In SQL*Plus, compile the `ddl_filter.sql` file as follows, where `schema_name` is the schema where the Oracle GoldenGate DDL objects are installed.

```
@ddl_filter schema_name
```

7. Test in the test environment to make certain that the filtering works. It is important to perform this testing, because any errors in the code could cause source and target DDL to become out of synchronization.
8. After a successful test, copy the file to the Oracle GoldenGate installation directory on the source production system.
9. Stop DDL activity on the source system.
10. Compile the `ddl_filter.sql` file as you did before.

```
@ddl_filter schema_name
```

11. Resume DDL activity on the source system.

Filtering With Built-in Filter Rules

This method is only valid for trigger-based capture.

You can add inclusion and exclusion rules to control the DDL operations that are sent to Extract by the DDL trigger. By storing rules and sending fewer DDL operations to Extract, you can improve capture performance.

1. Use the `DDLAUX.addRule()` function to define your rules according to the following instructions. This function is installed in the Oracle GoldenGate DDL schema after the DDL objects are installed with the `ddl_setup.sql` script.
2. To activate the rules, execute the function in SQL*Plus or enter a collection of rules in a SQL file and execute that file in SQL*Plus.

DDLAUX.addRule() Function Definition

```
FUNCTION addRule( obj_name IN VARCHAR2 DEFAULT NULL,
base_obj_name IN VARCHAR2 DEFAULT NULL,
owner_name IN VARCHAR2 DEFAULT NULL,
base_owner_name IN VARCHAR2 DEFAULT NULL,
base_obj_property IN NUMBER DEFAULT NULL,
obj_type IN NUMBER DEFAULT NULL,
command IN VARCHAR2 DEFAULT NULL,
inclusion IN boolean DEFAULT NULL ,
sno IN NUMBER DEFAULT NULL)
RETURN NUMBER;
```

Parameters for DDLAUX.addRule()

The information passed to this function are the following parameters, which correlate to the attributes of an object. All parameters are optional, and more than one parameter can be specified.

- `sno`: Specifies a serial number that identifies the rule. The order of evaluation of rules is from the lowest serial number to the highest serial number, until a match is found. The `sno` can be used to place inclusion rules ahead of an exclusion rule, so as to make an exception to the exclusion rule. Because this is a function and not a procedure, it returns the serial number of the rule, which should be used for the drop rule specified with `DDLAUX.dropRule()`. The serial number is generated automatically unless you specify one with this statement at the beginning of your code: `DECLARE sno NUMBER; BEGIN sno :=`

  For example:

  ```
  DECLARE
    sno NUMBER;
  BEGIN
    sno := tkggadmin..DDLAUX.ADDRULE(obj_name => 'GGS%' ,
                                     obj_type => TYPE_TABLE);
  END;
  /
  ```

- `obj_name`: Specifies the object name. If the name is case-sensitive, enclose it within double quotes.
- `owner_name`: Specifies the name of the object schema
- `base_obj_name`: Specifies the base object name of the DDL object (such as the base table if the object is an index). If the name is case-sensitive, enclose it within double quotes.
- `base_owner_name`: Specifies the base object schema name.
- `base_obj_property`: Specifies the base object property.
- `obj_type`: Specifies the object type.
- `command`: Specifies the command.
- `inclusion = TRUE`: Indicates that the specified objects are to be captured by the DDL trigger. If this parameter is not specified, the rule becomes an exclusion rule, and the specified objects are not captured. You can specify both an exclusion rule and an inclusion rule. If a DDL does not match any of the rules, it is included (passed to Extract) by default.

Calling `DDLAUX.addRule()` without any parameters generates an *empty rule* that excludes all DDL on all the objects.

Valid DDL Components for DDLAUX.addRule()

The following are the defined DDL object types, base object properties, and DDL commands that can be specified in the function code.

Valid object types are:

```
TYPE_INDEX
TYPE_TABLE
TYPE_VIEW
TYPE_SYNONYM
TYPE_SEQUENCE
TYPE_PROCEDURE
TYPE_FUNCTION
TYPE_PACKAGE
TYPE_TRIGGER
```

Valid base object properties are:

```
TB_IOT
TB_CLUSTER
TB_NESTED
TB_TEMP
TB_EXTERNAL
```

Valid commands are:

```
CMD_CREATE
CMD_DROP
CMD_TRUNCATE
CMD_ALTER
```

Examples of Rule-based Trigger Filtering

The following example excludes all temporary tables, except tables with names that start with `IMPTEMP`.

```
1. DDLAUX.ADDRULE(obj_name => 'IMPTEMP%', base_obj_property => TB_TEMP, obj_type =>
TYPE_TABLE, INCLUSION => TRUE);
2. DDLAUX.ADDRULE(base_obj_property => TB_TEMP, obj_type => TYPE_TABLE);
```

> **Note:**
>
> Since the `IMPTEMP%` tables must be included, that rule should come first.

The following example excludes all tables with name '`GGS%`'

```
DECLARE sno NUMBER; BEGIN sno := DDLAUX.ADDRULE(obj_name => 'GGS%' , obj_type =>
TYPE_TABLE); END
```

The following example excludes all temporary tables.

```
DDLAUX.ADDRULE(base_obj_property => TB_TEMP, obj_type => TYPE_TABLE);
```

The following example excludes all indexes on `TEMP` tables.

```
DDLAUX.ADDRULE(base_obj_property => TB_TEMP, obj_type => TYPE_INDEX);
```

The following example excludes all objects in schema `TKGGADMIN`.

```
DDLAUX.ADDRULE(owner_name => 'TKGGADMIN');
```

The following example excludes all objects in `TRUNCATE` operations made to `TEMP` tables.

```
DDLAUX.ADDRULE(base_obj_property => TB_TEMP, obj_type => TYPE_TABLE, command =>
CMD_TRUNCATE)
```

Dropping Filter Rules

Use the `DDLAUX.dropRule()` function with the drop rule. This function is installed in the Oracle GoldenGate DDL schema after the DDL objects are installed with the `ddl_setup.sql` script. As input, specify the serial number of the rule that you want to drop.

```
FUNCTION dropRule(sno IN NUMBER) RETURN BOOLEAN;
```

Filtering with the DDL Parameter

The `DDL` parameter is the main Oracle GoldenGate parameter for filtering DDL within the Extract and Replicat processes.

When used without options, the `DDL` parameter performs no filtering, and it causes all DDL operations to be propagated as follows:

- As an Extract parameter, it captures all supported DDL operations that are generated on all supported database objects and sends them to the trail.
- As a Replicat parameter, it replicates all DDL operations from the Oracle GoldenGate trail and applies them to the target. This is the same as the default behavior without this parameter.

When used with options, the `DDL` parameter acts as a filtering agent to include or exclude DDL operations based on:

- scope
- object type
- operation type
- object name
- strings in the DDL command syntax or comments, or both

Only one `DDL` parameter can be used in a parameter file, but you can combine multiple inclusion and exclusion options, along with other options, to filter the DDL to the required level.

- `DDL` filtering options are valid for a primary Extract that captures from the transaction source, but not for a data-pump Extract.
- When combined, multiple filter option specifications are linked logically as `AND` statements.
- All filter criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex DDL filtering criteria, it is recommended that you test your configuration in a test environment before using it in production.

For `DDL` parameter syntax and additional usage guidelines, see *Parameters and Functions Reference for Oracle GoldenGate*.

Note:

Before you configure DDL support, it might help to review How DDL is Evaluated for Processing.

Special Filter Cases

This topic describes the special cases that you must consider before creating your DDL filters.

The following are the special cases for creating filter conditions.

DDL EXCLUDE ALL

`DDL EXCLUDE ALL` is a special processing option that is intended primarily for Extract. `DDL EXCLUDE ALL` blocks the replication of DDL operations, but ensures that Oracle GoldenGate continues to keep the object metadata current. When Extract receives DDL directly from the logmining server (triggerless DDL capture mode), current metadata is always maintained.

You can use `DDL EXCLUDE ALL` when using a method other than Oracle GoldenGate to apply DDL to the target and you want Oracle GoldenGate to replicate data changes to the target objects. It provides the current metadata to Oracle GoldenGate as objects change, thus preventing the need to stop and start the Oracle GoldenGate processes. The following special conditions apply to `DDL EXCLUDE ALL`:

- `DDL EXCLUDE ALL` does not require the use of an `INCLUDE` clause.
- When using `DDL EXCLUDE ALL`, you can set the `WILDCARDRESOLVE` parameter to `IMMEDIATE` to allow immediate DML resolution if required.

To prevent all DDL metadata and operations from being replicated, omit the `DDL` parameter entirely.

Implicit DDL

User-generated DDL operations can generate implicit DDL operations. For example, the following statement generates two distinct DDL operations.

```
CREATE TABLE customers (custID number, name varchar2(50), address varchar2(75), address2
varchar2(75), city varchar2(50), state (varchar2(2), zip number, contact varchar2(50),
areacode number(3), phone number(7), primary key (custID));
```

The first (explicit) DDL operation is the `CREATE TABLE` statement itself.

The second DDL operation is an implicit `CREATE UNIQUE INDEX` statement that creates the index for the primary key. This operation is generated by the database engine, not a user application.

Guidelines for Filtering Implicit DDL

How to filter implicit DDL depends on the mechanism that you are using to filter DDL. See Filtering DDL Replication for more information.

- When the `DDL` parameter is used to filter DDL operations, Oracle GoldenGate filters out any implicit DDL by default, because the explicit DDL will generate the implicit DDL on the target. For example, the target database will create the appropriate index when the `CREATE TABLE` statement in the preceding example is applied by Replicat.

- - If your filtering rules exclude the explicit DDL from being propagated, you must also create a rule to exclude the implicit DDL. For example, if you exclude the `CREATE TABLE` statement in the following example, but do not exclude the implicit `CREATE UNIQUE INDEX` statement, the target database will try to create the index on a non-existent table.

    ```
    CREATE TABLE customers (custID number, name varchar2(50), address varchar2(75),
    address2 varchar2(75), city varchar2(50), state (varchar2(2), zip number,
    contact varchar2(50), areacode number(3), phone number(7), primary key (custID));
    ```

  - If your filtering rules permit the propagation of the explicit DDL, you do not need to exclude the implicit DDL. It will be handled correctly by Oracle GoldenGate and the target database.

How Oracle GoldenGate Handles Derived Object Names

DDL operations can contain a *base object* name and also a *derived object* name.

A base object is an object that contains data. A derived object is an object that inherits some attributes of the base object to perform a function related to that object. DDL statements that have both base and derived objects are:

- `RENAME` and `ALTER RENAME`
- `CREATE` and `DROP` on an index, synonym, or trigger

Consider the following DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

In this case, the table is the base object. Its name (`hr.tabPayroll`) is the *base name* and is subject to mapping with `TABLE` or `MAP` under the `MAPPED` scope. The derived object is the index, and its name (`hr.indexPayrollDate`) is the *derived name*.

You can map a derived name in its own `TABLE` or `MAP` statement, separately from that of the base object. Or, you can use one `MAP` statement to handle both. In the case of `MAP`, the conversion of derived object names on the target works as follows.

MAP Exists for Base Object, But Not Derived Object

If there is a `MAP` statement for the base object, but not for the derived object, the result is a schema based on the mapping that matches the derived object name. Derived objects are only mapped if the `MAPDERIVED` option is enabled, which is also the default option.

For example, consider the following:

Extract (source)

```
Table hr.*;
```

Replicat (target)

```
MAP hr.*, TARGET hrBackup.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.Payroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hrBackup.indexPayrollDate ON TABLE hrBackup.Payroll (payDate);
```

In this example, the mapping is such that it matches the derived object name because of which the derived object schema is changed from `hr` to `hrBackup`.

Here's another example, where there is no mapping that matches the derived object name so the derived object name remains the same.

Extract (source)

```
Table hr.tab*;
```

Replicat (target)

```
MAP hr.tab*, TARGET hrBackup.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hrBackup.tabPayroll (payDate);
```

MAP Exists for Base and Derived Objects

If there is a `MAP` statement for the base object and also one for the derived object, the result is an explicit mapping. Assuming the DDL statement includes `MAPPED`, Replicat converts the schema and name of each object according to its own `TARGET` clause. For example, assume the following:

Extract (source)

```
TABLE hr.tab*;  TABLE hr.index*;
```

Replicat (target)

```
MAP hr.tab*, TARGET hrBackup.*;MAP hr.index*, TARGET hrIndex.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hrIndex.indexPayrollDate ON TABLE hrBackup.tabPayroll (payDate);
```

Use an explicit mapping when the index on the target must be owned by a different schema from that of the base object, or when the name on the target must be different from that of the source.

MAP Exists for Derived Object, But Not Base Object

If there is a `MAP` statement for the derived object, but not for the base object, Replicat does not perform any name conversion for either object. The target DDL statement is the same as that of the source. To map a derived object, the choices are:

- Use an explicit `MAP` statement for the base object.
- If names permit, map both base and derived objects in the same `MAP` statement by means of a wildcard.
- Create a `MAP` statement for each object, depending on how you want the names converted.

New Tables as Derived Objects

The following explains how Oracle GoldenGate handles new tables that are created from:

- `RENAME` and `ALTER RENAME`
- `CREATE TABLE AS SELECT`

CREATE TABLE AS SELECT

The `CREATE TABLE AS SELECT` (CTAS) statements include `SELECT` statements and `INSERT` statements that reference any number of underlying objects. By default, Oracle GoldenGate obtains the data for the `AS SELECT` clause from the target database. You can force the CTAS operation to preserve the original inserts using this parameter.

> **Note:**
>
> For this reason, Oracle `XMLType` tables created from a `CTAS` (`CREATE TABLE AS SELECT`) statement cannot be supported. For `XMLType` tables, the row object IDs must match between source and target, which cannot be maintained in this scenario. `XMLType` tables created by an empty `CTAS` statement (that does not insert data in the new table) can be maintained correctly.
>
> In addition, you could use the `GETCTASDML` parameter that allows CTAS to replay the inserts of the CTAS thus preserving OIDs during replication. This parameter is only supported with Integrated Dictionary and any downstream Replicat must be 12.1.2.1 or greater to consume the trail otherwise, there may be divergence.

The objects in the `AS SELECT` clause must exist in the target database, and their names must be identical to the ones on the source.

In a `MAP` statement, Oracle GoldenGate only maps the name of the new table (`CREATE TABLE name`) to the `TARGET` specification, but does not map the names of the underlying objects from the `AS SELECT` clause. There could be dependencies on those objects that could cause data inconsistencies if the names were converted to the `TARGET` specification.

The following shows an example of a `CREATE TABLE AS SELECT` statement on the source and how it would be replicated to the target by Oracle GoldenGate.

```
CREATE TABLE a.tab1 AS SELECT * FROM a.tab2;
```

The `MAP` statement for Replicat is as follows:

```
MAP a.tab*, TARGET a.x*;
```

The target DDL statement that is applied by Replicat is the following:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.tab2;
```

The name of the table in the `AS SELECT * FROM` clause remains as it was on the source: `tab2` (rather than `xtab2`).

To keep the data in the underlying objects consistent on source and target, you can configure them for data replication by Oracle GoldenGate. In the preceding example, you could use the following statements to accommodate this requirement:

Source

```
TABLE a.tab*;
```

Target

```
MAPEXCLUDE a.tab2
MAP a.tab*, TARGET a.x*;
MAP a.tab2, TARGET a.tab2;
```

See Correctly Identifying Unqualified Object Names in DDL.

RENAME and ALTER TABLE RENAME

In `RENAME` and `ALTER TABLE RENAME` operations, the base object is always the new table name. In the following example, the base object name is considered to be `index_paydate`.

```
ALTER TABLE hr.indexPayrollDate RENAME TO index_paydate;
```

or...

```
RENAME hr.indexPayrollDate TO index_paydate;
```

The derived object name is `hr.indexPayrollDate`.

Disabling the Mapping of Derived Objects

Use the `DDLOPTIONS` parameter with the `NOMAPDERIVED` option to prevent the conversion of the name of a derived object according to a `TARGET` clause of a `MAP` statement that includes it. `NOMAPDERIVED` overrides any explicit `MAP` statements that contain the name of the base or derived object. Source DDL that contains derived objects is replicated to the target with the same schema and object names as on the source.

The following table shows the results of `MAPDERIVED` compared to `NOMAPDERIVED`, based on whether there is a `MAP` statement just for the base object, just for the derived object, or for both.

Base Object	Derived Object	MAP/NOMAP DERIVED?	Derived object converted per a MAP?	Derived object gets schema of base object?
mapped[1]	mapped	`MAPDERIVED`	yes	no
mapped	not mapped	`MAPDERIVED`	no	yes
not mapped	mapped	`MAPDERIVED`	no	no
not mapped	not mapped	`MAPDERIVED`	no	no
mapped	mapped	`NOMAPDERIVED`	no	no
mapped	not mapped	`NOMAPDERIVED`	no	no
not mapped	mapped	`NOMAPDERIVED`	no	no
not mapped	not mapped	`NOMAPDERIVED`	no	no

[1] Mapped means included in a `MAP` statement.

The following examples illustrate the results of `MAPDERIVED` as compared to `NOMAPDERIVED`. In the following table, both trigger and table are owned by `rpt` on the target because both base and derived names are converted by means of `MAPDERIVED`.

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER rpt.act_trig ON rpt.acct;`

In the following table, the trigger is owned by `fin`, because conversion is prevented by means of `NOMAPDERIVED`.

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER fin.act_trig ON rpt.acct;`

Note:

In the case of a `RENAME` statement, the new table name is considered to be the base table name, and the old table name is considered to be the derived table name.

Using DDL String Substitution

You can substitute strings within a DDL operation while it is being processed by Oracle GoldenGate.

This feature provides a convenience for changing and mapping directory names, comments, and other things that are not directly related to data structures. For example, you could substitute one tablespace name for another, or substitute a string within comments. String substitution is controlled by the `DDLSUBST` parameter. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Note:

Before you create a `DDLSUBST` parameter statement, it might help to review How DDL is Evaluated for Processing in this chapter.

Controlling the Propagation of DDL to Support Different Topologies

To support bidirectional and cascading replication configurations, it is important for Extract to be able to identify the DDL that is performed by Oracle GoldenGate and by other applications, such as the local business applications.

Depending on the configuration that you want to deploy, it might be appropriate to capture one or both of these sources of DDL on the local system.

> **Note:**
>
> Oracle GoldenGate DDL consists of `ALTER TABLE` statements performed by Extract to create log groups and the DDL that is performed by Replicat to replicate source DDL changes.

The following options of the `DDLOPTIONS` parameter control whether DDL on the local system is captured by Extract and then sent to a remote system, assuming Oracle GoldenGate DDL support is configured and enabled:

- The `GETREPLICATES` and `IGNOREREPLICATES` options control whether Extract captures or ignores the DDL that is generated by Oracle GoldenGate. The default is `IGNOREREPLICATES`, which does not propagate the DDL that is generated by Oracle GoldenGate. To identify the DDL operations that are performed by Oracle GoldenGate, the following comment is part of each Extract and Replicat DDL statement:

  ```
  /* GOLDENGATE_DDL_REPLICATION */
  ```

- The `GETAPPLOPS` and `IGNOREAPPLOPS` options control whether Extract captures or ignores the DDL that is generated by applications other than Oracle GoldenGate. The default is `GETAPPLOPS`, which propagates the DDL from local applications (other than Oracle GoldenGate).

The result of these default settings is that Extract ignores its own DDL and the DDL that is applied to the local database by a local Replicat, so that the DDL is not sent back to its source, and Extract captures all other DDL that is configured for replication. The following is the default `DDLOPTIONS` configuration.

```
DDLOPTIONS GETAPPLOPS, IGNOREREPLICATES
```

Propagating DDL in Active-Active (Bidirectional) Configurations

Oracle GoldenGate supports active-active DDL replication between two systems. For an active-active bidirectional replication, the following must be configured in the Oracle GoldenGate processes:

1. DDL that is performed by a business application on one system must be replicated to the other system to maintain synchronization. To satisfy this requirement, include the `GETAPPLOPS` option in the `DDLOPTIONS` statement in the Extract parameter files on both systems.
2. DDL that is applied by Replicat on one system must be captured by the local Extract and sent back to the other system. To satisfy this requirement, use the `GETREPLICATES` option in the `DDLOPTIONS` statement in the Extract parameter files on both systems.

   > **Note:**
   >
   > An internal Oracle GoldenGate token will cause the actual Replicat DDL statement itself to be ignored to prevent loopback. The purpose of propagating Replicat DDL back to the original system is so that the Replicat on that system can update its object metadata cache, in preparation to receive incoming DML, which will have the new metadata.

3. Each Replicat must be configured to update its object metadata cache whenever the remote Extract sends over a captured Replicat DDL statement. To satisfy this requirement, use the `UPDATEMETADATA` option in the `DDLOPTIONS` statement in the Replicat parameter files on both systems.

The resultant `DDLOPTIONS` statements should look as follows:

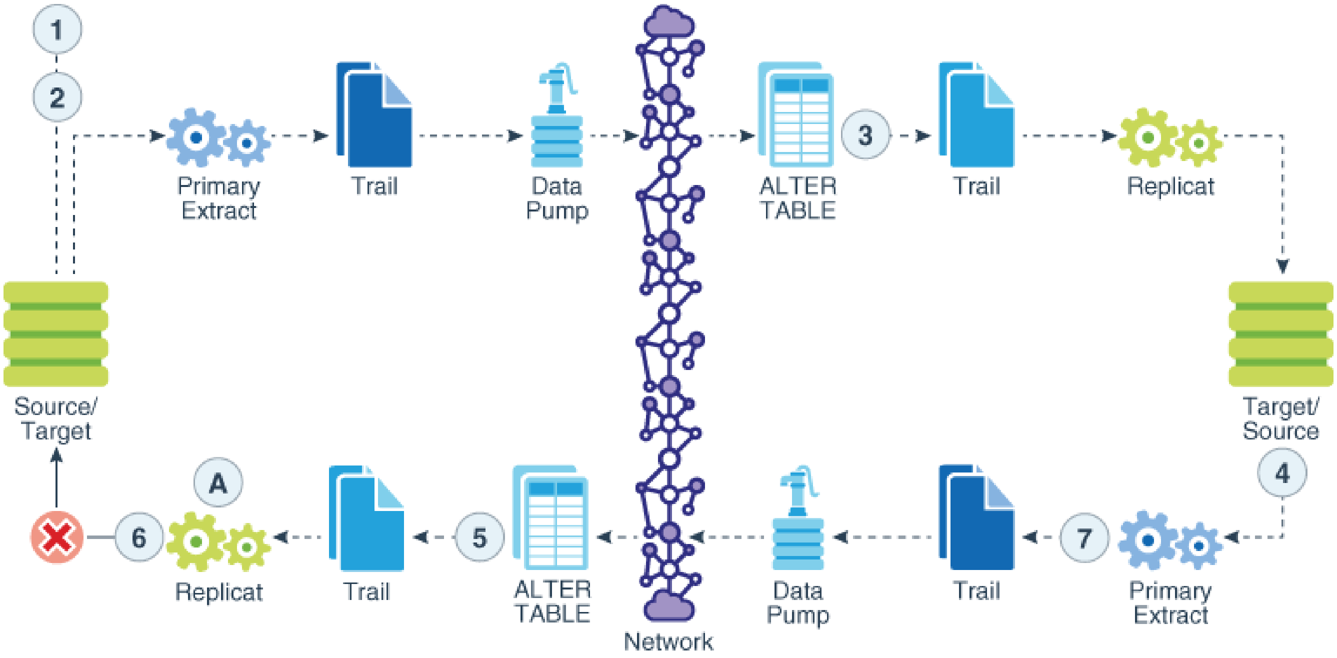
Extract (primary and secondary)

```
DDLOPTIONS GETREPLICATES, GETAPPLOPS
```

Replicat (primary and secondary)

```
DDLOPTIONS UPDATEMETADATA
```

> **WARNING:**
>
> Before you allow new DDL or DML to be issued for the same object(s) as the original DDL, allow time for the original DDL to be replicated to the remote system and then captured again by the Extract on that system. This will ensure that the operations arrive in correct order to the Replicat on the original system, to prevent DML errors caused by metadata inconsistencies. See the following diagram for more information.



The labels in the diagrams imply the following:

- 1: `ALTER TABLE` Customer ADD Birth_Date date
- 2; New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date

- 3: `ALTER TABLE`
- 4: New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date
- 5: `ALTER TABLE`
- 6: `DDLOPTIONS UPDATEMETADATA` New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date

For more information about `DDLOPTIONS`, see *Parameters and Functions Reference for Oracle GoldenGate*.

For more information about configuring a bidirectional configuration, see Quickstart Bidirectional Replication.

Propagating DDL in a Cascading Configuration

In a cascading configuration, use the following setting for `DDLOPTIONS` in the Extract parameter file on each intermediary system. This configuration forces Extract to capture the DDL from Replicat on an intermediary system and cascade it to the next system downstream.

```
DDLOPTIONS GETREPLICATES, IGNOREAPPLOPS
```

For more information about `DDLOPTIONS`, see `DDLOPTIONS` in *Parameters and Functions Reference for Oracle GoldenGate*.

Adding Supplemental Log Groups Automatically

Use the `DDLOPTIONS` parameter with the `ADDTRANDATA` option for performing tasks described in this topic. You can perform the following tasks using the `DDLOPTIONS`:

- Enable Oracle's supplemental logging automatically for new tables created with a `CREATE TABLE`.
- Update Oracle's supplemental logging for tables affected by an `ALTER TABLE` to add or drop columns.
- Update Oracle's supplemental logging for tables that are renamed.
- Update Oracle's supplemental logging for tables where unique or primary keys are added or dropped.

To use `DDLOPTIONS ADDSCHEMATRANDATA`, the `ADD SCHEMATRANDATA` command must be issued in GGSCI to enable schema-level supplemental logging.

By default, the `ALTER TABLE` that adds the supplemental logging is not replicated to the target unless the `GETREPLICATES` parameter is in use.

`DDLOPTIONS ADDTRANDATA` is not supported for multitenant container databases, see Configuring Logging Properties for more information.

Removing Comments from Replicated DDL

You can use the `DDLOPTIONS` parameter with the `REMOVECOMMENTS BEFORE` and `REMOVECOMMENTS AFTER` options to prevent comments that were used in the source DDL from being included in the target DDL.

By default, comments are not removed, so that they can be used for string substitution.

Replicating an IDENTIFIED BY Password

Use the `DDLOPTIONS` parameter with the `DEFAULTUSERPASSWORDALIAS` and `REPLICATEPASSWORD | NOREPLICATEPASSWORD` options to control how the password of a replicated `{CREATE | ALTER} USER` *`name`* `IDENTIFIED BY` *`password`* statement is handled. These options must be used together.

See the `USEPASSWORDVERIFIERLEVEL` option of `DDLOPTIONS` for important information about specifying the password verifier when Replicat operates against an Oracle 10*g* or 11*g* database.

> **Note:**
>
> Replication of `CREATE | ALTER PROFILE` will fail as the profile/password verification function must exist in the SYS schema. To replicate these DDLs successfully, password verification function must be created manually on both source/target(s) since DDL to SYS schema is excluded.

How DDL is Evaluated for Processing

This topic explains how Oracle GoldenGate processes DDL statements on the source and target systems.

It shows the order in which different criteria in the Oracle GoldenGate parameters are processed, and it explains the differences between how Extract and Replicat each process the DDL.

Extract

1. Extract captures a DDL statement.
2. Extract separates comments, if any, from the main statement.
3. Extract searches for the `DDL` parameter. (This example assumes it exists.)
4. Extract searches for the `IGNOREREPLICATES` parameter. If it is present, and if Replicat produced this DDL on this system, Extract ignores the DDL statement. (This example assumes no Replicat operations on this system.)
5. Extract determines whether the DDL statement is a `RENAME`. If so, the rename is flagged internally.
6. Extract gets the base object name and, if present, the derived object name.
7. If the statement is a `RENAME`, Extract changes it to `ALTER TABLE RENAME`.
8. Extract searches for the `DDLOPTIONS REMOVECOMMENTS BEFORE` parameter. If it is present, Extract removes the comments from the DDL statement, but stores them in case there is a `DDL INCLUDE` or `DDL EXCLUDE` clause that uses `INSTR` or `INSTRCOMMENTS`.
9. Extract determines the DDL scope: `MAPPED`, `UNMAPPED` or `OTHER`:
    - It is `MAPPED` if the operation and object types are supported for mapping, and the base object name and/or derived object name (if `RENAME`) is in a `TABLE` parameter.
    - It is `UNMAPPED` if the operation and object types are not supported for mapping, and the base object name and/or derived object name (if `RENAME`) is not in a `TABLE` parameter.

- Otherwise the operation is identified as `OTHER`.

10. Extract checks the `DDL` parameter for `INCLUDE` and `EXCLUDE` clauses, and it evaluates the `DDL` parameter criteria in those clauses. All options must evaluate to `TRUE` in order for the `INCLUDE` or `EXCLUDE` to evaluate to `TRUE`. The following occurs:
    - If an `EXCLUDE` clause evaluates to `TRUE`, Extract discards the DDL statement and evaluates another DDL statement. In this case, the processing steps start over.
    - If an `INCLUDE` clause evaluates to `TRUE`, or if the `DDL` parameter does not have any `INCLUDE` or `EXCLUDE` clauses, Extract includes the DDL statement, and the processing logic continues.
11. Extract searches for a `DDLSUBST` parameter and evaluates the `INCLUDE` and `EXCLUDE` clauses. If the criteria in those clauses add up to `TRUE`, Extract performs string substitution. Extract evaluates the DDL statement against each `DDLSUBST` parameter in the parameter file. For all true `DDLSUBST` specifications, Extract performs string substitution in the order that the `DDLSUBST` parameters are listed in the file.
12. Now that `DDLSUBT` has been processed, Extract searches for the `REMOVECOMMENTS AFTER` parameter. If it is present, Extract removes the comments from the DDL statement.
13. Extract searches for `DDLOPTIONS ADDTRANDATA`. If it is present, and if the operation is `CREATE TABLE`, Extract issues the `ALTER TABLE` *`name`* `ADD SUPPLEMENTAL LOG GROUP` command on the table.
14. Extract writes the DDL statement to the trail.

Replicat

1. Replicat reads the DDL statement from the trail.
2. Replicat separates comments, if any, from the main statement.
3. Replicat searches for `DDLOPTIONS REMOVECOMMENTS BEFORE`. If it is present, Replicat removes the comments from the DDL statement.
4. Replicat evaluates the DDL synchronization scope to determine if the DDL qualifies for name mapping. Anything else is of `OTHER` scope.
5. Replicat evaluates the `MAP` statements in the parameter file. If the source base object name for this DDL (as read from the trail) appears in any of the `MAP` statements, the operation is marked as `MAPPED` in scope. Otherwise it is marked as `UNMAPPED` in scope.
6. Replicat replaces the source base object name with the base object name that is specified in the `TARGET` clause of the `MAP` statement.
7. If there is a derived object, Replicat searches for `DDLOPTIONS MAPDERIVED`. If it is present, Replicat replaces the source derived name with the target derived name from the `MAP` statement.
8. Replicat checks the `DDL` parameter for `INCLUDE` and `EXCLUDE` clauses, and it evaluates the `DDL` parameter criteria contained in them. All options must evaluate to `TRUE` in order for the `INCLUDE` or `EXCLUDE` to evaluate to `TRUE`. The following occurs:
    - If any `EXCLUDE` clause evaluates to `TRUE`, Replicat discards the DDL statement and starts evaluating another DDL statement. In this case, the processing steps start over.
    - If any `INCLUDE` clause evaluates to `TRUE`, or if the `DDL` parameter does not have any `INCLUDE` or `EXCLUDE` clauses, Replicat includes the DDL statement, and the processing logic continues.

9. Replicat searches for the `DDLSUBST` parameter and evaluates the `INCLUDE` and `EXCLUDE` clauses. If the options in those clauses add up to `TRUE`, Replicat performs string substitution. Replicat evaluates the DDL statement against each `DDLSUBST` parameter in the parameter file. For all true `DDLSUBST` specifications, Replicat performs string substitution in the order that the `DDLSUBST` parameters are listed in the file.
10. Now that `DDLSUBT` has been processed, Replicat searches for the `REMOVECOMMENTS AFTER` parameter. If it is present, Replicat removes the comments from the DDL statement.
11. Replicat executes the DDL statement on the target database.
12. If there are no errors, Replicat processes the next DDL statement. If there are errors, Replicat performs the following steps.
13. Replicat analyzes the `INCLUDE` and `EXCLUDE` rules in the Replicat `DDLERROR` parameters in the order that they appear in the parameter file. If Replicat finds a rule for the error code, it applies the specified error handling; otherwise, it applies `DEFAULT` handling.
14. If the error handling does not enable the DDL statement to succeed, Replicat does one of the following: abends, ignores the operation, or discards it as specified in the rules.

> **Note:**
>
> If there are multiple targets for the same source in a `MAP` statement, the processing logic executes for each one.

Viewing DDL Report Information

By default, Oracle GoldenGate shows basic statistics about DDL at the end of the Extract and Replicat reports.

To enable expanded DDL reporting, use the `DDLOPTIONS` parameter with the `REPORT` option. Expanded reporting includes the following information about DDL processing:

- A step-by-step history of the DDL operations that were processed by Oracle GoldenGate.
- The DDL filtering and processing parameters that are being used.

Expanded DDL report information increases the size of the report file, but it might be useful in certain situations, such as for troubleshooting or to determine when an `ADD TRANDATA` to add supplemental logging was applied.

To view a report, use the `VIEW REPORT` command.

```
VIEW REPORT group
```

Viewing DDL Reporting in Replicat

The Replicat report lists:

- The entire syntax and source Oracle GoldenGate SCN of each DDL operation that Replicat processed from the trail. You can use the source SCN for tracking purposes, especially when there are restores from backup and Replicat is positioned backward in the trail.
- A subsequent entry that shows the scope of the operation (`MAPPED`, `UNMAPPED`, `OTHER`) and how object names were mapped in the target DDL statement, if applicable.

- Another entry that shows how processing criteria was applied.
- Additional entries that show whether the operation succeeded or failed, and whether or not Replicat applied error handling rules.

The following excerpt from a Replicat report illustrates a sequence of steps, including error handling:

```
2011-01-20 15:11:45  GGS INFO     2104  DDL found, operation [drop table myTableTemp ],
Source SCN [1186713.0].
 2011-01-20 15:11:45  GGS INFO     2100  DDL is of mapped scope, after mapping new
operation [drop table "QATEST2"."MYTABLETEMP" ].
 2011-01-20 15:11:45  GGS INFO     2100  DDL operation included [include objname
myTable*], optype [DROP], objtype [TABLE], objname [QATEST2.MYTABLETEMP].
 2011-01-20 15:11:45  GGS INFO     2100  Executing DDL operation.
 2011-01-20 15:11:48  GGS INFO     2105  DDL error ignored for next retry: error code
[942], filter [include objname myTableTemp], error text [ORA-00942: table or view does
not exist], retry [1].
 2011-01-20 15:11:48  GGS INFO     2100  Executing DDL operation , trying again due to
RETRYOP parameter.
 2011-01-20 15:11:51  GGS INFO     2105  DDL error ignored for next retry: error code
[942], filter [include objname myTableTemp], error text [ORA-00942: table or view does
not exist], retry [2].
 2011-01-20 15:11:51  GGS INFO     2100  Executing DDL operation, trying again due to
RETRYOP parameter.
 2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored for next retry: error code
[942], filter [include objname myTableTemp], error text [ORA-00942: table or view does
not exist], retry [3].
 2011-01-20 15:11:54  GGS INFO     2100  Executing DDL operation, trying again due to
RETRYOP parameter.
 2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored: error code [942], filter
[include objname myTableTemp], error text [ORA-00942: table or view does not exist].
```

Viewing DDL Reporting in Extract

The Extract report lists the following:

- The entire syntax of each captured DDL operation, the start and end SCN, the Oracle instance, the DDL sequence number (from the `SEQNO` column of the history table), and the size of the operation in bytes.
- A subsequent entry that shows how processing criteria was applied to the operation, for example string substitution or `INCLUDE` and `EXCLUDE` filtering.
- Another entry showing whether the operation was written to the trail or excluded.

The following, taken from an Extract report, shows an included operation and an excluded operation. There is a report message for the included operation, but not for the excluded one.

```
2011-01-20 15:11:41  GGS INFO     2100  DDL found, operation [create table myTable (
    myId number (10) not null,
    myNumber number,
    myString varchar2(100),
    myDate date,
    primary key (myId)
) ], start SCN [1186754], commit SCN [1186772] instance [test11g (1)], DDL seqno [4134].

2011-01-20 15:11:41  GGS INFO     2100  DDL operation included [INCLUDE OBJNAME
myTable*], optype [CREATE], objtype [TABLE], objname [QATEST1.MYTABLE].

2011-01-20 15:11:41  GGS INFO     2100  DDL operation written to extract trail file.

2011-01-20 15:11:42  GGS INFO     2100  Successfully added TRAN DATA for table with the
```

```
key, table [QATEST1.MYTABLE], operation [ALTER TABLE "QATEST1"."MYTABLE" ADD
SUPPLEMENTAL LOG GROUP "GGS_MYTABLE_53475" (MYID) ALWAYS  /* GOLDENGATE_DDL_REPLICATION
*/ ].

2011-01-20 15:11:43  GGS INFO     2100  DDL found, operation [create table myTableTemp (
    vid varchar2(100),
    someDate date,
    primary key (vid)
) ], start SCN [1186777], commit SCN [1186795] instance [test11g (1)], DDL seqno [4137].

2011-01-20 15:11:43  GGS INFO     2100  DDL operation excluded [EXCLUDE OBJNAME
myTableTemp OPTYPE CREATE], optype [CREATE], objtype [TABLE], objname
[QATEST1.MYTABLETEMP].
```

Statistics in the Process Reports

You can send current statistics for DDL processing to the Extract and Replicat reports by using the `SEND` command in GGSCI.

```
SEND {EXTRACT | REPLICAT} group REPORT
```

The statistics show totals for:

- All DDL operations
- Operations that are `MAPPED` in scope
- Operations that are `UNMAPPED` in scope
- Operations that are `OTHER` in scope
- Operations that were excluded (number of operations minus included ones)
- Errors (Replicat only)
- Retried errors (Replicat only)
- Discarded errors (Replicat only)
- Ignored operations (Replicat only)

Tracing DDL Processing

If you open a support case with Oracle GoldenGate Technical Support, you might be asked to turn on tracing. `TRACE` and `TRACE2` control DDL tracing.

Using Tools that Support Trigger-Based DDL Capture

This section documents the additional tools available to support trigger-based capture.

Tracing the DDL Trigger

To trace the activity of the Oracle GoldenGate DDL trigger, use the following tools.

- `ggs_ddl_trace.log` trace file: Oracle GoldenGate creates a trace file in the `USER_DUMP_DEST` directory of Oracle. On RAC, each node has its own trace file that captures DDL tracing for that node. You can query the trace file as follows:

  ```
  select value from sys.v_$parameter where name = 'user_dump_dest';
  ```

- `ddl_tracelevel` script: Edit and run this script to set the trace level. A value of `None` generates no DDL tracing, except for fatal errors and installation logging. The default value of `0` generates minimal tracing information. A value of `1` or `2` generates a much larger amount of information in the trace file. Do not use `1` or `2` unless requested to do so by a Oracle GoldenGate Technical Support analyst as part of a support case.

- `ddl_cleartrace` script: Run this script on a regular schedule to prevent the trace file from consuming excessive disk space as it expands. It deletes the file, but Oracle GoldenGate will create another one. The DDL trigger stops writing to the trace file when the Oracle directory gets low on space, and then resumes writing when space is available again. This script is in the Oracle GoldenGate directory. Back up the trace file before running the script.

Viewing Metadata in the DDL History Table

Use the `DUMPDDL` command in GGSCI to view the information that is contained in the DDL history table. This information is stored in proprietary format, but you can export it in human-readable form to the screen or to a series of SQL tables that you can query. The information in the DDL history table is the same as that used by the Extract process.

Handling DDL Trigger Errors

Use the `params.sql` non-executable script to handle failures of the Oracle GoldenGate DDL trigger in relation to whether the source DDL fails or succeeds. The `params.sql script` is in the root Oracle GoldenGate directory. The parameters to use are the following:

- **`ddl_fire_error_in_trigger`**: If set to `TRUE`, failures of the Oracle GoldenGate DDL trigger are raised with a Oracle GoldenGate error message and a database error message to the source end-user application. The source operations fails.

  If set to `FALSE`, no errors are raised, and a message is written to the trigger trace file in the Oracle GoldenGate directory. The source operation succeeds, but no DDL is replicated. The target application will eventually fail if subsequent data changes do not match the old target object structure. The default is `FALSE`.
- **`ddl_cause_error`**: If set to `TRUE`, tests the error response of the trigger by deliberately causing an error. To generate the error, Oracle GoldenGate attempts to `SELECT` zero rows without exception handling. Revert this flag to the default of `FALSE` after testing is done.

Installing Trigger-Based DDL Capture

This appendix contains instructions for installing the objects that support the trigger-based method of Oracle GoldenGate DDL support.
To configure Oracle GoldenGate to capture and replicate DDL, see Configuring DDL Support .

> **Note:**
>
> DDL support for sequences (`CREATE`, `ALTER`, `DROP`, `RENAME`) is compatible with, but not required for, replicating the sequence values themselves. To replicate just sequence values, you do not need to install the Oracle GoldenGate DDL support environment. You can just use the `SEQUENCE` parameter in the Extract configuration.

When to Use Trigger-based DDL Capture

This topic describes the configuration where you must use trigger-based DDL Extract.

You *must* use trigger-based DDL capture when Extract will operate in the following configurations:

Extract operates in classic capture mode against any version of Oracle Database.
If Extract will run in integrated mode against a version 11.2.0.4 or later Oracle Database, the DDL trigger is not required. By default, DDL capture is handled transparently through the database logmining server.

If Extract will capture from a multitenant container database, integrated capture mode must be used with the native DDL capture method.

See About Extract for more information about capture modes.

See Configuring DDL Support for more information about configuring DDL support.

Overview of the Objects that Support Trigger-based DDL Capture

This topic lists the requirements for installing Oracle GoldenGate trigger-based DDL environment.

To install the Oracle GoldenGate trigger-based DDL environment, you will be installing the database objects listed in the following table.

Object	Purpose	Default name
DDL marker table	Stores DDL information. This table only receives inserts.	`GGS_MARKER`
Sequence on marker table	Used for a column in the marker table.	`GGS_DDL_SEQ`
DDL history table	Stores object metadata history. This table receives inserts, updates, deletes.	`GGS_DDL_HIST`
Object ID history table	Contains object IDs of configured objects.	`GGS_DDL_HIST_ALT`
DDL trigger	Fires on DDL operations. Writes information about the operation to the marker and history tables. Installed with the trigger are some packages.	`GGS_DDL_TRIGGER_BEFORE`
DDL schema	Contains the DDL synchronization objects.	None; must be specified during installation and in the `GLOBALS` file.
User role	Establishes the role needed to execute DDL operations.	`GGS_GGSUSER_ROLE`
Internal setup table	Database table for internal use only.	`GGS_SETUP`
`ddl_pin`	Pins DDL tracing, the DDL package, and the DDL trigger for performance improvements.	`ddl_pin`
`ddl_cleartrace.sql`	Removes the DDL trace file.	`ddl_cleartrace.sql`
`ddl_status.sql`	Verifies that the Oracle GoldenGate DDL objects are installed	`ddl_status.sql`

Object	Purpose	Default name
`marker_status.sql`	Verifies that the marker table is installed.	`marker_status.sql`
`ddl_tracelevel.sql`	Sets the level for DDL tracing.	`ddl_tracelevel.sql`

Installing the DDL Objects

To install DDL objects, you need scripts to perform various tasks during the installation.

These scripts are located in the installation directory of Oracle GoldenGate Microservices Architecture. The specific location is: `oggma_install_home/lib/sql/legacy`.

Follow these steps to install the database objects that support Oracle GoldenGate DDL capture.

> **Note:**
>
> When using Extract in classic mode to capture in an Active Data Guard environment, the DDL objects must be installed on the source database, not the standby.

1. Choose a schema that can contain the Oracle GoldenGate DDL objects. This schema cannot be case-sensitive.
2. Grant the following permission to the Oracle GoldenGate schema.

   ```
   GRANT EXECUTE ON utl_file TO schema;
   ```

3. Create a default tablespace for the Oracle GoldenGate DDL schema. This tablespace must be dedicated to the DDL schema; do not allow any other schema to share it.
4. Set `AUTOEXTEND` to `ON` for the DDL tablespace, and size it to accommodate the growth of the `GGS_DDL_HIST` and `GGS_MARKER` tables. The `GGS_DDL_HIST` table, in particular, will grow in proportion to overall DDL activity.
5. (Optional) To cause user DDL activity to fail when the DDL tablespace fills up, edit the `params.sql` script and set the `ddl_fire_error_in_trigger` parameter to `TRUE`. Extract cannot capture DDL if the tablespace fills up, so stopping the DDL gives you time to extend the tablespace size and prevent the loss of DDL capture. Managing tablespace sizing this way, however, requires frequent monitoring of the business applications and Extract to avoid business disruptions. As a best practice, make certain to size the tablespace appropriately in the first place, and set `AUTOEXTEND` to `ON` so that the tablespace does not fill up.

   > **WARNING:**
   >
   > Make a backup of the `params.sql` script before you edit it to preserve its original state.

6. Create a `GLOBALS` file (or edit it, if one exists).

   ```
   EDIT PARAMS ./GLOBALS
   ```

> **Note:**
>
> `EDIT PARAMS` creates a simple text file. When you save the file after `EDIT PARAMS`, it is saved with the name `GLOBALS` in upper case, without a file extension, at the root of the Oracle GoldenGate directory. Do not alter the file name or location.

7. In the `GLOBALS` file, specify the name of the DDL schema by adding the following parameter to the `GLOBALS` file.

   ```
   GGSCHEMA schema_name
   ```

8. (Optional) To change the names of other objects listed in DDL synchronization objects, *the changes must be made now*, before proceeding with the rest of the installation. Otherwise, you will need to stop Oracle GoldenGate DDL processing and reinstall the DDL objects. It is recommended that you accept the default names of the database objects. To change any database object name (except the schema), do one or both of the following:

   - Record all name changes in the `params.sql` script. Edit this script and change the appropriate parameters. Do not run this script.
   - List the names shown in Table 8-1 in the `GLOBALS` file. The correct parameters to use are listed in the **Parameter** column of the table.

   Table 8-1 GLOBALS Parameters for Changing DDL Object Names

Object	Parameter
Marker table	`MARKERTABLE` *`new_table_name`*[1]
History table	`DDLTABLE` *`new_table_name`*

   [1] Do not qualify the name of any of these tables. The schema name for these table must be either the one that is specified with `GGSCHEMA` or the schema of the current user, if `GGSCHEMA` is not specified in `GLOBALS`.

9. To enable trigger-based DDL replication to recognize Oracle invisible indexes as unique identifiers, set the following parameter to `TRUE` in the `params.sql` script:

   ```
   define allow_invisible_index_keys = 'TRUE'
   ```

10. Save and close the `GLOBALS` file and the `params.sql` file.
11. Change directories to the Oracle GoldenGate installation directory.
12. Exit all Oracle sessions, including those of SQL*Plus, those of business applications, those of the Oracle GoldenGate processes, and those of any other software that uses Oracle. Prevent the start of any new sessions.
13. Run SQL*Plus and log in as a user that has `SYSDBA` privilege. This privilege is required to install the DDL trigger in the `SYS` schema, which is required by Oracle. All other DDL objects are installed in the schema that you created.
14. Run the `marker_setup.sql` script. Supply the name of the Oracle GoldenGate schema when prompted, and then press **Enter** to execute the script. The script installs support for the Oracle GoldenGate DDL marker system.

    ```
    @marker_setup.sql
    ```

15. Run the `ddl_setup.sql` script. You are prompted to specify the name of the DDL schema that you configured. (Note: `ddl_setup.sql` will fail if the tablespace for this schema is

shared by any other users. It will not fail, however, if the default tablespace does not have `AUTOEXTEND` set to `ON`, the recommended setting.)

```
@ddl_setup.sql
```

16. Run the `role_setup.sql` script. At the prompt, supply the DDL schema name. The script drops and creates the role that is needed for DDL synchronization, and it grants DML permissions on the Oracle GoldenGate DDL objects.

    ```
    @role_setup.sql
    ```

17. Grant the role that was created (default name is `GGS_GGSUSER_ROLE`) to all Oracle GoldenGate Extract users. You may need to make multiple grants if the processes have different user names.

    ```
    GRANT role TO user;
    ```

18. Run the `ddl_enable.sql` script to enable the DDL trigger.

    ```
    @ddl_enable.sql
    ```

To Install and Use the Optional Performance Tool

To improve the performance of the DDL trigger, make the `ddl_pin` script part of the database startup. It must be invoked with the Oracle GoldenGate DDL user name, as in:

```
@ddl_pin DDL_user
```

This script pins the PL/SQL package that is used by the trigger into memory. If executing this script from SQL*Plus, connect as `SYSDBA` from the Oracle GoldenGate installation directory. This script relies on the Oracle `dmbs_shared_pool` system package, so install that package before using `ddl_pin`.

Configure DDL Modification for Oracle GoldenGate for Sybase

Use the following steps to modify DDL for Oracle GoldenGate Extract for Sybase:

1. Pause or stop capturing application data for tables where you want to modify the DDL.
2. Ensure Extract processes all the transactions prior to making any DDL changes. An Event Marker table may help to ensure full completion. See Maintaining the DDL Marker Table for reference.
3. Stop Extract.
4. At source, execute `DELETE TRANDATA` for the tables on which DDL has to be performed.
5. Execute the `ALTER TABLE` statement on the database side, to add or drop the column in or from the tables.
6. Re-enable trandata using the `ADD TRANDATA` command for the same tables at source.
7. Start Extract.

Using Procedural Replication

Learn about procedural replication and how to configure it.

About Procedural Replication

Oracle GoldenGate procedural replication is used to replicate Oracle Database supplied PL/SQL procedures avoiding the shipping and applying of high volume records usually generated by these operations. Procedural replication implements dictionary changes that control user and session behavior and the swapping of objects in dictionary.

Procedural replication is not related to the replication of the `CREATE`, `ALTER`, and `DROP` statements (or DDL), rather it is the replication of a procedure call like:

```
CALL procedure_name(arg1, arg2, ...);
```

As opposed to:

```
exec procedure_name(arg1, arg2, ...)
```

After you enable procedural replication, calls to procedures in Oracle Database supplied packages at one database are replicated to one or more other databases and then executed at those databases. For example, a call to subprograms in the `DBMS_REDEFINITION` package can perform an online redefinition of a table. If the table is replicated at several databases, and if you want the same online redefinition to be performed on the table at each database, then you can make the calls to the subprograms in the `DBMS_REDEFINITION` package at one database, and Oracle GoldenGate can replicate those calls to the other databases.

To support procedural replication, your Oracle Database should be configured to identify procedures that are enabled for this optimization.

To use procedural replication, the following prerequisites must be met:

- Oracle GoldenGate with Extract and Replicat.
- System supplied packages are only working in combination with DML and DDL.

Procedural Replication Process Overview

Procedural replication uses a trail record to ensure that sufficient information is encapsulated with the record.

To use Oracle GoldenGate procedural replication, you need to enable it. Your Oracle Database must have a built in mechanism to identify the procedures that are enabled for this optimization.

PL/SQL pragmas are used to indicate which procedures can be replicated. When the pragma is specified, a callback is made to Logminer on entry and exit from the routine. The callback provides the name of the procedure call and arguments and indicates if the procedure exited successfully or with an error. Logminer augments the redo stream with the information from the callbacks. For supported procedures, the normal redo generated by the procedure is suppressed, and only the procedure call is replicated.

A new trail record is generated to identify procedural replication. This trail record leverages existing trail column data format for arguments passed to PL/SQL procedures. For LOBs, data is passed in chunks similar to existing trail format for LOBs. This trail record has sufficient information to replay the procedure as-is on the target.

When you enable procedural replication, it prevents writing of individual records impacted by the procedure to the trail file.

If an error is encountered when applying a PL/SQL procedure, Replicat can replay the entire PL/SQL procedure.

Enabling Procedural Replication

Procedural replication is disabled by default. You can enable it by setting the `TRANLOGOPTIONS` option, `ENABLE_PROCEDURAL_REPLICATION`, to yes.

Once you enable the procedural option for one Extract, it remains on and can not be disabled.

If you want to use Oracle GoldenGate in an Oracle Database Vault environment with procedural replication, then you must set the appropriate privileges.

To enable procedural replication:

1. Ensure that you are in triggerless mode, see Prerequisites for Configuring DDL.
2. Connect to the source database as an Oracle GoldenGate administrator with `dblogin`.
3. Set the `TRANLOGOPTIONS` parameter option to yes.

   ```
   TRANLOGOPTIONS INTEGRATEDPARAMS (ENABLE_PROCEDURAL_REPLICATION Y)
   ```

Procedural replication is enabled for Extract.

Determining Whether Procedural Replication Is On

Use the `GG_PROCEDURE_REPLICATION_ON` function in the `DBMS_GOLDENGATE_ADM` package to determine whether Oracle GoldenGate procedural replication is on or off.

If you want to use Oracle GoldenGate in an Oracle Database Vault environment with procedural replication, then you must set the appropriate privileges. See *Oracle Database Vault Administrator’s Guide*.

To enable procedural replication:

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Run the `GG_PROCEDURE_REPLICATION_ON` function.

Example 8-1 Running the `GG_PROCEDURE_REPLICATION_ON` Function

```
SET SERVEROUTPUT ON
DECLARE
  on_or_off   NUMBER;
BEGIN
  on_or_off := DBMS_GOLDENGATE_ADM.GG_PROCEDURE_REPLICATION_ON;
  IF on_or_off=1 THEN
    DBMS_OUTPUT.PUT_LINE('Oracle GoldenGate procedural replication is ON.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Oracle GoldenGate procedural replication is OFF.');
  END IF;
END;
/
```

Enabling and Disabling Supplemental Logging

Oracle GoldenGate provides GGSCI commands to allow you to enable or disable procedural supplemental logging.

To enable supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Add supplemental logging for procedural replication.

   ```
   ADD PROCEDURETRANDATA
   ```

   ```
   INFO OGG-13005 PROCEDURETRANDATA supplemental logging has been enabled.
   ```

Supplemental logging is enabled for procedure replication.

To disable supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Remove supplemental logging for procedure replication.

   ```
   DELETE PROCEDURETRANDATA
   ```

Supplemental logging is disabled for procedure replication.

To view information about supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Display supplemental logging information for procedure replication.

   ```
   INFO PROCEDURETRANDATA
   ```

Supplemental logging information for procedure replication is displayed.

Filtering Features for Procedural Replication

You can specify which procedures and packages you want to include or exclude for procedure replication.

You group supported packages and procedures using feature groups. You use the procedure parameter with the `INCLUDE` or `EXCLUDE` keyword to filter features for procedure replication.

In the procedure parameter, `INCLUDE` or `EXCLUDE` specify the beginning of a filtering clause. They specify the procedures to replicate (`INCLUDE`) or filter out (`EXCLUDE`). The filtering clause must consist of the `INCLUDE ALL_SUPPORTED` or `EXCLUDE ALL_SUPPORTED` keyword followed by any valid combination of the other filtering options of the procedure parameter. The `EXCLUDE` filter takes precedence over any `INCLUDE` filters that contain the same criteria.

> **Note:**
>
> When replicating Oracle Streams Advanced Queuing (AQ) procedures, you must use the `RULE` option in your parameter file as follows:
>
> ```
> PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
> ```
>
> or
>
> ```
> PROCEDURE INCLUDE FEATURE AQ, RULE
> ```
>
> Do *not* use `PROCEDURE INCLUDE FEATURE AQ` without the `RULE` option.

Including all system supplied packages at Extract:

1. Connect to Extract in the source database.

   ```
   EXTRACT edba
   ```

   ```
   USERIDALIAS admin_dbA DOMAIN ORADEV
   ```

2. Create a new trail file.

   ```
   EXTTRAIL ea
   ```

3. Enable procedure replication, if not already done.

   ```
   TRANLOGOPTIONS INTEGRATEDPARAMS (ENABLE_PROCEDURAL_REPLICATION Y)
   ```

4. Include filter for procedure replication.

   ```
   PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
   ```

You have successfully included all system supplied packages for procedure replication.

Excluding specific packages at Replicat:

1. Connect to Replicat in the target database.

   ```
   REPLICAT rdba
   ```

   ```
   USERIDALIAS admin_dbBDOMAIN ORADEV
   ```

2. Include filter for procedure replication.

   ```
   PROCEDURE EXCLUDE FEATURE RLS
   ```

You have successfully excluded specific packages for procedure replication.

Handling Procedural Replication Errors

Procedural replication uses `REPERROR` parameter to configure the behavior of Replicat when an procedural error occurs.

By default, Replicat will abend when a procedural replication occurs so using the following steps sets up error handling:

1. Connect to Replicat in the target database.

   ```
   REPLICAT rdba
   ```

   ```
   USERIDALIAS admin_dbBDOMAIN ORADEV
   ```

2. Include filter for procedure replication.

```
PROCEDURE EXCLUDE FEATURE RLS
```

3. Specify error handling parameter, see REPERROR in *Parameters and Functions Reference for Oracle GoldenGate* for other options.

   ```
   REPERROR (PROCEDURE, DISCARD)
   ```

You have successfully handled errors for procedural replication.

Procedural Replication Pragma Options

There are four pragma options for procedures: `AUTO`, `MANUAL`, `UNSUPPORTED`, and `NONE`.

PL/SQL enter and exit markers are logged for procedures with pragmas `AUTO`, `MANUAL`, and `UNSUPPORTED`. The redo logs generated between the enter and exit markers are grouped and discarded.

Following is a list of the packages and procedures that are pragma constructs for replication. Any package or procedure not in this list is not considered a pragma construct for PL/SQL replication and is equivalent to pragma `NONE`.

PL/SQL Procedures with Pragma are UNSUPPORTED

Procedures and packages with the pragma `UNSUPPORTED` stop apply at the point of procedure invocation so that manual intervention can be taken. The following procedures are pragma and `UNSUPPORTED`.

Schema	Package	Procedure	Pragma
SYS	DBMS_REDEFINITION	ABORT_UPDATE	PRAGMA UNSUPPORTED
SYS	DBMS_REDEFINITION	EXECUTE_UPDATE	PRAGMA UNSUPPORTED
XDB	DBMS_XDBZ	ADD_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDBZ	CHANGE_APPLICATION_MEMBERSHIP	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDBZ	DELETE_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDBZ	SET_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	CREATENONCEKEY	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	INSTALLDEFAULTWALLET	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	MOVEXDB_TABLESPACE	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	REBUILDHIERARCHICALINDEX	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTSCHEME	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	CLEARHTTPDIGESTS	PRAGMA UNSUPPORTED with COMMIT

Schema	Package	Procedure	Pragma
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTSCHEME	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMAUTHENTICATION	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMTRUST	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLEDIGESTAUTHENTICATION	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ISGLOBALPORTENABLED	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETDYNAMICGROUPSTORE	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETGLOBALPORTENABLED	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETHTTPCONFIGREALM	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	DROPPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	MODIFYPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	REGISTERPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLSCHEMA	COPYEVOLVE	PRAGMA UNSUPPORTED with COMMIT

PL/SQL Procedures with Pragma `AUTO`

For the procedures and packages with the pragma AUTO, the top-level PL/SQL API is called during apply.

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	ADD_AUTH_TO_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_CMD_RULE_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_FACTOR_LINK	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_INDEX_FUNCTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_NLS_DATA	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_OBJECT_TO_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_OWNER_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_POLICY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_REALM_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_RULE_TO_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_DATAPUMP_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_DDL	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	AUTHORIZE_DIAGNOSTIC_ADMIN	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_MAINTENANCE_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_PREPROCESSOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_PROXY_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_SCHEDULER_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_TTS_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CHANGE_IDENTITY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CHANGE_IDENTITY_VALUE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_DOMAIN_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_IDENTITY_MAP	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_MAC_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_POLICY_LABEL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_AUTH_FROM_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_CMD_RULE_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR_LINK	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR_TYPE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	DELETE_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_IDENTITY_MAP	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_INDEX_FUNCTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_MAC_POLICY_CASCADE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_OBJECT_FROM_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_OWNER_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_POLICY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_POLICY_LABEL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_REALM_CASCADE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_REALM_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE_FROM_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_SYSTEM_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV_DICTIONARY_ACCTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV_PATCH_ADMIN_AUDIT	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_ORADEBUG	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_DOMAIN_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV_DICTIONARY_ACCTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV_PATCH_ADMIN_AUDIT	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_ORADEBUG	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_POLICY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	RENAME_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DATAPUMP_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DDL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DIAGNOSTIC_ADMIN	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_MAINTENANCE_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_PREPROCESSOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_PROXY_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_SCHEDULER_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_TTS_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_MAC_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_POLICY_DESCRIPTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_POLICY_STATE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_REALM_AUTH	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_SYSTEM_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_ADMIN_AUDIT	PRAGMA AUTO
DVSYS	DBMS_MACADM	CREATE_MACOLS_CONTEXTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_MACOLS_CONTEXTS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_EVENTS	AFTER_CREATE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
LBACSYS	LBAC_EVENTS	AFTER_DROP	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_EVENTS	BEFORE_ALTER	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	ADD_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ADD_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ALTER_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ALTER_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	CONFIGURE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	CREATE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	DISABLE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_ALL_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_ALL_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ENABLE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	INSERT_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SAVE_DEFAULT_LABELS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_DEFAULT_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_LEVELS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_ROW_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_USER_LABELS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	STORE_LABEL_LIST	PRAGMA AUTO

Schema	Package	Procedure	Pragma
LBACSYS	LBAC_POLICY_ADMIN	ALTER_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	APPLY_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	APPLY_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	DISABLE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	DISABLE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	ENABLE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	ENABLE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	POLICY_SUBSCRIBE	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	POLICY_UNSUBSCRIBE	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	REMOVE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	REMOVE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL_ENABLED	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL_ENABLED_SQL	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	CREATE_VIEW	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	DROP_VIEW	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	NOAUDIT	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	NOAUDIT_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_GROUP	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_GROUP	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	CREATE_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	ALTER_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	DISABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	DROP_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	ENABLE_POLICY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
LBACSYS	SA_USER_ADMIN	DROP_USER_ACCESS	PRAGMA AUTO with COMMIT
LBACSYS	SA_USER_ADMIN	SET_PROG_PRIVS	PRAGMA AUTO with COMMIT
LBACSYS	SA_USER_ADMIN	SET_USER_PRIVS	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ	AQ$_BACKGROUND_OPER	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_DELETE_DIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_DELETE_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_DELETE_TIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_DIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_TIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_UPDATE_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_UPDATE_HIST_TAB_EX	PRAGMA AUTO
SYS	DBMS_AQ	DEQUEUE_INTERNAL	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD_JMS	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	REGISTRATION_REPLICATION	PRAGMA AUTO
SYS	DBMS_AQADM	ALTER_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	CREATE_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	DISABLE_DB_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	DROP_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	ENABLE_DB_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM	GRANT_TYPE_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_TABLE	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_AQADM_SYS	ALTER_SHARDED_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_SUBSCRIBER_11G	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_EXCEPTION_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_NP_QUEUE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_SHARDED_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_SHARDED_QUEUE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ENABLE_JMS_TYPES_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	GRANT_QUEUE_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	MIGRATE_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PATCH_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PATCH_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PSTUPD_CREATE_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PURGE_QUEUE_TABLE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REMOVE_ORPHMSGS_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REMOVE_SUBSCRIBER_11G_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REVOKE_QUEUE_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	START_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	STOP_QUEUE	PRAGMA AUTO
SYS	DBMS_AQELM	SET_MAILHOST	PRAGMA AUTO
SYS	DBMS_AQELM	SET_MAILPORT	PRAGMA AUTO
SYS	DBMS_AQELM	SET_PROXY	PRAGMA AUTO
SYS	DBMS_AQELM	SET_SENDFROM	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	BUMP_TID_SEQUENCE	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	CLEANUP_SCHEMA_IMPORT	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_CMT_TIME_TABLE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_DEQUEUELOG_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_EXP_ENTRY	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_HISTORY_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_INDEX_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QTAB_EXPDEP	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_META	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_SEQ	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_SIGNATURE_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_SUBSCRIBER_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_TIMEMGR_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_REBUILD_IDX	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_SHARDED_Q	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_WORK	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_WORK_REMAINING	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_MOUNT	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_MOUNTP	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_STORE	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	MOUNTSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	REGISTERSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	UNMOUNTSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	UNREGISTERSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS	NORMALIZEFS	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_DBFS_CONTENT_ADMIN	REORGANIZEFS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_CONTENT_ADMIN	SHRINKFS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	CREATEFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	DELETE_ORPHANS_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	DROPFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_ATTRV	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_FS	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_GRANTS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_SEQ	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_SNAP	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_TABP	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_TAB_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_VOL	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	INITFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	PARTITION_SEQUENCE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	RECACHE_SEQUENCE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	REGISTERFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	SETFSPROPERTIES_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	UNREGISTERFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DDL	SET_TRIGGER_FIRING_PROPERTY	PRAGMA AUTO with COMMIT
SYS	DBMS_DDL	SET_TRIGGER_FIRING_PROPERTY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	DISABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ENABLE_POLICY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_GOLDENGATE_ADM_INT_I	ADD_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_AUTO_CDR_DELTA_RES_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ALTER_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ALTER_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_DELTA_RES_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_COLUMN	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_COLUMN_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_END	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_START	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_DISABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_ENABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_ROLLING	DESTROY_META	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	INSERT_DGLRDDIR	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	INSERT_DGLRDEVT	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	SET_UPGRADE_FLAGS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPDATE_DGLRDINS_PROGRESS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDCON	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDDAT	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDINS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDPAR	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDSTA	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDSTS	PRAGMA AUTO
SYS	DBMS_ISCHED	CREATE_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_ISCHED	EXEC_JOB_RUN_LSA	PRAGMA AUTO
SYS	DBMS_ISCHED	SET_AGENT_REGISTRATION_PASS	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	SUBID_REPLICATE	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	ADD_DURABLE_SUB	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	ALTER_SUBSCRIBER_12G	PRAGMA AUTO
SYS	DBMS_PRVTAQIS	REMOVE_SUBSCRIBER_12G	PRAGMA AUTO
SYS	DBMS_REDACT	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	ALTER_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	APPLY_POLICY_EXPR_TO_COL	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	CREATE_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DISABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DROP_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	ENABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	FPM_MASK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	FPM_UNMASK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	UPDATE_FULL_REDACTION_VALUES	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	UPDATE_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	ABORT_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	ABORT_ROLLBACK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	COPY_TABLE_DEPENDENTS	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	FINISH_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	REGISTER_DEPENDENT_OBJECT	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	ROLLBACK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	SET_PARAM	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	START_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	SYNC_INTERIM_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	UNREGISTER_DEPENDENT_OBJECT	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_RLS_INT	ADD_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ADD_POLICY_CONTEXT	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ALTER_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ALTER_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	CREATE_POLICY_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DELETE_POLICY_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DISABLE_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_POLICY_CONTEXT	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ENABLE_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ENABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	REFRESH_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	REFRESH_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RULEADM_INTERNAL	ADD_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	ALTER_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	ALTER_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_RULE_SET	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_RULE_SET	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	REMOVE_RULE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	GRANT_OBJECT_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	REVOKE_OBJECT_PRIVILEGE	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_RULE_ADM	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_SCHEDULER	ADD_EVENT_QUEUE_SUBSCRIBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_JOB_EMAIL_NOTIFICATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_TO_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_WINDOW_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ANALYZE_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	AUTO_PURGE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CHECK_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	COPY_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_DATABASE_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_EVENT_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_FILE_WATCHER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOBS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOBS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB_CLASS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_PROGRAM	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_RESOURCE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW_GROUP	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	DEFINE_ANYDATA_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_EVENT_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_EVENT_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_RULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_METADATA_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DELETE_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DISABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DISABLE1_CALENDAR_CHECK	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_AGENT_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN_RULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_DATABASE_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_FILE_WATCHER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_JOB_CLASS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_RESOURCE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_WINDOW_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ENABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	END_DETACHED_JOB_RUN	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	EVALUATE_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_AGENT_INFO	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_SCHEDULER_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PURGE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_EVENT_QUEUE_SUBSCRIBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_FROM_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_JOB_EMAIL_NOTIFICATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_WINDOW_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RESET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RESET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RUN_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RUN_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE_NULL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ANYDATA_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ANYDATA_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ATTRIBUTES	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_RESOURCE_CONSTRAINT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_SCHEDULER_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SHOW_ERRORS	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	CLEAR_SQL_TRANSLATION_ERROR	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	CREATE_PROFILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DEREGISTER_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DEREGISTER_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DROP_PROFILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	ENABLE_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	ENABLE_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	REGISTER_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	REGISTER_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_ERROR_TRANSLATION_COMMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_SQL_TRANSLATION_COMMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_SQL_TRANSLATION_MODULE	PRAGMA AUTO with COMMIT
SYS	DBMS_XDS	ALTER_STATIC_ACL_REFRESH	PRAGMA AUTO
SYS	DBMS_XDS	DISABLE_OLAP_POLICY	PRAGMA AUTO
SYS	DBMS_XDS	DISABLE_XDS	PRAGMA AUTO
SYS	DBMS_XDS	DROP_OLAP_POLICY	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_XDS	DROP_XDS	PRAGMA AUTO
SYS	DBMS_XDS	ENABLE_OLAP_POLICY	PRAGMA AUTO
SYS	DBMS_XDS	ENABLE_XDS	PRAGMA AUTO
SYS	DBMS_XDS	PURGE_ACL_REFRESH_HISTORY	PRAGMA AUTO
SYS	DBMS_XDS	SCHEDULE_STATIC_ACL_REFRESH	PRAGMA AUTO
SYS	DBMS_XDS	SET_TRACE_LEVEL	PRAGMA AUTO
SYS	DBMS_XDS	XDS$REFRESH_STATIC_ACL	PRAGMA AUTO
SYS	LOGSTDBY_INTERNAL	EDS_EVOLVE_TABLE_I	PRAGMA AUTO with COMMIT
SYS	LOGSTDBY_INTERNAL	EDS_REMOVE_TABLE_I	PRAGMA AUTO with COMMIT
SYS	XS_ACL	ADD_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_ACL	ADD_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_ACL	APPEND_ACES	PRAGMA AUTO
SYS	XS_ACL	APPEND_ACES	PRAGMA AUTO
SYS	XS_ACL	CREATE_ACL	PRAGMA AUTO
SYS	XS_ACL	DELETE_ACL	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACES	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_ACL	SET_PARENT_ACL	PRAGMA AUTO
SYS	XS_ACL	SET_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_ADMIN_UTIL	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	XS_ADMIN_UTIL	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPEND_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPEND_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPLY_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_POLICY	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_DATA_SECURITY	DELETE_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DELETE_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DISABLE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ENABLE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_NAMESPACE	ADD_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_TEMPLATE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DELETE_TEMPLATE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_HANDLER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_TO_DBUSER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_DYNAMIC_ROLE	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_ROLE	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	DELETE_PRINCIPAL	PRAGMA AUTO
SYS	XS_PRINCIPAL	ENABLE_BY_DEFAULT	PRAGMA AUTO
SYS	XS_PRINCIPAL	ENABLE_ROLES_BY_DEFAULT	PRAGMA AUTO
SYS	XS_PRINCIPAL	GRANT_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	GRANT_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	REMOVE_PROXY_FROM_DBUSER	PRAGMA AUTO
SYS	XS_PRINCIPAL	REMOVE_PROXY_USERS	PRAGMA AUTO
SYS	XS_PRINCIPAL	REMOVE_PROXY_USERS	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_ACL	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_PRINCIPAL	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_DYNAMIC_ROLE_DURATION	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_DYNAMIC_ROLE_SCOPE	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_EFFECTIVE_DATES	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_GUID	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_PROFILE	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_USER_SCHEMA	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_USER_STATUS	PRAGMA AUTO
SYS	XS_PRINCIPAL_INT	SET_VERIFIER_HELPER	PRAGMA AUTO
SYS	XS_ROLESET	ADD_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	ADD_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	CREATE_ROLESET	PRAGMA AUTO
SYS	XS_ROLESET	DELETE_ROLESET	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	CREATE_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	DELETE_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_SECURITY_CLASS	SET_DESCRIPTION	PRAGMA AUTO
SYS	DBMS_RESCONFIG	ADDREPOSITORYRESCONFIG	PRAGMA AUTO with COMMIT
SYS	DBMS_RESCONFIG	ADDRESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	APPENDRESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	DELETEREPOSITORYRESCONFIG	PRAGMA AUTO with COMMIT
SYS	DBMS_RESCONFIG	DELETERESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	DELETERESCONFIG	PRAGMA AUTO
SYS	DBMS_XDBZ	DISABLE_HIERARCHY	PRAGMA AUTO with COMMIT
SYS	DBMS_XDBZ	ENABLE_HIERARCHY	PRAGMA AUTO with COMMIT
SYS	DBMS_XDB_VERSION	CHECKIN_INT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	CHECKOUT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	MAKEVERSIONED_INT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	UNCHECKOUT_INT	PRAGMA AUTO
SYS	DBMS_XLSB	DELETERESOURCE	PRAGMA AUTO
SYS	DBMS_XLSB	DELNAMELOCKS	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCE	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCENXOB	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCENXOBCLOB	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCEREF	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTTOHTABLE	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTTOUSERHTAB	PRAGMA AUTO
SYS	DBMS_XLSB	LINKRESOURCE	PRAGMA AUTO
SYS	DBMS_XLSB	SAVEACL	PRAGMA AUTO
SYS	DBMS_XLSB	SETREFCOUNT	PRAGMA AUTO
SYS	DBMS_XLSB	TOUCHOID	PRAGMA AUTO

PL/SQL Procedures with Pragma `MANUAL`

For the procedures and packages pragma-ed `MANUAL`, the top-level PL/SQL API is not called.

> **Note:**
>
> From Oracle GoldenGate 23c onward, you do not need to invoke `DBMS_GOLDENGATE_AUTH` package.

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	AQ$_BACKGROUND_OPER_PAS	PRAGMA MANUAL

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	DEQUEUE_INTERNAL_PAS	PRAGMA MANUAL
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED_PAS	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_PROPAGATION_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_SUBSCRIBER_11G_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DISABLE_PROP_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DROP_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DROP_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ENABLE_PROP_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	GRANT_QUEUE_PRIVILEGE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	MIGRATE_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	PURGE_QUEUE_TABLE	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	RECOVER_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REMOVE_ORPHMSGS_NR	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REMOVE_SUBSCRIBER_11G	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REVOKE_QUEUE_PRIVILEGE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	SCHEDULE_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	START_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	STOP_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	UNSCHEDULE_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_TABLE_START	PRAGMA MANUAL with COMMIT
SYS	DBMS_PRVTAQIS	SUBID_REPLICATE_INT	PRAGMA MANUAL
SYS	LOGSTDBY_INTERNAL	EDS_ADD_TABLE_I	PRAGMA MANUAL with COMMIT
SYS	XS_ADMIN_UTIL	DROP_SCHEMA_OBJECTS	PRAGMA MANUAL

Schema	Package	Procedure	Pragma
XDB	DBMS_XDBZ0	DISABLE_HIERARCHY_INTERNAL	PRAGMA MANUAL
XDB	DBMS_XDBZ0	ENABLE_HIERARCHY_INTERNAL	PRAGMA MANUAL

PL/SQL Procedures with Pragma `NONE`

For the procedures and packages pragma-ed `NONE`, PL/SQL markers are not generated and no grouping is performed. Redo logs generated by these procedures are applied or skipped based on table level replication semantics.

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	DISABLE_EVENT	PRAGMA NONE
DVSYS	DBMS_MACADM	DV_SANITY_CHECK	PRAGMA NONE
DVSYS	DBMS_MACADM	ENABLE_EVENT	PRAGMA NONE
DVSYS	DBMS_MACADM	SET_PRESERVE_CASE	PRAGMA NONE
DVSYS	DBMS_MACADM	INIT_SESSION	PRAGMA NONE
DVSYS	DBMS_MACADM	UPDATE_POLICY_LABEL_CONTEXT	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	LABEL_AUDIT_RAISE	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	RESTORE_DEFAULT_LABELS	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	SET_POLICY_LABEL_CONTEXT	PRAGMA NONE
DVSYS	DBMS_MACOUT	DISABLE	PRAGMA NONE
DVSYS	DBMS_MACOUT	ENABLE	PRAGMA NONE
DVSYS	DBMS_MACOUT	PL	PRAGMA NONE
DVSYS	DBMS_MACOUT	PUT_LINE	PRAGMA NONE
DVSYS	DBMS_MACOUT	SET_FACTOR	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	SET_ROLE	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE_TR	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE_WR	PRAGMA NONE
DVSYS	DBMS_MACUTL	CHECK_DVSYS_DML_ALLOWED	PRAGMA NONE
DVSYS	DBMS_MACUTL	RAISE_ERROR	PRAGMA NONE
DVSYS	DBMS_MACUTL	RAISE_UNAUTHORIZED_OPERATION	PRAGMA NONE
DVSYS	EVENT	SET	PRAGMA NONE
DVSYS	EVENT	SETDEFAULT	PRAGMA NONE
DVSYS	EVENT	SET_C	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	BIND_AGENT	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	LISTEN	PRAGMA NONE
SYS	DBMS_AQ	LISTEN	PRAGMA NONE
SYS	DBMS_AQ	POST	PRAGMA NONE
SYS	DBMS_AQ	REGISTER	PRAGMA NONE
SYS	DBMS_AQ	UNBIND_AGENT	PRAGMA NONE
SYS	DBMS_AQ	UNREGISTER	PRAGMA NONE
SYS	DBMS_AQADM	ADD_ALIAS_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_CONNECTION_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_CONNECTION_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SHARDED_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_EXCEPTION_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_NP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_SHARDED_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	DEL_ALIAS_FROM_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	DEL_CONNECTION_FROM_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	DISABLE_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_SHARDED_QUEUE	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQADM	ENABLE_JMS_TYPES	PRAGMA NONE
SYS	DBMS_AQADM	ENABLE_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	GET_PROP_SEQNO	PRAGMA NONE
SYS	DBMS_AQADM	GET_REPLAY_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_TYPE_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_TYPE_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_WATERMARK	PRAGMA NONE
SYS	DBMS_AQADM	GRANT_QUEUE_PRIVILEGE	PRAGMA NONE
SYS	DBMS_AQADM	MIGRATE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_RECEIVER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_RECEIVER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_SENDER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_SENDER	PRAGMA NONE
SYS	DBMS_AQADM	PURGE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	RECOVER_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	REMOVE_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	RESET_REPLAY_INFO	PRAGMA NONE
SYS	DBMS_AQADM	REVOKE_QUEUE_PRIVILEGE	PRAGMA NONE
SYS	DBMS_AQADM	SCHEDULE_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	SET_WATERMARK	PRAGMA NONE
SYS	DBMS_AQADM	START_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	START_TIME_MANAGER	PRAGMA NONE
SYS	DBMS_AQADM	STOP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	STOP_TIME_MANAGER	PRAGMA NONE
SYS	DBMS_AQADM	UNSCHEDULE_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES_GET_NRP	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES_NO_QUEUE	PRAGMA NONE
SYS	DBMS_AQELM	GET_MAILHOST	PRAGMA NONE
SYS	DBMS_AQELM	GET_MAILPORT	PRAGMA NONE
SYS	DBMS_AQELM	GET_PROXY	PRAGMA NONE
SYS	DBMS_AQELM	GET_SENDFROM	PRAGMA NONE
SYS	DBMS_AQELM	GET_TXTIMEOUT	PRAGMA NONE
SYS	DBMS_AQELM	HTTP_SEND	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQELM	SEND_EMAIL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ_NO_RECPL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ_NO_RECPL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_BYTES_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_MAP_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_OBJECT_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_STREAM_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_TEXT_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_LISTEN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_QUEUE_SUBSCRIBERS	PRAGMA NONE
SYS	DBMS_AQIN	SET_DEQ_SORT	PRAGMA NONE
SYS	DBMS_AQIN	SET_MULTI_RETRY	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_GET_PROP_STAT	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_GET_TRANS_TYPE	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_REGISTER	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_UNREGISTER	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_UPDATE_PROP_STAT_QNAME	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_DBSESSION_GUID	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_GLOBAL_AQCLNTDB_CTX_CLNT	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_GLOBAL_AQCLNTDB_CTX_DB	PRAGMA NONE
SYS	DBMS_AQJMS	GET_DB_USERNAME_FOR_AGENT	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQJMS	SET_DBSESSION_GUID	PRAGMA NONE
SYS	DBMS_AQJMS	SET_GLOBAL_AQCLNTDB_CTX	PRAGMA NONE
SYS	DBMS_AQJMS	SUBSCRIBER_EXISTS	PRAGMA NONE
SYS	DBMS_AQJMS	SUBSCRIBER_EXISTS	PRAGMA NONE
SYS	DBMS_ISCHED	GET_AGENT_PASS_VERIFIER	PRAGMA NONE
SYS	DBMS_ISCHED	OBFUSCATE_CREDENTIAL_PASSWORD	PRAGMA NONE
SYS	DBMS_REDEFINITION	CAN_REDEF_TABLE	PRAGMA NONE
SYS	DBMS_REDEFINITION	REDEF_TABLE	PRAGMA NONE
SYS	DBMS_SCHEDULER	CHECK_SYS_PRIVS	PRAGMA NONE
SYS	DBMS_SCHEDULER	CLOSE_WINDOW	PRAGMA NONE
SYS	DBMS_SCHEDULER	CREATE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	CREATE_CREDENTIAL	PRAGMA NONE
SYS	DBMS_SCHEDULER	EVALUATE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	FILE_WATCH_FILTER	PRAGMA NONE
SYS	DBMS_SCHEDULER	GENERATE_EVENT_LIST	PRAGMA NONE
SYS	DBMS_SCHEDULER	GENERATE_JOB_NAME	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_AGENT_VERSION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_CHAIN_RULE_ACTION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_CHAIN_RULE_CONDITION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_DEFAULT_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_JOB_STEP_CF	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_SYS_TIME_ZONE_NAME	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_VARCHAR2_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_VARCHAR2_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	IS_SCHEDULER_CREATED_AGENT	PRAGMA NONE
SYS	DBMS_SCHEDULER	OPEN_WINDOW	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_NAME	PRAGMA NONE
SYS	DBMS_SCHEDULER	RUN_JOB	PRAGMA NONE
SYS	DBMS_SCHEDULER	SET_AGENT_REGISTRATION_PASS	PRAGMA NONE
SYS	DBMS_SCHEDULER	STIME	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	STOP_JOB	PRAGMA NONE
SYS	DBMS_SCHEDULER	SUBMIT_REMOTE_EXTERNAL_JOB	PRAGMA NONE
SYS	XS_PRINCIPAL	SET_PASSWORD	PRAGMA NONE
SYS	XS_PRINCIPAL	SET_VERIFIER	PRAGMA NONE

Listing the Procedures Supported for Oracle GoldenGate Procedural Replication

The `DBA_GG_SUPPORTED_PROCEDURES` view displays information about the supported packages for Oracle GoldenGate procedural replication.

When a procedure is supported and Oracle GoldenGate procedural replication is on, calls to the procedure are replicated, unless the procedure is excluded specifically.

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Query the `DBA_GG_SUPPORTED_PROCEDURES` view.

Example 8-2 Displaying Information About the Packages Supported for Oracle GoldenGate Procedural Replication

This query displays the following information about the packages:

- The owner of each package
- The name of each package
- The name of each procedure
- The minimum database release from which the procedure is supported
- Whether there is an exclusion rule that prevents the procedure from being replicated for some database objects

```
COLUMN OWNER FORMAT A10
COLUMN PACKAGE_NAME FORMAT A15
COLUMN PROCEDURE_NAME FORMAT A15
COLUMN MIN_DB_VERSION FORMAT A14
COLUMN EXCLUSION_RULE_EXISTS FORMAT A14

SELECT OWNER,
       PACKAGE_NAME,
       PROCEDURE_NAME,
       MIN_DB_VERSION,
       EXCLUSION_RULE_EXISTS
  FROM DBA_GG_SUPPORTED_PROCEDURES;
```

Your output looks similar to the following:

```
OWNER      PACKAGE_NAME    PROCEDURE_NAME  MIN_DB_VERSION EXCLUSION_RULE
---------- --------------- --------------- -------------- --------------
XDB        DBMS_XDB_CONFIG ADDTRUSTMAPPING 12.2           NO
CTXSYS     CTX_DDL         ALTER_INDEX     12.2           NO
SYS        DBMS_FGA        DROP_POLICY     12.2           NO
SYS         XS_ACL          DELETE_ACL      12.2           NO
.
.
.
```

Monitoring Oracle GoldenGate Procedural Replication

A set of data dictionary views enable you to monitor Oracle GoldenGate procedural replication.

You can use the following views to monitor Oracle GoldenGate procedural replication:

View	Description
`DBA_GG_SUPPORTED_PACKAGES`	Provides details about supported packages for Oracle GoldenGate procedural replication. When a package is supported and Oracle GoldenGate procedural replication is on, calls to subprograms in the package are replicated.
`DBA_GG_SUPPORTED_PROCEDURES`	Provides details about the procedures that are supported for Oracle GoldenGate procedural replication.
`DBA_GG_PROC_OBJECT_EXCLUSION`	Provides details about all database objects that are on the exclusion list for Oracle GoldenGate procedural replication. A database object is added to the exclusion list using the `INSERT_PROCREP_EXCLUSION_OBJ` procedure in the `DBMS_GOLDENGATE_ADM` package. When a database object is on the exclusion list, execution of a subprogram n the package is not replicated if the subprogram operates on the excluded object.

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, or `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Query the views related to Oracle GoldenGate procedural replication.

Mapping and Manipulating Data

This chapter describes how you can integrate data between source and target tables.

Guidelines for Using Self-describing Trails

Self-describing trail files are the default if the trail file format is 12.2 or higher, if you are not using `SOURCEDEFS OVERRIDE` or `TARGETDEFS OVERRIDE`. Oracle recommends that you use self-describing trail files. You should only use `SOURCEDEFS OVERRIDE` and `TARGETDEFS OVERRIDE` for backward compatibility requirements.

The following are the guidelines for using self-describing trails:

- If using the self-describing trails, then the column names on the source are mapped to the column names in the target table. Order of columns doesn't matter and if column names are different, then they need to be explicitly mapped using `COLMAP`.

- If the source Oracle GoldenGate release is 12.1 or earlier, then you need to use `SOURCEDEFS OVERRIDE` or `TARGETDEFS OVERRIDE`. See `SOURCEDEFS OVERRIDE` and `TARGETDEFS OVERRIDE` in the *Parameters and Functions Reference for Oracle GoldenGate*.

Parameters that Control Mapping and Data Integration

All data selection, mapping, and manipulation that Oracle GoldenGate performs is accomplished by using one or more options of the `TABLE` and `MAP` parameters.

- Use `TABLE` in the Extract parameter file.
- Use `MAP` in the Replicat parameter file.

`TABLE` and `MAP` specify the database objects that are affected by the other parameters in the parameter file. See Specifying Object Names in Oracle GoldenGate Input for instructions for specifying object names in these parameters.

Mapping between Dissimilar Databases

Mapping and conversion between tables that have different data structures requires either a source-definitions file, a target-definitions file, or in some cases both. Mapping between dissimilar databases is controlled by the self-describing trails, and mapping is done by column name, regardless of the data type for the source or target column.

If you don't want automatic mapping based on the self-describing trails or want backward compatibility then you can use `SOURCEDEFS` or `TARGETDEFS`.

Deciding Where Data Mapping and Conversion Will Take Place

If the configuration you are planning involves a large amount of column mapping or data conversion, observe the following guidelines to determine which process or processes will perform these functions.

Mapping and Conversion on Windows and UNIX Systems

When Oracle GoldenGate is operating only on Windows-based and UNIX-based systems, column mapping and conversion can be performed in the Extract process, or in the Replicat process. To prevent the added overhead of this processing on the Extract process, you can configure the mapping and conversion to be performed on the Replicat process or on an intermediary system.

In the case where there are multiple sources and one target, it might be more efficient to perform the mapping and conversion on the source.

Mapping and Conversion on NonStop Systems

If you are mapping or converting data from a Windows or UNIX system to a NonStop Enscribe target, the mapping or conversion must be performed on the Windows or UNIX source system. Replicat for NonStop cannot convert three-part or two-part SQL table names and data types to the three-part file names that are used for the Enscribe platform. Extract can format the trail data with Enscribe names and target data types.

Globalization Considerations when Mapping Data

When planning to map and convert data between databases and platforms, take into consideration what is supported or not supported by Oracle GoldenGate in terms of globalization.

Conversion between Character Sets

Oracle GoldenGate converts between source and target character sets if they are different, so that object names and column data are compared, mapped, and manipulated properly from one database to another. See Supported Character Sets, for a list of supported character sets.

To ensure accurate character representation from one database to another, the following must be true:

- The character set of the target database must be a superset or equivalent of the character set of the source database. *Equivalent* means not equal, but having the same set of characters. For example, Shift-JIS and EUC-JP technically are not completely equal, but have the same characters in most cases.
- If your client applications use different character sets, the database character set must also be a superset or equivalent of the character sets of the client applications.
- In many databases, including Oracle, it is possible to force a character into a database that is not part of the Character Set. Oracle GoldenGate considers this as an invalid value, and may not map this character correctly when replicating data. For these types of situations you can use the `REPLACEBADCHAR` parameter as described in the *Parameters and Functions Reference for Oracle GoldenGate*.

In this configuration, every character is represented when converting from a client or source character set to the local database character set.

A Replicat process can support conversion from one source character set to one target character set.

Database Object Names

Oracle GoldenGate processes catalog, schema, table and column names in their native language as determined by the character set encoding of the source and target databases. This support preserves single-byte and multibyte names, symbols, accent characters, and case-sensitivity with locale taken into account where available, at all levels of the database hierarchy.

Column Data

Oracle GoldenGate supports the conversion of column data between character sets when the data is contained in the following column types:

- Character-type columns: `CHAR/VARCHAR/CLOB` to `CHAR/VARCHAR/CLOB` of another character set; and `CHAR/VARCHAR/CLOB` to and from `NCHAR/NVARCHAR/NCLOB`.
- Columns that contain string-based numbers and date-time data. Conversions of these columns is performed between z/OS EBCDIC and non-z/OS ASCII data. Conversion is not performed between ASCII and ASCII versions of this data, nor between EBCDIC and EBCDIC versions, because the data are compatible in these cases.

> **Note:**
>
> Oracle GoldenGate supports timestamp data from `0001-01-03 00:00:00` to `9999-12-31 23:59:59`. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. A value of zero month, zero day field, or an all zero date value isn't supported. For example, values such as 0000-00-00 00:00:00, or any date value that includes a zero month or zero day field isn't supported.

Character-set conversion for column data is limited to a direct mapping of a source column and a target column in the `COLMAP` or `USEDEFAULTS` clauses of the Replicat `MAP` parameter. A direct mapping is a name-to-name mapping without the use of a stored procedure or column-conversion function. Replicat performs the character-set conversion. No conversion is performed by Extract or a data pump.

Preservation of Locale

Oracle GoldenGate takes the locale of the database into account when comparing case-insensitive object names. See Supported Locales for a list of supported locales.

Support for Escape Sequences

Oracle GoldenGate supports the use of an escape sequence to represent a string column, literal text, or object name in the parameter file. You can use an escape sequence if the operating system does not support the required character, such as a control character, or for any other purpose that requires a character that cannot be used in a parameter file.

An escape sequence can be used anywhere in the parameter file, but is particularly useful in the following elements within a `TABLE` or `MAP` statement:

- An object name
- `WHERE` clause
- `COLMAP` clause to assign a Unicode character to a Unicode column, or to assign a native-encoded character to a column.
- Oracle GoldenGate column conversion functions within a `COLMAP` clause.

Oracle GoldenGate supports the following types of escape sequence:

- `\uFFFF` Unicode escape sequence. Any `UNICODE` code point can be used except surrogate pairs.
- `\377` Octal escape sequence
- `\xFF` Hexadecimal escape sequence

The following rules apply:

- If used for mapping of an object name in `TABLE` or `MAP`, no restriction apply. For example, the following `TABLE` specification is valid:

  ```
  TABLE schema."\u3000ABC";
  ```

- If used with a column-mapping function, any code point can be used, but only for an `NCHAR/NVARCHAR` column. For an `CHAR/VARCHAR` column, the code point is limited to the equivalent of 7-bit ASCII.
- The source and target data types must be identical (for example, `NCHAR` to `NCHAR`).

- Begin each escape sequence with a reverse solidus (code point `U+005C`), followed by the character code point. (A solidus is more commonly known as the backslash symbol.) Use the escape sequence, instead of the actual character, within your input string in the parameter statement or column-conversion function.

> **Note:**
>
> To specify an actual backslash in the parameter file, specify a double backslash. For example, the following finds a backslash in `COL1: @STRFIND (COL1, '\\' ).`

To Use the \uFFFF Unicode Escape Sequence

- The `\uFFFF` Unicode escape sequence must begin with a lowercase `u`, followed by exactly four hexadecimal digits.
- Supported ranges are as follows:
  - `0` to `9` (`U+0030` to `U+0039`)
  - `A` to `F` (`U+0041` to `U+0046`)
  - `a` to `f` (`U+0061` to `U+0066`)

`\u20ac` is the Unicode escape sequence for the Euro currency sign.

> **Note:**
>
> For reliable cross-platform support, use the Unicode escape sequence. Octal and hexadecimal escape sequences are not standardized on different operating systems.

To Use the \377 Octal Escape Sequence

- Must contain exactly three octal digits.
- Supported ranges:
  - Range for first digit is `0` to `3` (`U+0030` to `U+0033`)
  - Range for second and third digits is `0` to `7` (`U+0030` to `U+0037`)

    `\200` is the octal escape sequence for the Euro currency sign on Microsoft Windows

To Use the \xFF Hexadecimal Escape Eequence

- Must begin with a lowercase `x` followed by exactly two hexadecimal digits.
- Supported ranges:
  - `0` to `9` (`U+0030` to `U+0039`)
  - `A` to `F` (`U+0041` to `U+0046`)
  - `a` to `f` (`U+0061` to `U+0066`)

`\x80` is the hexadecimal escape sequence for the Euro currency sign on Microsoft Windows 1252 Latin1 code page.

Mapping Columns Using TABLE and MAP

Oracle GoldenGate provides for column mapping at the table level and at the global level. Default column mapping is also provided in the absence of explicit column mapping rules.

This section contains the following guidelines for mapping columns:

Supporting Case and Special Characters in Column Names

By default, Oracle GoldenGate follows SQL-92 rules for specifying column names and literals. In Oracle GoldenGate parameter files, conversion functions, user exits, and commands, case-sensitive column names must be enclosed within double quotes if double quotes are required by the database to enforce case-sensitivity. For other case-sensitive databases that do not require quotes, case-sensitive column names must be specified as they are stored in the database. Literals must be enclosed within single quotes. See Differentiating Case-Sensitive Column Names from Literals for more information.

Configuring Table-level Column Mapping with COLMAP

If you are using self-describing trails then any column on the source object is mapped to the same column name on the target object. You only need to manage column names that are different between source and target or if you need to transform a column.

However, if not using self-describing trails then the default mapping is done by column order and not the column name. So column 1 on the source will be mapped to column 1 on the target, column 2 to column 2 and so on.

Use the `COLMAP` option of the `MAP` and `TABLE` parameters to:

- map individual source columns to target columns that have different names.
- specify default column mapping when an explicit column mapping is not needed.
- Provide instructions for selecting, mapping, translating, and moving data from a source column into a target column.

Using USEDEFAULTS to Enable Default Column Mapping

You can use the `USEDEFAULTS` option of `COLMAP` to specify automatic default column mapping for any corresponding source and target columns that have identical names. `USEDEFAULTS` can save you time by eliminating the need to map every target column explicitly.

Default mapping causes Oracle GoldenGate to map those columns and, if required, translate the data types based on the data-definitions file. Do not specify default mapping for columns that are mapped already with an explicit mapping statement.

The following example of a column mapping illustrates the use of both default and explicit column mapping for a source table `ACCTBL` and a target table `ACCTTAB`. Most columns are the same in both tables, except for the following differences:

- The source table has a `CUST_NAME` column, whereas the target table has a `NAME` column.
- A ten-digit `PHONE_NO` column in the source table corresponds to separate `AREA_CODE`, `PHONE_PREFIX`, and `PHONE_NUMBER` columns in the target table.
- Separate `YY, MM`, and `DD` columns in the source table correspond to a single `TRANSACTION_DATE` column in the target table.

To address those differences, `USEDEFAULTS` is used to map the similar columns automatically, while explicit mapping and conversion functions are used for dissimilar columns.

The following sample shows the column mapping using the `COLMAP` option of the `MAP` and `TABLE` parameters. It describes the mapping of the source table `ACCTBL` to the target table `ACCTTAB`.

```
MAP SALES.ACCTBL, TARGET SALES.ACCTTAB,
                COLMAP (   USEDEFAULTS,
                           NAME = CUST_NAME,
                           TRANSACTION_DATE = @DATE ('YYYY-MM-DD', 'YY',YEAR,
'MM', MONTH, 'DD', DAY),
                           AREA_CODE = @STREXT (PHONE_NO, 1, 3),
                           PHONE_PREFIX = @STREXT (PHONE_NO, 4, 6),
                           PHONE_NUMBER = @STREXT (PHONE_NO, 7, 10)
)
;
```

Table 8-2 Sample Column Mapping

Parameter statement	Description
`COLMAP`	Begins the `COLMAP` statement.
`USEDEFAULTS,`	Maps source columns as-is when the target column names are identical.
`NAME = CUST_NAME,`	Maps the source column `CUST_NAME` to the target column `NAME`.
`TRANSACTION_DATE = @DATE ('YYYY-MM-DD', 'YY', YEAR, 'MM', MONTH, 'DD', DAY),`	Converts the transaction date from the source date columns to the target column `TRANSACTION_DATE` by using the `@DATE` column conversion function.

`AREA_CODE = @STREXT (PHONE_NO, 1, 3),`
`PHONE_PREFIX = @STREXT (PHONE_NO, 4, 6),`
`PHONE_NUMBER = @STREXT (PHONE_NO, 7, 10))`
`;`
	Converts the source column `PHONE_NO` into the separate target columns of `AREA_CODE`, `PHONE_PREFIX`, and `PHONE_NUMBER` by using the `@STREXT` column conversion function.

See Understanding Default Column Mapping for more information about the rules followed by Oracle GoldenGate for default column mapping.

Specifying the Columns to be Mapped in the COLMAP Clause

The `COLMAP` syntax is the following:

```
COLMAP ([USEDEFAULTS, ] target_column = source_expression)
```

In this syntax, *`target_column`* is the name of the target column and *`source_expression`*. Some examples of `source_expressions` are:

- The name of a source column, such as `ORD_DATE`.
- Numeric constant, such as `123`.
- String constant enclosed within single quotes, such as `'ABCD'`.
- An expression using an Oracle GoldenGate column-conversion function. Within a `COLMAP` statement, you can use any of the Oracle GoldenGate column-conversion functions to transform data for the mapped columns, for example:

  ```
  @STREXT (COL1, 1, 3)
  ```

- Here's an example of using `BEFORE column_name`: `BEFORE ORD_DATE`
- Here's an example of using `AFTER column_name` : `AFTER ORD_DATE`. This is the default option if a column name is listed.

If the column mapping involves case-sensitive columns from different database types, specify each column as it is stored in the database.

- If the database requires double quotes to enforce case-sensitivity, specify the case-sensitive column name within double quotes.
- If the database is case-sensitive without requiring double quotes, specify the column name as it is stored in the database.

The following shows a mapping between a target column in an Oracle database and a source column in a case-sensitive SQL Server database.

```
COLMAP ("ColA" = ColA)
```

See Specifying Object Names in Oracle GoldenGate Input for more information about specifying names to Oracle GoldenGate.

See Globalization Considerations when Mapping Data for globalization considerations when mapping source and target columns in databases that have different character sets and locales.

Avoid using `COLMAP` to map a value to a key column (which causes the operation to become a primary key update), The `WHERE` clause that Oracle GoldenGate uses to locate the target row will not use the correct before image of the key column. Instead, it will use the after image. This will cause errors if you are using any functions based on that key column, such as a `SQLEXEC` statement.

Column Mapping Limitations

Here are the column mapping limitations:

- `LOB` columns cannot be used in `FILTER`, `WHERE` columns, or as a `source_expression` in a `COLMAP` statement. `LOB` columns are `BLOB`, `CLOB`, `NCLOB`, `XMLType`, User-Defined Data Types, Nested Tables, `VARRAYs` and other special data types.
- If the source column contains more than 4000 bytes, it cannot be used in transformation routines, as the value is stored in the trail as an `LOB` record. For example a `VARCHAR2(4000 CHAR)` in Oracle and the Japanese character set is stored as 3 bytes for each character. This implies that the column could be 12000 bytes long and Oracle GoldenGate would store this value as an `LOB` field.

- The full SQL statement that Oracle GoldenGate would execute would exceed 4MB in size. For example, if you have a table with thousands of `VARCHAR2(4000)` columns and you want to put 4000 bytes in each one, this could cause the total SQL statement that Oracle GoldenGate is going to execute to exceed the maximum size of 4MB.

Configuring Global Column Mapping with COLMATCH

Use the `COLMATCH` parameter to create global rules for column mapping. With `COLMATCH`, you can map between similarly structured tables that have different column names for the same sets of data. `COLMATCH` provides a more convenient way to map columns of this type than does using table-level mapping with a `COLMAP` clause in individual `TABLE` or `MAP` statements.

Case-sensitivity is supported as follows:

- For MySQL, SQL Server, and Teradata, if the database is case-sensitive, `COLMATCH` looks for an exact case and name match regardless of whether or not a name is specified in quotes.
- For Oracle Database and DB2 databases, where names can be either case-sensitive or case-insensitive in the same database and double quotes are required to show case-sensitivity, `COLMATCH` requires an exact case and name match when a name is in quotes in the database.

Syntax

```
COLMATCH
{NAMES target_column = source_column |
PREFIX prefix |
SUFFIX suffix |
RESET}
```

Argument	Description
`NAMES` *`target_column`* `=` *`source_column`*	Maps based on column names. Put double quotes around the column name if it is case-sensitive and the database requires quotes to enforce case-sensitivity. For these database types, an unquoted column name is treated as case-insensitive by Oracle GoldenGate. For databases that support case-sensitivity without requiring quotes, specify the column name as it is stored in the database. If the `COLMATCH` is between columns in different database types, make certain the names reflect the appropriate case representation for each one. For example, the following specifies a case-sensitive target column name "`aBc`" in an Oracle Database and a case-sensitive source column name `aBc` in a case-sensitive SQL Server database. `COLMATCH NAMES "aBc" = aBc`

Argument	Description

`PREFIX` *`prefix`* `|` `SUFFIX` *`suffix`*	Ignores the specified name prefix or suffix.
	Put double quotes around the prefix or suffix if the database requires quotes to enforce case-sensitivity, for example "`P_`". For those database types, an unquoted prefix or suffix is treated as case-insensitive.
	For databases that support case-sensitivity without requiring quotes, specify the prefix or suffix as it is stored in the database. For example, `P_` specifies a capital `P` prefix.
	The following example specifies a case-insensitive prefix to ignore. The target column name `P_ABC` is mapped to source column name `ABC`, and target column name `P_abc` is mapped to source column name `abc`.
	`COLMATCH PREFIX p_`
	The following example specifies a case-sensitive `suffix` to ignore. The target column name `ABC_k` is mapped to the source column name `ABC`, and the target column name "`abc_k`" is mapped to the source column name "abc".
	`SUFFIX "_k"`

`RESET`	Turns off previously defined `COLMATCH` rules for subsequent `TABLE` or `MAP` statements.

The following example illustrates when to use `COLMATCH.` The source and target tables are identical except for slightly different table and column names.The database is case-insensitive.

ACCT Table	ORD Table

`CUST_CODE`
`CUST_NAME`
`CUST_ADDR`
`PHONE`
`S_REP`
`S_REPCODE`
	`CUST_CODE`
	`CUST_NAME`
	`ORDER_ID`
	`ORDER_AMT`
	`S_REP`
	`S_REPCODE`

ACCOUNT Table	ORDER Table

`CUSTOMER_CODE`
`CUSTOMER_NAME`
`CUSTOMER_ADDRESS`
`PHONE`
`REP`
`REPCODE`
	`CUSTOMER_CODE`
	`CUSTOMER_NAME`
	`ORDER_ID`
	`ORDER_AMT`
	`REP`
	`REPCODE`

To map the source columns to the target columns in this example, as well as to handle subsequent maps for other tables, the syntax is:

```
COLMATCH NAMES CUSTOMER_CODE = CUST_CODE
COLMATCH NAMES CUSTOMER_NAME = CUST_NAME
COLMATCH NAMES CUSTOMER_ADDRESS = CUST_ADDR
COLMATCH PREFIX S_
MAP SALES.ACCT, TARGET SALES.ACCOUNT, COLMAP (USEDEFAULTS);
MAP SALE.ORD, TARGET SALES.ORDER, COLMAP (USEDEFAULTS);
COLMATCH RESET
MAP SALES.REG, TARGET SALE.REG;
MAP SALES.PRICE, TARGET SALES.PRICE;
```

Based on the rules in the example, the following occurs:

- Data is mapped from the `CUST_CODE` columns in the source `ACCT` and `ORD` tables to the `CUSTOMER_CODE` columns in the target `ACCOUNT` and `ORDER` tables.
- The `S_` prefix will be ignored.
- Columns with the same names, such as the `PHONE` and `ORDER_AMT` columns, are automatically mapped by means of `USEDEFAULTS` without requiring explicit rules.
- The previous global column mapping is turned off for the tables `REG` and `PRICE`. Source and target columns in those tables are automatically mapped because all of the names are identical.

Understanding Default Column Mapping

For self-describing trails, if an explicit column mapping does not exist, either by using `COLMATCH` or `COLMAP`, Oracle GoldenGate maps source and target columns by default according to the following rules.

This doesn't apply if you are using `SOURCEDEFS` or `TARGETDEFS`.

- If a source column is found whose name and case exactly match those of the target column, the two are mapped.
- If no case match is found, fallback name mapping is used. Fallback mapping performs a case-insensitive target table mapping to find a name match. Inexact column name matching is applied using upper cased names. This behavior is controlled by the `GLOBALS` parameter `NAMEMATCHIGNORECASE`. You can disable fallback name matching with the `NAMEMATCHEXACT` parameter, or you can keep it enabled but with a warning message by using the `NAMEMATCHNOWARNING` parameter.
- Target columns that do not correspond to any source column take default values determined by the database.

If the default mapping cannot be performed, the target column defaults to one of the values shown in the following table.

Column Type	Value
Numeric	Zero (0)
Character or `VARCHAR`	Spaces
Date or Datetime	Current date and time
Columns that can take a `NULL` value	Null

Data Type Conversions

The following explains how Oracle GoldenGate maps data types.

Numeric Columns

Numeric columns are converted to match the type and scale of the target column. If the scale of the target column is smaller than that of the source, the number is truncated on the right. If the scale of the target column is larger than that of the source, the number is padded with zeros on the right.

You can specify a substitution value for invalid numeric data encountered when mapping number columns by using the `REPLACEBADNUM` parameter. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Character-type Columns

Character-type columns can accept character-based data types such as `VARCHAR`, numeric in string form, date and time in string form, and string literals. If the scale of the target column is smaller than that of the source, the column is truncated on the right. If the scale of the target column is larger than that of the source, the column is padded with spaces on the right.

Literals must be enclosed within single quotes.

You can control the response of the Oracle GoldenGate process when a valid code point does not exist for either the source or target character set when mapping character columns by using the `REPLACEBADCHAR` parameter. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Datetime Columns

Datetime `(DATE`, `TIME`, and `TIMESTAMP`) columns can accept datetime and character columns, as well as string literals. Literals must be enclosed within single quotes. To map a character column to a datetime column, make certain it conforms to the Oracle GoldenGate external SQL format of `YYYY-MM-DD HH:MI:SS.FFFFFF`.

Oracle GoldenGate supports timestamp data from `0001-01-03 00:00:00` to `9999-12-31 23:59:59`. If a timestamp is converted from GMT to local time, these limits also apply to the resulting timestamp. Depending on the timezone, conversion may add or subtract hours, which can cause the timestamp to exceed the lower or upper supported limit.

Required precision varies according to the data type and target platform. If the scale of the target column is smaller than that of the source, data is truncated on the right. If the scale of the target column is larger than that of the source, the column is extended on the right with the values for the current date and time.

Selecting and Converting SQL Operations

By default, Oracle GoldenGate captures and applies `INSERT`, `UPDATE`, and `DELETE` operations. You can use the following parameters in the Extract or Replicat parameter file to control which kind of operations are processed, such as only inserts or only inserts and updates.

```
GETINSERTS | IGNOREINSERTS
```

```
GETUPDATES | IGNOREUPDATES
```

```
GETDELETES | IGNOREDELETES
```

You can convert one type of SQL operation to another by using the following parameters in the Replicat parameter file:

- Use `INSERTUPDATES` to convert source update operations to inserts into the target table. This is useful for maintaining a transaction history on that table. The transaction log record must contain all of the column values of the table, not just changed values. Some databases do not log full row values to their transaction log, but only values that changed.
- Use `INSERTDELETES` to convert all source delete operations to inserts into the target table. This is useful for retaining a history of all records that were ever in the source database.
- Use `UPDATEDELETES` to convert source deletes to updates on the target.

Selecting and Filtering Rows

Filtering can only be performed on columns that are available to Oracle GoldenGate. In the `TRANLOG` Extract Oracle GoldenGate has access to all columns that are present in the redo logs and in the database. If the columns are not in the redo logs, they must be explicitly fetched (using `FETCHCOLS`) to be able to filter them. In the Extract pump and in the Replicat, the columns must be available in the trail file. Because of this, any column that you want to use in a `FILTER` or `WHERE` clause must be explicitly logged using `ADD TRANDATA COLS`, and you have to retain the default of `LOGALLSUPCOLS`.

To filter out or select rows for extraction or replication, use the `FILTER` and `WHERE` clauses of the `TABLE` and `MAP` parameters.

The `FILTER` clause offers you more functionality than the `WHERE` clause because you can employ any of the Oracle GoldenGate column conversion functions, whereas the `WHERE` clause accepts basic `WHERE` operators.

Selecting Rows with a FILTER Clause

Use a `FILTER` clause to select rows based on a numeric value by using basic operators or one or more Oracle GoldenGate column-conversion functions.

> **Note:**
>
> To filter a column based on a string, use one of the Oracle GoldenGate string functions or use a `WHERE` clause.

The syntax for `FILTER` in a `TABLE` statement is as follows:

```
TABLE source_table,
, FILTER (
[, ON INSERT | ON UPDATE| ON DELETE]
[, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]
, filter_clause);
```

The syntax for `FILTER` in a `MAP` statement is as follows and includes an error-handling option.

```
MAP source_table, TARGET target_table,
, FILTER (
[, ON INSERT | ON UPDATE| ON DELETE]
[, IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE]
[, RAISEERROR error_number]
, filter_clause);
```

Valid `FILTER` clause elements are the following:

- An Oracle GoldenGate column-conversion function. These functions are built into Oracle GoldenGate so that you can perform tests, manipulate data, retrieve values, and so forth. See Testing and Transforming Data for more information about Oracle GoldenGate conversion functions.
- Numbers
- Columns that contain numbers
- Functions that return numbers
- Arithmetic operators:
  - `+` (plus)
  - `-` (minus)
  - `*` (multiply)
  - `/` (divide)
  - `\` (remainder)
- Comparison operators:
  - `>` (greater than)
  - `>=` (greater than or equal)
  - `<` (less than)
  - `<=` (less than or equal)
  - `=` (equal)
  - `<>` (not equal)
  - Results derived from comparisons can be zero (indicating `FALSE`) or non-zero (indicating `TRUE`).
- Parentheses (for grouping results in the expression)
- Conjunction operators: `AND`, `OR`

Use the following `FILTER` options to specify which SQL operations a filter clause affects. Any of these options can be combined.

```
ON INSERT | ON UPDATE | ON DELETE IGNORE INSERT | IGNORE UPDATE | IGNORE DELETE
```

Use the `RAISEERROR` option of `FILTER` in the `MAP` parameter to generate a user-defined error when the filter fails. This option is useful when you need to trigger an event in response to the failure.

Use the `@RANGE` function within a `FILTER` clause to distribute the processing workload among multiple `MAP` or `TABLE` statements.

Here's a sample:

```
REPERROR (9999, EXCEPTION)
MAP OWNER.SRCTAB, TARGET OWNER.TARGTAB,
                   SQLEXEC (ID CHECK, ON UPDATE, QUERY ' SELECT COUNT FROM
TARGTAB WHERE PKCOL = :P1 ', PARAMS (P1 = PKCOL)),
                   FILTER (BALANCE > 15000),
                   FILTER (ON UPDATE, @BEFORE (COUNT) = CHECK.COUNT)
;
MAP OWNER.SRCTAB, TARGET OWNER.TARGEXC,
EXCEPTIONSONLY,
COLMAP (   USEDEFAULTS,
ERRTYPE = 'UPDATE FILTER FAILED'
)
;
```

Table 8-3 Using Multiple FILTER Statements

Parameter file	Description
`REPERROR (9999, EXCEPTION)`	Raises an exception for the specified error.

`MAP OWNER.SRCTAB,`
`TARGET OWNER.TARGTAB,`
	Starts the `MAP` statement.

`SQLEXEC (ID CHECK, ON UPDATE,`
`QUERY ' SELECT COUNT FROM TARGTAB '`
`'WHERE PKCOL = :P1 ',`
`PARAMS (P1 = PKCOL)),`
	Performs a query to retrieve the present value of the `COUNT` column whenever an update is encountered. There is a `BEFOREFILTER` option also that allows the query or stored procedure to be executed prior to processing the `FILTER` clause. This allows values from the `SQLEXEC` portion to be used inside the `FILTER` at runtime.

`FILTER (BALANCE > 15000),`	Uses a `FILTER` clause to select rows where the balance is greater than 15000.
`FILTER (ON UPDATE, @BEFORE (COUNT) = CHECK.COUNT)`	Uses another `FILTER` clause to ensure that the value of the source `COUNT` column before an update matches the value in the target column before applying the target update.
`;`	The semicolon concludes the `MAP` statement.

`MAP OWNER.SRCTAB,`
`TARGET OWNER.TARGEXC,`
`EXCEPTIONSONLY,`
`COLMAP (USEDEFAULTS,`
`ERRTYPE = 'UPDATE FILTER FAILED');`
	Designates an exceptions `MAP` statement. The `REPERROR` clause for error 9999 ensures that the exceptions map to `TARGEXC` will be executed.

Example 8-3 Calling the @COMPUTE Function

The following example calls the `@COMPUTE` function to extract records in which the price multiplied by the amount exceeds 10,000.

```
MAP SALES.TCUSTORD, TARGET SALES.TORD,
FILTER (@COMPUTE (PRODUCT_PRICE * PRODUCT_AMOUNT) > 10000);
```

Example 8-4 Calling the @STREQ Function

The following uses the `@STREQ` function to extract records where the value of a character column is `'JOE'`.

```
TABLE ACCT.TCUSTORD, FILTER (@STREQ ("Name", 'joe') > 0);
```

Example 8-5 Selecting Records

The following selects records in which the `AMOUNT` column is greater than 50 and executes the filter on `UPDATE` and `DELETE` operations.

```
TABLE ACT.TCUSTORD, FILTER (ON UPDATE, ON DELETE, AMOUNT > 50);
```

Example 8-6 Using the @RANGE Function

(Replicat group 1 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (1, 2, ID));
```

(Replicat group 2 parameter file)

```
MAP sales.acct, TARGET sales.acct, FILTER (@RANGE (2, 2, ID));
```

You can combine several `FILTER` clauses in one `MAP` or `TABLE` statement, as shown in #unique_738/unique_738_Connect_42_G1110854, which shows part of a Replicat parameter file. Oracle GoldenGate executes the filters in the order listed, until one fails or until all are passed. If one filter fails, they all fail.

Selecting Rows with a WHERE Clause

Use any of the elements described in the table below in a `WHERE` clause to select or exclude rows (or both) based on a conditional statement. Each `WHERE` clause must be enclosed within parentheses. Literals must be enclosed within single quotes.

Table 8-4 Permissible WHERE Operators

Element	Examples
Column names	`PRODUCT_AMT`
Numeric values	`-123, 5500.123`
Literal strings	`'AUTO', 'Ca'`
Built-in column tests	`@NULL`, `@PRESENT`, `@ABSENT` (column is null, present or absent in the row). These tests are built into Oracle GoldenGate. See Considerations for Selecting Rows with FILTER and WHERE.
Comparison operators	=, <>, >, <, >=, <=
Conjunctive operators	`AND, OR`
Grouping parentheses	Use open and close parentheses ( ) for logical grouping of multiple elements.

Oracle GoldenGate does not support `FILTER` for columns that have a multi-byte character set or a character set that is incompatible with the character set of the local operating system.

Arithmetic operators and floating-point data types are not supported by `WHERE`. To use more complex selection conditions, use a `FILTER` clause or a user exit routine. See Using User Exits to Extend Oracle GoldenGate Capabilities for more information.

The syntax for `WHERE` is identical in the `TABLE` and `MAP` statements:

```
TABLE table, WHERE (clause);
```

```
MAP source_table, TARGET target_table, WHERE (clause);
```

Considerations for Selecting Rows with FILTER and WHERE

The following suggestions can help you create a successful selection clause.

> **Note:**
>
> The examples in this section assume a case-insensitive database.

Ensuring Data Availability for Filters

If the database only logs values for *changed* columns to the transaction log, there can be errors if any of the unchanged columns are referenced by selection criteria. Oracle GoldenGate ignores such row operations, outputs them to the discard file, and issues a warning.

To avoid missing-column errors, create your selection conditions as follows:

- Use only primary-key columns as selection criteria, if possible.
- Make required column values available by enabling supplemental logging for those columns. Alternatively, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter. These options are valid for all supported databases. They query the database to fetch the values if they are not present in the log. To retrieve the values before the `FILTER` or `WHERE` clause is executed, include the `FETCHBEFOREFILTER` option in the `TABLE` statement before the `FILTER` or `WHERE` clause. For example:

  ```
  TABLE DEMO.PEOPLE, FETCHBEFOREFILTER, FETCHCOLS (age), FILTER (age > 50);
  ```

- Test for a column's presence first, then for the column's value. To test for a column's presence, use the following syntax.

  ```
  column_name {= | <>} {@PRESENT | @ABSENT}
  ```

  The following example returns all records when the `amount` column is over 10,000 and does not cause a record to be discarded when `amount` is absent.

  ```
  WHERE (amount = @PRESENT AND amount > 10000)
  ```

Comparing Column Values

To ensure that elements used in a comparison match, compare appropriate column types:

- Character columns to literal strings.
- Numeric columns to numeric values, which can include a sign and decimal point.
- Date and time columns to literal strings, using the format in which the column is retrieved by the application.

Testing for NULL Values

To evaluate columns for `NULL` values, use the following syntax.

```
column {= | <>} @NULL
```

The following returns `TRUE` if the column value is `NULL`, and thereby replicates the row. It returns `FALSE` for all other cases (including a column missing from the record).

```
WHERE (amount = @NULL)
```

The following returns `TRUE` only if the column is present in the record and is not `NULL`.

```
WHERE (amount = @PRESENT AND amount <> @NULL)
```

> **Note:**
>
> If a value in the trail contains more than 4000 bytes then the `@NULL` function will return `TRUE`.

Retrieving Before and After Values

For update and delete operations, it can be useful to retrieve the `BEFORE` values of the source columns (the values before the update occurred). For inserts, all column values are considered `AFTER` images.

These values are stored in the trail and can be used in filters and column mappings. For example, you can:

- Retrieve the before image of a row as part of a column-mapping specification in an exceptions `MAP` statement, and map those values to an exceptions table for use in testing or troubleshooting conflict resolution routines.
- Perform delta calculations. For example, if a table has a `Balance` column, you can calculate the net result of a particular transaction by subtracting the original balance from the new balance, as in the following example:

  ```
  MAP "owner"."src", TARGET "owner"."targ",
  COLMAP (PK1 = PK1, delta = balance - @BEFORE (balance));
  ```

  > **Note:**
  >
  > The previous example indicates a case-sensitive database such as Oracle. The table names are in quote marks to reflect case-sensitivity.

To Reference the Before Value

1. Use the `@BEFORE` column conversion function with the name of the column for which you want a before value, as follows:

   ```
   @BEFORE (column_name)
   ```

2. Use the `GETUPDATEBEFORES` parameter in the Extract parameter file to capture before images from the transaction record, or use it in the Replicat parameter file to use the before image in a column mapping or filter. If using the Conflict Resolution and Detection (CDR) feature, you can use the `GETBEFORECOLS` option of `TABLE`. To use these parameters, all columns must be present in the transaction log. If the database only logs the values of columns that changed, using the `@BEFORE` function may result in a "column missing" condition and the column map is executed as if the column were not in the record. See Ensuring Data Availability for Filters to ensure that column values are available.

Oracle GoldenGate also provides the `@AFTER` function to retrieve after values when needed for filtering, for use in conversion functions, or other purposes. For more information about `@BEFORE` and `@AFTER`, see *Parameters and Functions Reference for Oracle GoldenGate*.

Selecting Columns

To control which columns of a source table are extracted by Oracle GoldenGate, use the `COLS` and `COLSEXCEPT` options of the `TABLE` parameter. Use `COLS` to select columns for extraction, and use `COLSEXCEPT` to select all columns except those designated by `COLSEXCEPT`.

Restricting the columns that are extracted can be useful when a target table does not contain the same columns as the source table, or when the columns contain sensitive information, such as a personal identification number or other proprietary business information.

Using Transaction History

Oracle GoldenGate enables you to retain a history of changes made to a target record and to map information about the operation that caused each change. This history can be useful for creating a transaction-based reporting system that contains a separate record for every operation performed on a table, as opposed to containing only the most recent version of each record.

For example, the following series of operations made to a target table named `CUSTOMER` would leave no trace of the ID of `Dave`. The last operation deletes the record, so there is no way to find out Dave's account history or his ending balance.

Table 8-5 Operation History for Table CUSTOMER

Sequence	Operation	ID	BALANCE
1	`Insert`	`Dave`	`1000`
2	`Update`	`Dave`	`900`
3	`Update`	`Dave`	`1250`
4	`Delete`	`Dave`	`1250`

Retaining this history as a series of records can be useful in many ways. For example, you can generate the net effect of transactions.

To Implement Transaction Reporting

1. To prepare Extract to capture before values, use the `GETUPDATEBEFORES` parameter in the Extract parameter file. A before value (or before image) is the existing value of a column before an update is performed. Before images enable Oracle GoldenGate to create the transaction record.
2. To prepare Replicat to post all operations as inserts, use the `INSERTALLRECORDS` parameter in the Replicat parameter file. Each operation on a table becomes a new record in that table.
3. To map the transaction history, use the return values of the `GGHEADER` option of the `@GETENV` column conversion function. Include the conversion function as the source expression in a `COLMAP` statement in the `TABLE` or `MAP` parameter.

Using the sample series of transactions shown in #unique_748/unique_748_Connect_42_G1111011 the following parameter configurations can be created to generate a more transaction-oriented view of customers, rather than the latest state of the database.

Process	Parameter statements

Extract	`GETUPDATEBEFORES`
	`TABLE ACCOUNT.CUSTOMER;`

Replicat	`INSERTALLRECORDS`
	`MAP SALES.CUSTOMER, TARGET SALES.CUSTHIST,`
	`COLMAP (TS = @GETENV ('GGHEADER', 'COMMITTIMESTAMP'),`
	`BEFORE_AFTER = @GETENV ('GGHEADER', 'BEFOREAFTERINDICATOR'),`
	`OP_TYPE = @GETENV ('GGHEADER', 'OPTYPE'),`
	`ID = ID,`
	`BALANCE = BALANCE);`

Note:

This is not representative of a complete parameter file for an Oracle GoldenGate process. Also note that these examples represent a case-insensitive database.

This configuration makes possible queries such as the following, which returns the net sum of each transaction along with the time of the transaction and the customer ID.

```
SELECT AFTER.ID, AFTER.TS, AFTER.BALANCE - BEFORE.BALANCE
FROM CUSTHIST AFTER, CUSTHIST BEFORE
WHERE AFTER.ID = BEFORE.ID AND AFTER.TS = BEFORE.TS AND
AFTER.BEFORE_AFTER = 'A' AND BEFORE.BEFORE_AFTER = 'B';
```

Testing and Transforming Data

Data testing and transformation can be performed by either Extract or Replicat and is implemented by using the Oracle GoldenGate built-in column-conversion functions within a `COLMAP` clause of a `TABLE` or `MAP` statement. With these conversion functions, you can:

- Transform dates.
- Test for the presence of column values.
- Perform arithmetic operations.
- Manipulate numbers and character strings.
- Handle null, invalid, and missing data.
- Perform tests.

This chapter provides an overview of some of the Oracle GoldenGate functions related to data manipulation. For the complete reference, see Reference for Oracle GoldenGate for Windows and UNIX.

If you need to use logic beyond that which is supplied by the Oracle GoldenGate functions, you can call your own functions by implementing Oracle GoldenGate user exits. See Using User Exits to Extend Oracle GoldenGate Capabilities for more information about user exits.

Oracle GoldenGate conversion functions take the following general syntax:

Syntax

`@function (argument)`

Table 8-6 Conversion Function Syntax

Syntax element	Description
@*`function`*	The Oracle GoldenGate function name. Function names have the prefix `@`, as in `@COMPUTE` or `@DATE`. A space between the function name and the open-parenthesis before the input argument is optional.
`argument`	A function argument.

Table 8-7 Function Arguments

Argument element	Example
A numeric constant	`123`
A string literal enclosed within single quote marks	`'ABCD'`
The name of a source column	`PHONE_NO or phone_no, or "Phone_No" or Phone_no` Depends on whether the database is case-insensitive, is case-sensitive and requires quote marks to enforce the case, or is case-sensitive and does not require quotes.
An arithmetic expression	`COL2 * 100`
A comparison expression	`((COL3 > 100) AND (COL4 > 0))`
Other Oracle GoldenGate functions	`AMOUNT = @IF (@COLTEST (AMT, MISSING, INVALID), 0, AMT)`

Handling Column Names and Literals in Functions

By default, literal strings must be enclosed in single quotes in a column-conversion function. Case-sensitive column names must be enclosed within double quotes if required by the database, or otherwise entered in the case in which they are stored in the database.

Using the Appropriate Function

Use the appropriate function for the type of column that is being manipulated or evaluated. For example, numeric functions can be used only to compare numeric values. To compare character values, use one of the Oracle GoldenGate character-comparison functions. LOB columns cannot be used in conversion functions.

This statement would fail because it uses `@IF`, which is a numerical function, to compare string values.

```
@IF (SR_AREA = 'Help Desk', 'TRUE', 'FALSE')
```

The following statement would succeed because it compares a numeric value.

```
@IF (SR_AREA = 20, 'TRUE', 'FALSE')
```

See Manipulating Numbers and Character Strings for more information.

> **Note:**
>
> Errors in argument parsing sometimes are not detected until records are processed. Verify syntax before starting processes.

Transforming Dates

Use the `@DATE`, `@DATEDIF`, and `@DATENOW` functions to retrieve dates and times, perform computations on them, and convert them.

This example computes the time that an order is filled

Example 8-7 Computing Time

```
ORDER_FILLED = @DATE (
    'YYYY-MM-DD HH:MI:SS',
    'JTS',
    @DATE ('JTS',
    'YYMMDDHHMISS',
    ORDER_TAKEN_TIME) +
    ORDER_MINUTES * 60 * 1000000)
```

Performing Arithmetic Operations

To return the result of an arithmetic expression, use the `@COMPUTE` function. The value returned from the function is in the form of a string. Arithmetic expressions can be combinations of the following elements.

- Numbers
- The names of columns that contain numbers
- Functions that return numbers
- Arithmetic operators:
  - `+` (plus)
  - `-` (minus)
  - `*` (multiply)
  - `/` (divide)
  - `\` (remainder)
- Comparison operators:
  - `>` (greater than)

  - >= (greater than or equal)
  - < (less than)
  - <= (less than or equal)
  - = (equal)
  - <> (not equal)

  Results that are derived from comparisons can be zero (indicating `FALSE`) or non-zero (indicating `TRUE`).

- Parentheses (for grouping results in the expression)
- The conjunction operators `AND`, `OR`. Oracle GoldenGate only evaluates the necessary part of a conjunction expression. Once a statement is `FALSE`, the rest of the expression is ignored. This can be valuable when evaluating fields that may be missing or null. For example, if the value of `COL1` is 25 and the value of `COL2` is 10, then the following are possible:

```
@COMPUTE ( (COL1 > 0) AND (COL2 < 3) ) returns 0.
@COMPUTE ( (COL1 < 0) AND (COL2 < 3) ) returns 0. COL2 < 3 is never evaluated.
@COMPUTE ((COL1 + COL2)/5) returns 7.
```

Omitting @COMPUTE

The `@COMPUTE` keyword is not required when an expression is passed as a function argument.

```
@STRNUM ((AMOUNT1 + AMOUNT2), LEFT)
```

The following expression returns the same result as the previous one:

```
@STRNUM ((@COMPUTE (AMOUNT1 + AMOUNT2), LEFT)
```

Manipulating Numbers and Character Strings

To convert numbers and character strings, Oracle GoldenGate supplies the following functions:

Table 8-8 Conversion Functions for Numbers and Characters

Purpose	Conversion Function
Convert a binary or character string to a number.	`@NUMBIN` `@NUMSTR`
Convert a number to a string.	`@STRNUM`
Compare strings.	`@STRCMP` `@STRNCMP`
Concatenate strings.	`@STRCAT` `@STRNCAT`
Extract from a string.	`@STREXT` `@STRFIND`
Return the length of a string.	`@STRLEN`
Substitute one string for another.	`@STRSUB`
Convert a string to upper case.	`@STRUP`

Table 8-8 (Cont.) Conversion Functions for Numbers and Characters

Purpose	Conversion Function

Trim leading or trailing spaces, or both.	`@STRLTRIM`
	`@STRRTRIM`
	`@STRTRIM`

Handling Null, Invalid, and Missing Data

When column data is missing, invalid, or null, an Oracle GoldenGate conversion function returns a corresponding value.

If `BALANCE` is `1000`, but `AMOUNT` is `NULL`, the following expression returns `NULL`:

```
NEW_BALANCE = @COMPUTE (BALANCE + AMOUNT)
```

These exception conditions render the entire calculation invalid. To ensure a successful conversion, use the `@COLSTAT`, `@COLTEST` and `@IF` functions to test for, and override, the exception condition.

Using @COLSTAT

Use the `@COLSTAT` function to return an indicator to Extract or Replicat that a column is missing, null, or invalid. The indicator can be used as part of a larger manipulation formula that uses additional conversion functions.

The following example returns a `NULL` into target column `ITEM`.

```
ITEM = @COLSTAT (NULL)
```

The following `@IF` calculation uses `@COLSTAT` to return `NULL` to the target column if `PRICE` and `QUANTITY` are less than zero.

```
ORDER_TOTAL = PRICE * QUANTITY, @IF ((PRICE < 0) AND (QUANTITY < 0), @COLSTAT (NULL))
```

Using @COLTEST

Use the `@COLTEST` function to check for the following conditions:

- `PRESENT` tests whether a column is present and not null.
- `NULL` tests whether a column is present and null.
- `MISSING` tests whether a column is not present.
- `INVALID` tests whether a column is present but contains invalid data.

The following example checks whether the `AMOUNT` column is present and `NULL` and whether it is present but invalid.

```
@COLTEST (AMOUNT, NULL, INVALID)
```

Using @IF

Use the `@IF` function to return one of two values based on a condition. Use it with the `@COLSTAT` and `@COLTEST` functions to begin a conditional argument that tests for one or more exception conditions and then directs processing based on the results of the test.

```
NEW_BALANCE = @IF (@COLTEST (BALANCE, NULL, INVALID) OR
@COLTEST (AMOUNT, NULL, INVALID), @COLSTAT (NULL), BALANCE + AMOUNT)
```

This conversion returns one of the following:

- `NULL` when `BALANCE` or `AMOUNT` is `NULL` or `INVALID`
- `MISSING` when either column is missing
- The sum of the columns.

Performing Tests

The `@CASE`, `@VALONEOF`, and `@EVAL` functions provide additional methods for performing tests on data before manipulating or mapping it.

Using @CASE

Use `@CASE` to select a value depending on a series of value tests.

```
@CASE (PRODUCT_CODE, 'CAR', 'A car', 'TRUCK', 'A truck')
```

This example returns the following:

- `A car` if `PRODUCT_CODE` is `CAR`
- `A truck` if `PRODUCT_CODE` is `TRUCK`
- A `FIELD_MISSING` indication if `PRODUCT_CODE` fits neither of the other conditions

Using @VALONEOF

Use `@VALONEOF` to compare a column or string to a list of values.

```
@IF (@VALONEOF (STATE, 'CA', 'NY'), 'COAST', 'MIDDLE')
```

In this example, if `STATE` is `CA` or `NY`, the expression returns `COAST`, which is the response returned by `@IF` when the value is non-zero (meaning `TRUE`).

Using @EVAL

Use `@EVAL` to select a value based on a series of independent conditional tests.

```
@EVAL (AMOUNT > 10000, 'high amount', AMOUNT > 5000, 'somewhat high')
```

This example returns the following:

- `high amount` if `AMOUNT` is greater than `10000`
- `somewhat high` if `AMOUNT` is greater than `5000`, and less than or equal to `10000`, (unless the prior condition was satisfied)
- A `FIELD_MISSING` indication if neither condition is satisfied.

Using Tokens

You can capture and store data within the *user token* area of a trail record header. Token data can be retrieved and used in many ways to customize the way that Oracle GoldenGate delivers information. For example, you can use token data in:

- Column maps

- Stored procedures called by a `SQLEXEC` statement
- User exits
- Macros

Defining Tokens

To use tokens, you define the token name and associate it with data. The data can be any valid character data or values retrieved from Oracle GoldenGate column-conversion functions.

The token area in the record header permits up to 16,000 bytes of data. Token names, the length of the data, and the data itself must fit into that space.

To define a token, use the `TOKENS` option of the `TABLE` parameter in the Extract parameter file.

Syntax

```
TABLE table_spec, TOKENS (token_name = token_data [, ...]);
```

Where:

- *`table_spec`* is the name of the source table. A container or catalog name, if applicable, and an owner name must precede the table name.
- *`token_name`* is a name of your choice for the token. It can be any number of alphanumeric characters and is not case-sensitive.
- *`token_data`* is a character string of up to 2000 bytes. The data can be either a string that is enclosed within single quotes or the result of an Oracle GoldenGate column-conversion function. The character set of token data is not converted. The token must be in the character set of the source database for Extract and in the character set of the target database for Replicat. In the trail file, user tokens are stored in UTF-8.

```
TABLE ora.oratest, TOKENS (
TK-OSUSER = @GETENV ('GGENVIRONMENT' , 'OSUSERNAME'),
TK-GROUP = @GETENV ('GGENVIRONMENT' , 'GROUPNAME')
TK-HOST =  @GETENV('GGENVIRONMENT' , 'HOSTNAME'));
```

As shown in this example, the Oracle GoldenGate `@GETENV` function is an effective way to populate token data. This function provides several options for capturing environment information that can be mapped to tokens and then used on the target system for column mapping.

Using Token Data in Target Tables

To map token data to a target table, use the `@TOKEN` column-conversion function in the source expression of a `COLMAP` clause in a Replicat `MAP` statement. The `@TOKEN` function provides the name of the token to map. The `COLMAP` syntax with `@TOKEN` is:

Syntax

```
COLMAP (target_column = @TOKEN ('token_name'))
```

The following `MAP` statement maps target columns `host`, `gg_group`, and so forth to tokens `tk-host`, `tk-group`, and so forth. Note that the arguments must be enclosed within single quotes.

User tokens	Values
`tk-host`	`:sysA`
`tk-group`	`:extora`
`tk-osuser`	`:jad`
`tk-domain`	`:admin`
`tk-ba_ind`	`:B`
`tk-commit_ts`	`:2011-01-24 17:08:59.000000`
`tk-pos`	`:3604496`
`tk-rba`	`:4058`
`tk-table`	`:oratest`
`tk-optype`	`:insert`

Example 8-8 MAP Statement

```
MAP ora.oratest, TARGET ora.rpt,
COLMAP (USEDEFAULTS,
host = @token ('tk-host'),
gg_group = @token ('tk-group'),
osuser= @token ('tk-osuser'),
domain = @token ('tk-domain'),
ba_ind= @token ('tk-ba_ind'),
commit_ts = @token ('tk-commit_ts'),
pos = @token ('tk-pos'),
rba = @token ('tk-rba'),
tablename = @token ('tk-table'),
optype = @token ('tk-optype'));
```

The tokens in this example will look similar to the following within the record header in the trail:

Error Management

Learn about configuring the Oracle GoldenGate processes to handle errors.

Oracle GoldenGate reports processing errors in several ways by means of its monitoring and reporting tools.

Also see: #unique_767.

Automatic Conflict Detection and Resolution

You can configure Oracle GoldenGate to automatically detect and resolve conflicts that occur when same data is updated concurrently at different sites.

Conflict detection and resolution is required in active-active configurations, where Oracle GoldenGate must maintain data synchronization among multiple databases that contain the same data sets.

About Automatic Conflict Detection and Resolution

When Oracle GoldenGate replicates changes between Oracle databases, you can configure and manage Oracle GoldenGate conflict detection and resolution automatically in these databases.

This feature is intended for use with active-active configurations, where Oracle GoldenGate must maintain data synchronization among multiple databases that contain the same data sets.

> **Note:**
>
> Automatic conflict detection and resolution (ACDR) feature that is available only when using Oracle GoldenGate with Oracle Database. For non-Oracle databases, there is a manual conflict detection and resolution (CDR) feature available with Oracle GoldenGate. Oracle GoldenGate CDR is configured in the Replicat parameter file.

Automatic Conflict Detection and Resolution

You can configure automatic conflict detection and resolution in an Oracle GoldenGate configuration that replicates tables between Oracle databases. To configure automatic conflict detection and resolution for a table, you need to call the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package. A prerequisite for setting up automatic conflict detection and resolution, the Oracle GoldenGate user must have the appropriate privileges. See Grant User Privileges for Oracle Database for Oracle GoldenGate Classic Architecture learn about user privileges.

The administrator user must be logged in to the appropriate PDB when calling the `ADD_AUTO_CDR`. The following constants, which represent bit flags are now added:

- `EARLIEST_TIMESTAMP_RESOLUTION` sets `TOMBSTONE KEY VERSIONING` automatically
- `DELETE_ALWAYS_WINS` sets `TOMBSTONE KEY VERSIONING` automatically.
- `IGNORE_SITE_PRIORITY`

The following example uses an `ALTER` command for the `HR.EMPLOYEES` table:

```
BEGIN
 dbms_goldengate_adm.alter_auto_cdr
 (schema_name => 'HR'
 ,table_name => 'EMPLOYEES'
 ,additional_options =>
DBMS_GOLDENGATE_ADM.ADDITIONAL_OPTIONS_ADD_KEY_VERSION );
END;
/
```

See the description for `additional_options` in `ADD_AUTO_CDR Procedure` of *Oracle Database PL/SQL Packages and Types Reference*.

When Oracle GoldenGate captures changes that originated at an Oracle Database, each change is encapsulated in a row logical change record (LCR). A row LCR is a structured representation of a DML row change. Each row LCR includes the operation type, old column values, and new column values. Multiple row LCRs can be part of a single database transaction.

When more than one replica of a table allows changes to the table, a conflict can occur when a change is made to the same row in two different databases at nearly the same time. Oracle GoldenGate replicates changes using the row LCRs. It detects a conflict by comparing the old values in the row LCR for the initial change from the origin database with the current values of the corresponding table row at the destination database identified by the key columns. If any column value does not match, then there is a conflict.

After a conflict is detected, Oracle GoldenGate can resolve the conflict by overwriting values in the row with some values from the row LCR, ignoring the values in the row LCR, or computing a delta to update the row values.

Automatic conflict detection and resolution does not require application changes for the following reasons:

- Oracle Database automatically creates and maintains invisible timestamp columns.
- Inserts, updates, and deletes use the delete tombstone log table to determine if a row was deleted.
- LOB column conflicts can be detected.
- Oracle Database automatically configures supplemental logging on required columns.

Requirements for Automatic Conflict Detection and Resolution

Supplemental logging is required to ensure that each row LCR has the information required to detect and resolve a conflict. Supplemental logging places additional information in the redo log for the columns of a table when a DML operation is performed on the table. When you configure a table for Oracle GoldenGate conflict detection and resolution, supplemental logging is configured automatically for all of the columns in the table. The additional information in the redo log is placed in an LCR when a table change is replicated.

Extract must be used for capturing. Integrated Replicat or parallel Replicat in integrated mode must be used on the apply side. `LOGALLSUPCOLS` should remain the default.

There is a hidden field `KEYVER$$` of type timestamp that is optionally added to the `DELETE TOMBSTONE` table. This field is required for `EARLIEST TIMESTAMP`, `DELETE ALWAYS WINS`, and `SITE PRIORITY` resolution and it also exists in the base table. The existence of the field in the base table needs to be provided in the trail file metadata as a flag or token.

Primary Key updates is also supported in the `DELETE TOMBSTONE` table. An entry is inserted into the `DELETE TOMBSTONE` table for the row of the original key value (before image). The logic in the Extract which matches inserts in the `DELETE TOMBSTONE` table to deletes also needs to be matched to PK updates, or unique key (UK) with at least one non-nullable field, if there is no PK.

Site priority needs support from the Replicat, both the parameters are implemented and the setting is passed to the apply.

Latest Timestamp Conflict Detection and Resolution

When you run the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package to configure a table for automatic Oracle GoldenGate conflict detection and resolution, a hidden timestamp column is added to the table. This hidden timestamp column records the time of a row change, and this information is used to detect and resolve conflicts.

When a row LCR is applied, a conflict can occur for an `INSERT`, `UPDATE`, or `DELETE` operation. The following table describes each type of conflict and how it is resolved.

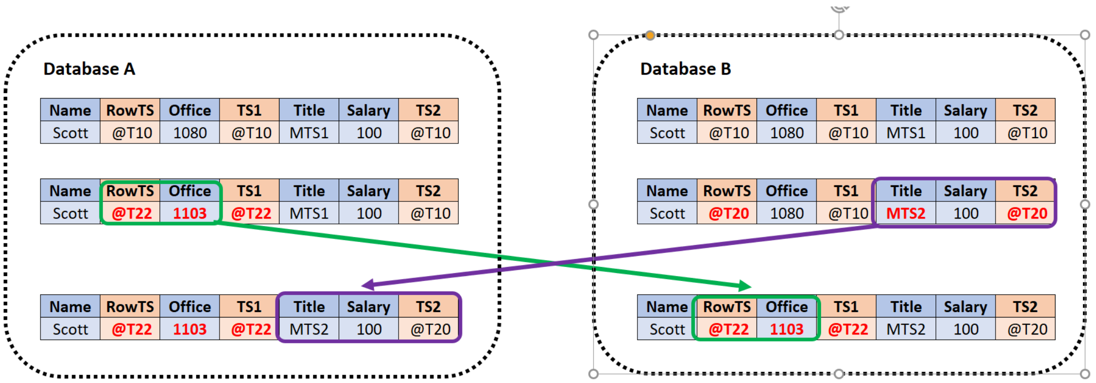
Operation	Conflict Detection	Conflict Resolution
`INSERT`	A conflict is detected when the table has the same value for a key column as the new value in the row LCR.	If the timestamp of the row LCR is later than the timestamp in the table row, then the values in the row LCR replace the values in the table. If the timestamp of the row LCR is earlier than the timestamp in the table row, then the row LCR is discarded, and the table values are retained.
`UPDATE`	A conflict is detected in each of the following cases: • There is a mismatch between the timestamp value in the row LCR and the timestamp value of the corresponding row in the table. • There is a mismatch between an old value in a column group in the row LCR does not match the column value in the corresponding table row. A column group is a logical grouping of one or more columns in a replicated table. • The table row does not exist. If the row is in the tombstone table, then this is referred to as an update-delete conflict.	If there is a value mismatch and the timestamp of the row LCR is later than the timestamp in the table row, then the values in the row LCR replace the values in the table. If there is a value mismatch and the timestamp of the row LCR is earlier than the timestamp in the table row, then the row LCR is discarded, and the table values are retained. If the table row does not exist and the timestamp of the row LCR is later than the timestamp in the tombstone table row, then the row LCR is converted from an `UPDATE` operation to an `INSERT` operation and inserted into the table. If the table row does not exist and the timestamp of the row LCR is earlier than the timestamp in the tombstone table row, then the row LCR is discarded. If the table row does not exist and there is no corresponding row in the tombstone table, then the row LCR is converted from an `UPDATE` operation to an `INSERT` operation and inserted into the table.

Operation	Conflict Detection	Conflict Resolution
`DELETE`	A conflict is detected in each of the following cases: • There is a mismatch between the timestamp value in the row LCR and the timestamp value of the corresponding row in the table. • The table row does not exist.	If the timestamp of the row LCR is later than the timestamp in the table, then delete the row from the table. If the timestamp of the row LCR is earlier than the timestamp in the table, then the row LCR is discarded, and the table values are retained. If the delete is successful, then log the row LCR by inserting it into the tombstone table. If the table row does not exist, then log the row LCR by inserting it into the tombstone table.

Delta Conflict Detection and Resolution

With delta conflict detection, a conflict occurs when a value in the old column list of the row LCR differs from the value for the corresponding row in the table.

To configure delta conflict detection and resolution for a table, run the `ADD_AUTO_CDR_DELTA_RES` procedure in the `DBMS_GOLDENGATE_ADM` package. The delta resolution method does not depend on a timestamp or an extra resolution column. With delta conflict resolution, the conflict is resolved by adding the difference between the new and old values in the row LCR to the value in the table. This resolution method is generally used for financial data such as an account balance. For example, if a bank balance is updated at two sites concurrently, then the converged value accounts for all debits and credits.



Database A

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T10	1080	@T10	MTS1	100	@T10

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T22	1103	@T22	MTS1	100	@T10

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T22	1103	@T22	MTS2	100	@T20

Database B

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T10	1080	@T10	MTS1	100	@T10

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T20	1080	@T10	MTS2	100	@T20

Name	RowTS	Office	TS1	Title	Salary	TS2
Scott	@T22	1103	@T22	MTS2	100	@T20

This example shows a row being replicated at database A and database B. The `Balance` column is designated as the column on which delta conflict resolution is performed, and the `TS1` column is the invisible timestamp column to track the time of each change to the `Balance` column. A change is made to the `Balance` value in the row in both databases at nearly the same time (`@T20` in database A and `@T22` in database B). These changes result in a conflict, and delta conflict resolution is used to resolve the conflict in the following way:

- At database A, the value of `Balance` was changed from `100` to `110`. Therefore, the value was increased by 10.

- At database B, the value of `Balance` was changed from `100` to `120`. Therefore, the value was increased by 20.
- To resolve the conflict at database A, the value of the difference between the new and old values in the row LCR to the value in the table. The difference between the new and old values in the LCR is 20 (120–100=20). Therefore, the current value in the table (110) is increased by 20 so that the value after conflict resolution is 130.
- To resolve the conflict at database B, the value of the difference between the new and old values in the row LCR to the value in the table. The difference between the new and old values in the LCR is 10 (110–100=10). Therefore, the current value in the table (120) is increased by 10 so that the value after conflict resolution is 130.

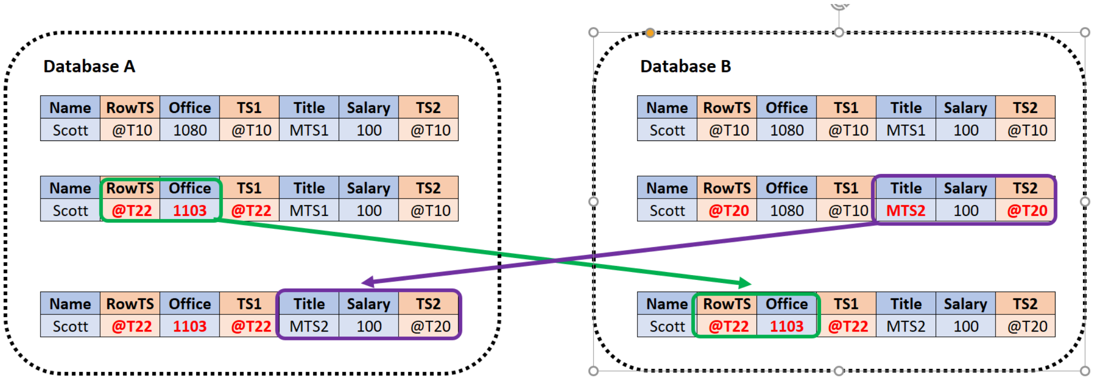
After delta conflict resolution, the value of the `Balance` column is the same for the row at database A and database B.

Column Groups

A column group is a logical grouping of one or more columns in a replicated table. When you add a column group, conflict detection and resolution is performed on the columns in the column group separately from the other columns in the table.

When you configure a table for Oracle GoldenGate conflict detection and resolution with the `ADD_AUTO_CDR` procedure, all of the scalar columns in the table are added to a default column group. To define other column groups for the table, run the `ADD_AUTO_CDR_COLUMN_GROUP` procedure. Any columns in the table that are not part of a user-defined column group remain in the default column group for the table.

Column groups enable different databases to update different columns in the same row at nearly the same time without causing a conflict. When column groups are configured for a table, conflicts can be avoided even if different databases update the same row in the table. A conflict is not detected if the updates change the values of columns in different column groups.


Database A
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T10 | 1080 | @T10 | MTS1 | 100 | @T10
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T22 | 1103 | @T22 | MTS1 | 100 | @T10
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T22 | 1103 | @T22 | MTS2 | 100 | @T20
Database B
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T10 | 1080 | @T10 | MTS1 | 100 | @T10
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T20 | 1080 | @T10 | MTS2 | 100 | @T20
Name | RowTS | Office | TS1 | Title | Salary | TS2
Scott | @T22 | 1103 | @T22 | MTS2 | 100 | @T20

This example shows a row being replicated at database A and database B. The following two column groups are configured for the replicated table at each database:

- One column group includes the `Office` column. The invisible timestamp column for this column group is `TS1`.
- Another column group includes the `Title` and `Salary` columns. The invisible timestamp column for this column group is `TS2`.

These column groups enable database A and database B to update the same row at nearly the same time without causing a conflict. Specifically, the following changes are made:

- At database A, the value of `Office` was changed from `1080` to `1030`.
- At database B, the value of `Title` was changed from `MTS1` to `MTS2`.

Because the `Office` column and the `Title` column are in different column groups, the changes are replicated without a conflict being detected. The result is that values in the row are same at both databases after each change has been replicated.

Piecewise LOB Updates

A set of lob operations composed of `LOB WRITE`, `LOB ERASE`, and `LOB TRIM` is a piecewise LOB update. When a table that contains LOB columns is configured for conflict detection and resolution, each LOB column is placed in its own column group, and the column group has its own hidden timestamp column. The timestamp column is updated on the first piecewise LOB operation.

For a LOB column, a conflict is detected and resolved in the following ways:

- If the timestamp for the LOB's column group is later than the corresponding LOB column group in the row, then the piecewise LOB update is applied.
- If the timestamp for the LOB's column group is earlier than the corresponding LOB column group in the row, then the LOB in the table row is retained.
- If the row does not exist in the table, then an error occurs

Configuring Automatic Conflict Detection and Resolution

You can configure Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the `DBMS_GOLDENGATE_ADM` package.

For the Replicat parameter file you need to add a `MAP` statement that includes the table to be replicated and the `MAPINVISIBLECOLUMNS` parameter.

Configuring Latest Timestamp Conflict Detection and Resolution

The `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package configures latest timestamp conflict detection and resolution. The `ADD_AUTO_CDR_COLUMN_GROUP` procedure adds optional column groups.

For Oracle Database 23c and higher, the `ADD_AUTO_CDR` procedure also provides the option to remove hidden columns. The `ADD_AUTO_CDR` procedure includes the flag `REMOVE_HIDDEN_COLUMNS` to drop tables or columns that are unused. To know more, see `ADD_AUTO_CDR Procedure` in the *Oracle Database PL/SQL Packages and Types Reference*

With latest timestamp conflict detection and resolution, a conflict is detected when the timestamp column of the row LCR does not match the timestamp of the corresponding table row. The row LCR is applied if its timestamp is later. Otherwise, the row LCR is discarded, and the table row is not changed. When you run the `ADD_AUTO_CDR` procedure, it adds an invisible timestamp column for each row in the specified table and configures timestamp conflict detection and resolution. When you use the `ADD_AUTO_CDR_COLUMN_GROUP` procedure to add one or more column groups, it adds a timestamp for the column group and configures timestamp conflict detection and resolution for the column group.

You can configure an Oracle GoldenGate administrator using the `GRANT_ADMIN_PRIVILEGE` procedure in the `DBMS_GOLDENGATE_ADM` package.

1. Connect to the inbound server database as a Oracle GoldenGate administrator.

2. Run the `ADD_AUTO_CDR` procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Run the `ADD_AUTO_CDR_COLUMN_GROUP` procedure and specify one or more column groups in the table.
4. Repeat the previous steps in each Oracle Database that replicates the table.

Example 8-9 Configuring the Latest Timestamp Conflict Detection and Resolution for a Table

This example configures latest timestamp conflict detection and resolution for the `hr.employees` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR(
    SCHEMA_NAME => 'HR',
    TABLE_NAME  => 'EMPLOYEES');
END;
/
```

Example 8-10 Configuring Column Groups

This example configures the following column groups for timestamp conflict resolution on the `HR.EMPLOYEES` table:

- The `JOB_IDENTIFIER_CG` column group includes the `JOB_ID`, `DEPARTMENT_ID`, and `MANAGER_ID` columns.
- The `COMPENSATION_CG` column group includes the `SALARY` and `COMMISSION_PCT` columns.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_COLUMN_GROUP(
    SCHEMA_NAME       => 'HR',
    TABLE_NAME        => 'EMPLOYEES',
    COLUMN_LIST       => 'JOB_ID, DEPARTMENT_ID, MANAGER_ID',
    COLUMN_GROUP_NAME => 'JOB_IDENTIFIER_CG');
END;
/

BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_COLUMN_GROUP(
    SCHEMA_NAME       => 'HR',
    TABLE_NAME        => 'EMPLOYEES',
    COLUMN_LIST       => 'SALARY, COMMISSION_PCT',
    COLUMN_GROUP_NAME => 'COMPENSATION_CG');
END;
/
```

Configuring Delta Conflict Detection and Resolution

The `ADD_AUTO_CDR_DELTA_RES` procedure in the `DBMS_GOLDENGATE_ADM` package configures delta conflict detection and resolution.

With delta conflict resolution, you specify one column for which conflicts are detected and resolved. The conflict is detected if the value of the column in the row LCR does not match the corresponding value in the table. The conflict is resolved by adding the difference between the new and old values in the row LCR to the value in the table.

You can configure an Oracle GoldenGate administrator using the `GRANT_ADMIN_PRIVILEGE` procedure in the `DBMS_GOLDENGATE_ADM` package.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `ADD_AUTO_CDR` procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Run the `ADD_AUTO_CDR_DELTA_RES` procedure and specify the column on which delta conflict detection and resolution is performed.
4. Repeat the previous steps in each Oracle Database that replicates the table.

Example 8-11   Configuring Delta Conflict Detection and Resolution for a Table

This example configures delta conflict detection and resolution for the `order_total` column in the `oe.orders` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR(
    SCHEMA_NAME => 'OE',
    TABLE_NAME  => 'ORDERS');
END;
/

BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_DELTA_RES(
    SCHEMA_NAME => 'OE',
    TABLE_NAME  => 'ORDERS',
    COLUMN_NAME => 'ORDER_TOTAL');
END;
/
```

Managing Automatic Conflict Detection and Resolution

You can manage Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the `DBMS_GOLDENGATE_ADM` package.

Altering Conflict Detection and Resolution for a Table

The `ALTER_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package alters conflict detection and resolution for a table.

Oracle GoldenGate automatic conflict detection and resolution must be configured for the table:

1. Connect to the inbound server database as the Oracle GoldenGate administrator.
2. Run the `ALTER_AUTO_CDR` procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-12 Altering Conflict Detection and Resolution for a Table

This example alters conflict detection and resolution for the `HR.EMPLOYEES` table to specify that delete conflicts are tracked in a tombstone table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR(
    SCHEMA_NAME       => 'HR',
    TABLE_NAME        => 'EMPLOYEES',
    TOMBSTONE_DELETES => TRUE);
END;
/
```

Altering a Column Group

The `ALTER_AUTO_CDR_COLUMN_GROUP` procedure alters a column group.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `ALTER_AUTO_CDR_COLUMN_GROUP` procedure and specify one or more column groups in the table.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-13 Altering a Column Group

This example removes the `MANAGER_ID` column from the `JOB_IDENTIFIER_CG` column group for the `HR.EMPLOYEES` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR_COLUMN_GROUP(
    SCHEMA_NAME        => 'HR',
    TABLE_NAME         => 'EMPLOYEES',
    COLUMN_GROUP_NAME  => 'JOB_IDENTIFIER_CG',
    REMOVE_COLUMN_LIST => 'MANAGER_ID');
END;
/
```

> **Note:**
>
> If there is more than one column, then use a comma-separated list.

Purging Tombstone Rows

The `PURGE_TOMBSTONES` procedure removes tombstone rows that were recorded before a specified date and time. This procedure removes the tombstone rows for all tables configured for conflict resolution in the database.

It might be necessary to purge tombstone rows periodically to keep the tombstone log from growing too large over time.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `PURGE_TOMBSTONES` procedure and specify the date and time.

Example 8-14 Purging Tombstone Rows

This example purges all tombstone rows recorded before 3:00 p.m. on December, 1, 2015 Eastern Standard Time. The timestamp must be entered in `TIMESTAMP WITH TIME ZONE` format.

```
EXEC DBMS_GOLDENGATE_ADM.PURGE_TOMBSTONES('2015-12-01 15:00:00.000000 EST');
```

Removing Conflict Detection and Resolution From a Table

With Oracle Database 23c and higher, removing Automatic Conflict Detection and Resolution (ACDR) entirely from the table has lesser impact on the table because the ACDR related columns are marked as `UNUSED`.

Use the `REMOVE_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package to tag a table as `UNUSED`, which minimizes blocking. You can choose to drop a column or retain it at a later stage.

See Removing a Column Group for examples.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `REMOVE_AUTO_CDR` procedure and specify the table.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-15 Removing Conflict Detection and Resolution for a Table

This example removes conflict detection and resolution for the `HR.EMPLOYEES` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR(
    SCHEMA_NAME => 'HR',
    TABLE_NAME  => 'EMPLOYEES');
END;
/
```

You can choose to drop columns by using the `ADD_AUTO_CDR.REMOVE_HIDDEN_COLUMNS` procedure to view and remove hidden columns.

Here is an example that you can use to view hidden columns in a table.

The following query uses the `DBA_UNUSED_COL_TABS` package to determine if there unused columns in the `EMPLOYEES` table.

```
SELECT OWNER, TABLE_NAME, COUNT
  FROM DBA_UNUSED_COL_TABS
  WHERE OWNER = 'HR'
  AND TABLE_NAME = 'EMPLOYEES'
  ORDER BY OWNER, TABLE_NAME;
```

The output displays as follows:

```
OWNER        TABLE_NAME    COUNT
--------    ------------ ----------
HR             EMPLOYEES      1
```

The following query lists out the hidden columns that were tagged by the system when ACDR was removed for the column group in the `EMPLOYEES` table.

```
SELECT OWNER, TABLE_NAME, COLUMN_ID, COLUMN_NAME, DATA_TYPE, HIDDEN_COLUMN
  FROM DBA_TAB_COLS
  WHERE OWNER = 'HR'
  AND TABLE_NAME = 'EMPLOYEES'
  AND HIDDEN_COLUMN = 'YES' AND USER_GENERATED= 'NO'
  ORDER BY OWNER, TABLE_NAME, COLUMN_ID;
```

The output displays as follows:

```
OWNER         TABLE_NAME    COLUMN_ID  COLUMN_NAME      DATA_TYPE
HIDDEN_COLUMN
------------ ------------ ---------- -------------    ------------
---------------
HR            EMPLOYEES    SYS_C00014_22092220:30:52$   TIMESTAMP(6)       YES
```

Removing a Column Group

With Oracle Database 23c and higher, removing Automatic Conflict Detection and Resolution (ACDR) from column groups has lesser impact on the table because the ACDR related columns are marked as `UNUSED`. You can choose to drop a column or retain it at a later stage.

Use the `REMOVE_AUTO_CDR_COLUMN_GROUP` procedure in the `DBMS_GOLDENGATE_ADM` package to tag a table, which minimizes blocking. See the example in Removing Conflict Detection and Resolution From a Table.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `REMOVE_AUTO_CDR_COLUMN_GROUP` procedure and specify the name of the column group.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-16 Removing a Column Group

This example removes the `COMPENSATION_CG` column group from the `HR.EMPLOYEES` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_COLUMN_GROUP(
    SCHEMA_NAME       => 'HR',
    TABLE_NAME        => 'EMPLOYEES',
    COLUMN_GROUP_NAME => 'COMPENSATION_CG');
END;
/
```

Removing Delta Conflict Detection and Resolution

The `REMOVE_AUTO_CDR_DELTA_RES` procedure in the `DBMS_GOLDENGATE_ADM` package removes delta conflict detection and resolution for a column.

Delta conflict detection and resolution must be configured for the specified column.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `REMOVE_AUTO_CDR_DELTA_RES` procedure and specify the column.

3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-17 Removing Delta Conflict Detection and Resolution for a Table

This example removes delta conflict detection and resolution for the `ORDER_TOTAL` column in the `OE.ORDERS` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_DELTA_RES(
    SCHEMA_NAME => 'OE',
    TABLE_NAME  => 'ORDERS',
    COLUMN_NAME => 'ORDER_TOTAL');
END;
/
```

Monitoring Automatic Conflict Detection and Resolution

You can monitor Oracle GoldenGate automatic conflict detection and resolution in an Oracle Database by querying data dictionary views.

Displaying Information About the Tables Configured for Conflicts

The `ALL_GG_AUTO_CDR_TABLES` view displays information about the tables configured for Oracle GoldenGate automatic conflict detection and resolution.

1. Connect to the database.
2. Query the `ALL_GG_AUTO_CDR_TABLES` view.

Example 8-18 Displaying Information About the Tables Configured for Conflict Detection and Resolution

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner for each table.
- The table name for each table.
- The tombstone table used to store rows deleted for update-delete conflicts, if a tombstone table is configured for the table.
- The hidden timestamp column used for conflict resolution for each table.

```
COLUMN TABLE_OWNER FORMAT A15
COLUMN TABLE_NAME FORMAT A15
COLUMN TOMBSTONE_TABLE FORMAT A15
COLUMN ROW_RESOLUTION_COLUMN FORMAT A25

SELECT TABLE_OWNER,
       TABLE_NAME,
       TOMBSTONE_TABLE,
       ROW_RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_TABLES
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

Your output looks similar to the following:

```
TABLE_OWNER     TABLE_NAME      TOMBSTONE_TABLE ROW_RESOLUTION_COLUMN
--------------- --------------- --------------- -------------------------
HR              EMPLOYEES       DT$_EMPLOYEES   CDRTS$ROW
OE              ORDERS          DT$_ORDERS      CDRTS$ROW
```

Displaying Information About Conflict Resolution Columns

The `ALL_GG_AUTO_CDR_COLUMNS` view displays information about the columns configured for Oracle GoldenGate automatic conflict detection and resolution.

The columns can be configured for row or column automatic conflict detection and resolution. The columns can be configured for latest timestamp conflict resolution in a column group. In addition, a column can be configured for delta conflict resolution.

1. Connect to the database as an Oracle GoldenGate administrator.
2. Query the `ALL_GG_AUTO_CDR_COLUMNS` view.

Example 8-19   Displaying Information About Column Groups

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner for each table.
- The table name for each table.
- If the column is in a column group, then the name of the column group.
- The column name.
- If the column is configured for latest timestamp conflict resolution, then the name of the hidden timestamp column for the column.

```
COLUMN TABLE_OWNER FORMAT A10
COLUMN TABLE_NAME FORMAT A10
COLUMN COLUMN_GROUP_NAME FORMAT A17
COLUMN COLUMN_NAME FORMAT A15
COLUMN RESOLUTION_COLUMN FORMAT A23

SELECT TABLE_OWNER,
       TABLE_NAME,
       COLUMN_GROUP_NAME,
       COLUMN_NAME,
       RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_COLUMNS
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

Your output looks similar to the following:

```
TABLE_OWNE TABLE_NAME COLUMN_GROUP_NAME COLUMN_NAME     RESOLUTION_COLUMN
---------- ---------- ----------------- ---------------
-----------------------
HR         EMPLOYEES  COMPENSATION_CG   COMMISSION_PCT  CDRTS$COMPENSATION_CG
HR         EMPLOYEES  COMPENSATION_CG   SALARY          CDRTS$COMPENSATION_CG
HR         EMPLOYEES  JOB_IDENTIFIER_CG MANAGER_ID
CDRTS$JOB_IDENTIFIER_CG
```

```
HR         EMPLOYEES  JOB_IDENTIFIER_CG JOB_ID
CDRTS$JOB_IDENTIFIER_CG
HR         EMPLOYEES  JOB_IDENTIFIER_CG DEPARTMENT_ID
CDRTS$JOB_IDENTIFIER_CG
HR         EMPLOYEES  IMPLICIT_COLUMNS$ PHONE_NUMBER    CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ LAST_NAME       CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ HIRE_DATE       CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ FIRST_NAME      CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ EMAIL           CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ EMPLOYEE_ID     CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_MODE      CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_ID        CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_DATE      CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ CUSTOMER_ID     CDRTS$ROW
OE         ORDERS     DELTA$            ORDER_TOTAL
OE         ORDERS     IMPLICIT_COLUMNS$ PROMOTION_ID    CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_STATUS    CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ SALES_REP_ID    CDRTS$ROW
```

In this example, the columns with `IMPLICIT_COLUMNS$` for the column group name are configured for row conflict detection and resolution, but they are not part of a column group. The columns with `DELTA$` for the column group name are configured for delta conflict detection and resolution, and these columns do not have a resolution column.

Displaying Information About Column Groups

The `ALL_GG_AUTO_CDR_COLUMN_GROUPS` view displays information about the column groups configured for Oracle GoldenGate automatic conflict detection and resolution.

You can configure Oracle GoldenGate automatic conflict detection and resolution using the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package. You can configure column groups using the `ADD_AUTO_CDR_COLUMN_GROUP` procedure in the `DBMS_GOLDENGATE_ADM` package.

1. Connect to the database as an Oracle GoldenGate administrator.
2. Query the `ALL_GG_AUTO_CDR_COLUMN_GROUPS` view.

Example 8-20 Displaying Information About Column Groups

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner.
- The table name.
- The name of the column group.
- The hidden timestamp column used for conflict resolution for each column group.

```
COLUMN TABLE_OWNER FORMAT A15
COLUMN TABLE_NAME FORMAT A15
COLUMN COLUMN_GROUP_NAME FORMAT A20
COLUMN RESOLUTION_COLUMN FORMAT A25

SELECT TABLE_OWNER,
       TABLE_NAME,
       COLUMN_GROUP_NAME,
       RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_COLUMN_GROUPS
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

The output looks similar to the following:

```
TABLE_OWNER     TABLE_NAME      COLUMN_GROUP_NAME    RESOLUTION_COLUMN
--------------- --------------- -------------------- -------------------------
HR              EMPLOYEES       COMPENSATION_CG      CDRTS$COMPENSATION_CG
HR              EMPLOYEES       JOB_IDENTIFIER_CG    CDRTS$JOB_IDENTIFIER_CG
```

Handling Processing Errors

This chapter describes how to configure the Oracle GoldenGate processes to handle errors. Oracle GoldenGate reports processing errors in several ways by means of its monitoring and reporting tools. For more information about these tools, see Monitoring Oracle GoldenGate Processing.

Overview of Oracle GoldenGate Error Handling

Oracle GoldenGate provides error-handling options for:

- Extract
- Replicat
- TCP/IP

Handling Extract Errors

There is no specific parameter to handle Extract errors when DML operations are being extracted, but Extract does provide a number of parameters that can be used to prevent anticipated problems. These parameters handle anomalies that can occur during the processing of DML operations, such as what to do when a row to be fetched cannot be located, or what to do when the transaction log is not available. The following is a partial list of these parameters.

- `FETCHOPTIONS`
- `WARNLONGTRANS`
- `DBOPTIONS`
- `TRANLOGOPTIONS`

To handle extraction errors that relate to DDL operations, use the `DDLERROR` parameter.

For a complete parameter list, see *Parameters and Functions Reference for Oracle GoldenGate*.

Handling Replicat Errors during DML Operations

To control the way that Replicat responds to an error during one of its DML statements, use the `REPERROR` parameter in the Replicat parameter file. You can use `REPERROR` as a global parameter or as part of a `MAP` statement. You can handle most errors in a default fashion (for example, to cease processing) with `DEFAULT` and `DEFAULT2` options, and also handle other errors in a specific manner.

The following comprise the range of `REPERROR` responses:

- `ABEND`: roll back the transaction and stop processing.
- `DISCARD`: log the error to the discard file and continue processing.
- `EXCEPTION`: send the error for exceptions processing. See Handling Errors as Exceptions for more information.
- `IGNORE`: ignore the error and continue processing.
- `RETRYOP [MAXRETRIES n]`: retry the operation, optionally up to a specific number of times.
- `TRANSABORT [, MAXRETRIES n] [, DELAY[C]SECS n]`: abort the transaction and reposition to the beginning, optionally up to a specific number of times at specific intervals.
- `RESET`: remove all previous `REPERROR` rules and restore the default of `ABEND`.
- `TRANSDISCARD`: discard the entire replicated source transaction if any operation within that transaction, including the commit, causes a Replicat error that is listed in the error specification. This option is useful when integrity constraint checking is disabled on the target.
- `TRANSEXCEPTION`: perform exceptions mapping for every record in the replicated source transaction, according to its exceptions-mapping statement, if any operation within that transaction (including the commit) causes a Replicat error that is listed in the error specification.

Most options operate on the individual record that generated an error, and Replicat processes the other, successful operations in the transaction. The exceptions are `TRANSDISCARD` and `TRANSEXCEPTION`: These options affect all records in a transaction if any record in that transaction generates an error. (The `ABEND` option also applies to the entire transaction, but does not apply error handling.)

See *Parameters and Functions Reference for Oracle GoldenGate* for `REPERROR` syntax and usage.

Handling Errors as Exceptions

When the action of `REPERROR` is `EXCEPTION` or `TRANSEXCEPTION`, you can map the values of operations that generate errors to an exceptions table and, optionally, map other information about the error that can be used to resolve the error. See About the Exceptions Table.

To map the exceptions to the exceptions table, use either of the following options of the `MAP` parameter:

- `MAP` with `EXCEPTIONSONLY`
- `MAP` with `MAPEXCEPTION`

Using `EXCEPTIONSONLY`

`EXCEPTIONSONLY` is valid for one pair of source and target tables that are explicitly named and mapped one-to-one in a `MAP` statement; that is, there cannot be wildcards. To use `EXCEPTIONSONLY`, create two `MAP` statements for each source table that you want to use `EXCEPTIONSONLY` for on the target:

- The first, a standard `MAP` statement, maps the source table to the actual target table.

- The second, an *exceptions MAP statement*, maps the source table to the *exceptions table* (instead of to the target table). An exceptions `MAP` statement executes immediately after an error on the source table to send the row values to the exceptions table.

  To identify a `MAP` statement as an exceptions `MAP` statement, use the `INSERTALLRECORDS` and `EXCEPTIONSONLY` options. The exceptions `MAP` statement must immediately follow the regular `MAP` statement that contains the same source table. Use a `COLMAP` clause in the exceptions `MAP` statement if the source and exceptions-table columns are not identical, or if you want to map additional information to extra columns in the exceptions table, such as information that is captured by means of column-conversion functions or `SQLEXEC`.

For more information about these parameters, see *Parameters and Functions Reference for Oracle GoldenGate*.

- A regular `MAP` statement that maps the source table `ggs.equip_account` to its target table `equip_account2`.
- An exceptions `MAP` statement that maps the same source table to the exceptions table `ggs.equip_account_exception`.

In this case, four extra columns were created, in addition to the same columns that the table itself contains:

```
DML_DATE
OPTYPE
DBERRNUM
DBERRMSG
```

To populate the `DML_DATE` column, the `@DATENOW` column-conversion function is used to get the date and time of the failed operation, and the result is mapped to the column. To populate the other extra columns, the `@GETENV` function is used to return the operation type, database error number, and database error message.

The `EXCEPTIONSONLY` option of the exceptions `MAP` statement causes the statement to execute only after a failed operation on the source table. It prevents every operation from being logged to the exceptions table.

The `INSERTALLRECORDS` parameter causes all failed operations for the specified source table, no matter what the operation type, to be logged to the exceptions table as *inserts*.

> **Note:**
>
> There can be no primary key or unique index restrictions on the exception table. Uniqueness violations are possible in this scenario and would generate errors.

Example 8-21 `EXCEPTIONSONLY`

This example shows how to use `REPERROR` with `EXCEPTIONSONLY` and an exceptions `MAP` statement. This example only shows the parameters that relate to `REPERROR`; other parameters not related to error handling are also required for Replicat.

```
REPERROR (DEFAULT, EXCEPTION)
MAP ggs.equip_account, TARGET ggs.equip_account2,
COLMAP (USEDEFAULTS);
MAP ggs.equip_account, TARGET ggs.equip_account_exception,
EXCEPTIONSONLY,
INSERTALLRECORDS
COLMAP (USEDEFAULTS,
DML_DATE = @DATENOW (),
OPTYPE = @GETENV ('LASTERR', 'OPTYPE'),
DBERRNUM = @GETENV ('LASTERR', 'DBERRNUM'),
DBERRMSG = @GETENV ('LASTERR', 'DBERRMSG'));
```

In this example, the `REPERROR` parameter is set for `DEFAULT` error handling, and the `EXCEPTION` option causes the Replicat process to treat failed operations as exceptions and continue processing.

Using MAPEXCEPTION

`MAPEXCEPTION` is valid when the names of the source and target tables in the `MAP` statement are wildcarded. Place the `MAPEXCEPTION` clause in the regular `MAP` statement, the same one where you map the source tables to the target tables. Replicat maps all operations that generate errors from all of the wildcarded tables to the same exceptions table; therefore, the exceptions table should contain a superset of all of the columns in all of the wildcarded tables.

Because you cannot individually map columns in a wildcard configuration, use the `COLMAP` clause with the `USEDEFAULTS` option to handle the column mapping for the wildcarded tables (or use the `COLMATCH` parameter if appropriate), and use explicit column mappings to map any additional information, such as that captured with column-conversion functions or `SQLEXEC`.

When using `MAPEXCEPTION`, include the `INSERTALLRECORDS` parameter in the `MAPEXCEPTION` clause. `INSERTALLRECORDS` causes all operation types to be applied to the exceptions table as `INSERT` operations. This is required to keep an accurate record of the exceptions and to prevent integrity errors on the exceptions table.

For more information about these parameters, see *Parameters and Functions Reference for Oracle GoldenGate*.

Example 8-22 MAPEXCEPTION

This is an example of how to use `MAPEXCEPTION` for exceptions mapping. The `MAP` and `TARGET` clauses contain wildcarded source and target table names. Exceptions that occur when processing any table with a name beginning with `TRX` are captured to the `fin.trxexceptions` table using the designated mapping.

```
MAP src.trx*, TARGET trg.*,
MAPEXCEPTION (TARGET fin.trxexceptions,
INSERTALLRECORDS,
COLMAP (USEDEFAULTS,
ACCT_NO = ACCT_NO,
OPTYPE = @GETENV ('LASTERR', 'OPTYPE'),
DBERR = @GETENV ('LASTERR', 'DBERRNUM'),
DBERRMSG = @GETENV ('LASTERR', 'DBERRMSG')
)
);
```

About the Exceptions Table

Use an exceptions table to capture information about an error that can be used for such purposes as troubleshooting your applications or configuring them to handle the error. At minimum, an exceptions table should contain enough columns to receive the entire row image from the failed operation. You can define extra columns to contain other information that is captured by means of column-conversion functions, `SQLEXEC`, or other external means.

To ensure that the trail record contains values for all of the columns that you map to the exceptions table, you can use either the `LOGALLSUPCOLS` parameter or the following parameters in the Extract parameter file:

- Use the `NOCOMPRESSDELETES` parameter so that all columns of a row are written to the trail for `DELETE` operations.
- Use the `GETUPDATEBEFORES` parameter so that Extract captures the before image of a row and writes them to the trail.

Handling Replicat errors during DDL Operations

To control the way that Replicat responds to an error that occurs for a DDL operation on the target, use the `DDLERROR` parameter in the Replicat parameter file. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Handling TCP/IP Errors

To provide instructions for responding to TCP/IP errors, use the `TCPERRS` file. This file is in the Oracle GoldenGate directory

Table 8-9 TCPERRS Columns

Column	Description
Error	Specifies a TCP/IP error for which you are defining a response.
Response	Controls whether or not Oracle GoldenGate tries to connect again after the defined error. Valid values are either `RETRY` or `ABEND`.
Delay	Controls how long Oracle GoldenGate waits before attempting to connect again.
Max Retries	Controls the number of times that Oracle GoldenGate attempts to connect again before aborting.

If a response is not explicitly defined in the `TCPERRS` file, Oracle GoldenGate responds to TCP/IP errors by abending.

Example 8-23 TCPERRS File

```
# TCP/IP error handling parameters
# Default error response is abend
#
# Error           Response   Delay(csecs)   Max Retries

ECONNABORTED      RETRY      1000           10
ECONNREFUSED      RETRY      1000           12
ECONNRESET        RETRY      500            10
ENETDOWN          RETRY      3000           50
ENETRESET         RETRY      1000           10
ENOBUFS           RETRY      100            60
ENOTCONN          RETRY      100            10
EPIPE             RETRY      500            10
ESHUTDOWN         RETRY      1000           10
ETIMEDOUT         RETRY      1000           10
NODYNPORTS        RETRY      100            10
```

The `TCPERRS` file contains default responses to basic errors. To alter the instructions or add instructions for new errors, open the file in a text editor and change any of the values in the columns shown in #unique_799/unique_799_Connect_42_G1028649:

Maintaining Updated Error Messages

The error, information, and warning messages that Oracle GoldenGate processes generate are stored in a data file named `ggmessage.dat` in the Oracle GoldenGate installation directory. The version of this file is checked upon process startup and must be identical to that of the process in order for the process to operate.

Resolving Oracle GoldenGate Errors

To get help with specific troubleshooting issues, go to My Oracle Support at `https://support.oracle.com` and search the Knowledge Base.

Trail File Management

The Extract process captures the changes from the transaction logs of the source system (database) into trail files that are consumed by other Oracle GoldenGate processes.

Extract can write into one or multiple sets of trail files. A trail is a sequence of files that are created and aged as needed. Processes that read a trail are:

- **Replicat**: Replicat reads the trail file received on the target deployment.
- **Distribution Service**: Extracts data from a local trail for further processing, if needed, and transfers it to the target system.
- **Receiver Service**: Receives the trail and transfers to Replicat, which reads the trail and applies change data to the target database.

You can create more than one trail to separate the data of different tables or applications, or to satisfy the requirements of a specific replication topology, such as a cascading topology. You link tables specified with a `TABLE` statement to a trail specified with an `EXTTRAIL` parameter statement in the Extract parameter file.

- Assign Storage for Oracle GoldenGate Trails
- Estimate Space for the Trail
- Add a Trail

Also see #unique_803.

Add the Local Trail

These steps add the local trail to which the primary Extract writes captured data.

On the source system, issue the `ADD EXTTRAIL` command on the command line:

```
ADD EXTTRAIL pathname, EXTRACT group name
```

Where:

- `EXTTRAIL` specifies that the trail is to be created on the local system.
- *`pathname`* is the relative or fully qualified name of the trail, including the two-character name.
- `EXTRACT` *`group name`* is the name of the primary Extract group.

> **Note:**
>
> Oracle GoldenGate creates this trail automatically during processing.

Example 8-24

```
ADD EXTTRAIL /north/ea, EXTRACT exte
```

Add the Remote Trail

Although it is read by Replicat, this trail must be associated with the Extract, so it must be added on the source system, not the target.

These steps add the remote trail:

On the source system, issue the following command:

```
ADD RMTTRAIL pathname, EXTRACT group name
```

Where:

- `RMTTRAIL` specifies that the trail is to be created on the target system.
- pathname is the relative or fully qualified name of the trail, including the two-character name.
- `EXTRACT` *`group name`* is the name of the data-pump Extract group.

> **Note:**
>
> Oracle GoldenGate creates this trail automatically during processing.

Example 8-25

```
ADD RMTTRAIL /south/re, EXTRACT exts
```

Encrypting the Extract and Replicat Passwords

It is strongly recommended that you encrypt the passwords of the user profiles that will be used for the primary and data pump Extracts, and for the Replicat process. The standard Oracle GoldenGate encryption method of AES (Advanced Encryption Standard) is supported by the IBM i platform. To encrypt the password, see Working with Runtime Parameters. It also contains information about how to encrypt data within disk storage and across TCP/IP.

> **Note:**
>
> The Oracle GoldenGate credential store is not supported by the iSeries platform.

Using Command Line Interfaces

To start either the Admin Client or GGSCI, you need to change the current working directory to the Oracle GoldenGate home directory (`OGG_HOME`).

> **Note:**
>
> The environment variable `OGG_HOME` and `OGG_VAR_HOME` must be set before starting the Admin Client or GGSCI.

Using Wildcards in Command Arguments

You can use wildcards with certain Oracle GoldenGate commands to control multiple Extract and Replicat groups as a unit. The wildcard symbol that is supported by Oracle GoldenGate is the asterisk (*). An asterisk represents any number of characters. For example, to start all Extract groups whose names contain the letter X, issue the following command.

```
START EXTRACT *X*
```

Globalization Support for the Command Interface

All command input and related console output are rendered in the default character set of the local operating system. To specify characters that are not compatible with the character set of the local operating system, use Unicode notation. For example, the following Unicode notation is equivalent to the name of a table that has the Euro symbol as its name:

```
ADD TRANDATA \u20AC1
```

For more information, see Support for Escape Sequences for more information about using Unicode notation.

> **Note:**
>
> Oracle GoldenGate group names are case-insensitive.

Using Command History

The execution of multiple commands is made easier with the following tools:

- Use the `HISTORY` command to display a list of previously executed commands.
- Use the `!` command to execute a previous command again without editing it.
- Use the `FC` command to edit a previous command and then execute it again.

Storing and Calling Frequently Used Command Sequences

You can automate a frequently-used series of commands by using an `OBEY` file and the `OBEY` command. The `OBEY` file takes the character set of the local operating system. To specify a character that is not compatible with that character set, use the Unicode notation.

See Support for Escape Sequences for more information about using Unicode notation.

To use OBEY

1. Create and save a text file that contains the commands, one command per line. This is your `OBEY` file. The name can be anything supported by the operating system. You can nest other `OBEY` files within an `OBEY` file.
2. Run the Admin Client or GGSCI.
3. (Optional) If using an `OBEY` file that contains nested `OBEY` files, issue the following command. This command enables the use of nested `OBEY` files for the current session and is required whenever using nested `OBEY` files. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

   ```
   ALLOWNESTED
   ```

4. Call the `OBEY` file by using the `OBEY` command from the command line interface (Admin Client or GGSCI).

   ```
   OBEY file_name
   ```

   Where:

   `file_name` is the relative or fully qualified name of the `OBEY` file.

Example 8-26 OBEY command file

```
ADD EXTRACT myext, TRANLOG, BEGIN now
START EXTRACT myext

ADD REPLICAT myrep, EXTTRAIL /ggs/dirdat/aa
START REPLICAT myrep

INFO EXTRACT myext, DETAIL
INFO REPLICAT myrep, DETAIL
```

The following example illustrates an `OBEY` command file for use with the `OBEY` command. It creates and starts Extract and Replicat groups and retrieves processing information.

See *Parameters and Functions Reference for Oracle GoldenGate* for more information about the `OBEY` command.

Getting Started with the Oracle GoldenGate Process Interfaces

This chapter describes how to use the Oracle GoldenGate GGSCI command line interface (CLI), batch, and shell scripts, and parameter files. Oracle GoldenGate Classic Architecture provides the GGSCI interface to work with Oracle GoldenGate processes.

Automating Commands

Oracle GoldenGate supports the issuing of commands through scripts or jobs. This section describes these options for UNIX- or Linux-based platforms and the IBMi platform.

On a UNIX or Linux system, or within a runtime environment that supports UNIX or Linux applications, you can issue Oracle GoldenGate commands from a script such as a startup script, shutdown script, or failover script by running GGSCI and calling an input file. The script file must be encoded in the operating system character set. Unicode notation can be used for characters that are not supported by the operating system character set. Before creating a script, see Globalization Support for the Command Interface.

To Input a Script

Use the following syntax from the command line of the operating system.

```
ggsci < input_file
```

Where:

- The angle bracket (<) character pipes the file into the GGSCI program.
- *input_file* is a text file, known as an `OBEY` file, containing the commands that you want to issue, in the order they are to be issued.

> **Note:**
>
> To stop the Manager process from a batch file, make certain to add the `!` argument to the end of the `STOP MANAGER` command. Otherwise, GGSCI issues a prompt that requires a response and causes the process to enter into a loop.

Issuing Commands Through the IBM i CLI

Oracle GoldenGate for IBM DB2 for i includes a set of native IBM i commands that enables the operation of the most common Oracle GoldenGate programs from the IBM i command-line interface (CLI). Because these commands are native, they do not need to be run from a PASE environment. With this support, it is possible to issue commands interactively or by using the typical job submission tools such as SBMJOB to operate Oracle GoldenGate non-interactively.

The commands are as follows and correspond to the Oracle GoldenGate programs of the same name. They reside in the Oracle GoldenGate installation library.

`DEFGEN`

`EXTRACT`

`KEYGEN`

`LOGDUMP`

`MGR`

`REPLICAT`

Specifying Object Names in Oracle GoldenGate Input

The following rules apply when specifying object names in parameter files (such as in `TABLE` and `MAP` statements), column-conversion functions, commands, and in other input.

Specifying Filesystem Path Names in Parameter Files on Windows Systems

On Windows systems, if the name of any directory in a filesystem path name begins with a number, the path must be specified with forward slashes, not backward slashes, when listing that path in Oracle GoldenGate input, such as parameter files or commands. This requirement prevents Oracle GoldenGate from interpreting the name as an octal escape sequence. For example, the following paths contain a directory named `\2014` that will be interpreted as the octal sequence `\201`:

```
C:\ogg\2014\install\dirdat\aa
C:\ogg\install\2014\dirdat\aa
```

The preceding path can be used with forward slashes as follows:

```
C:/ogg/2014/install/dirdat/aa
C:/ogg/install/2014/dirdat/aa
```

For more information, see Support for Escape Sequences.

Supported Database Object Names

Object names in parameter files, command, and other input can be any length and in any supported character set. For supported character sets, see Supported Character Sets.

Oracle GoldenGate supports most characters in object and column names. Specify object names in double quote marks if they contain special characters such as white spaces or symbols.

The following lists of supported and non-supported characters covers all databases supported by Oracle GoldenGate; a given database platform may or may not support all listed characters.

Supported Special Characters

Oracle GoldenGate supports all characters that are supported by the database, including the following special characters. Object names that contain these special characters must be enclosed within double quotes in parameter files.

Character	Description
/	Forward slash (See Specifying Names that Contain Slashes)
*	Asterisk (Must be escaped by a backward slash when used in parameter file, as in: `\*`)
?	Question mark (Must be escaped by a backward slash when used in parameter file, as in: `\?`)
@	At symbol (Supported, but is often used as a resource locator by databases. May cause problems in object names)
#	Pound symbol
$	Dollar symbol
%	Percent symbol (Must be `%%` when used in parameter file)
^	Caret symbol
( )	Open and close parentheses
_	Underscore
-	Dash

Character	Description
<space>	Space

Non-supported Special Characters

The following characters are not supported in object names and non-key column names.

Character	Description
\	Backward slash (Must be `\\` when used in parameter file)
{ }	Begin and end curly brackets (braces)
[ ]	Begin and end brackets
=	Equal symbol
+	Plus sign
!	Exclamation point
~	Tilde
|	Pipe
&	Ampersand
:	Colon
;	Semi-colon
,	Comma
' '	Single quotes
" "	Double quotes
'	Accent mark (Diacritical mark)
.	Period
<	Less-than symbol (or beginning angle bracket)
>	Greater-than symbol (or ending angle bracket)

Specifying Names that Contain Slashes

If a table name contains a forward-slash character (/) in any part of its name, that name component must be enclosed within double quotes unless the object name is from an IBM i platform . The following are some examples:

```
"c/d"
"/a".b
a."b/"
```

If the name contains a forward slash that is not enclosed within double quotes, Oracle GoldenGate treats it as a name that originated on the IBM i platform (from a DB2 for i database). The forward slash in the name is interpreted as a separator character.

Qualifying Database Object Names

Object names must be fully qualified in the parameter file. This means that every name specification must be qualified, not only those supplied as input to Oracle GoldenGate parameter syntax, but also names in a SQL procedure or query that is supplied as `SQLEXEC` input, names in user exit input, and all other input supplied in the parameter file.

Oracle GoldenGate supports two-part and three-part object names, as appropriate for the database.

Two-part Names

Most databases require only two-part names to be specified, in the following format:

`owner.object`

For example: `HR.EMP`

Where:

`owner` is a schema or database, depending on how the database defines a logical namespace that contains database objects. *`object`* is a table or other supported database object.

The databases for which Oracle GoldenGate supports two-part names are as follows, shown with their appropriate two-part naming convention:

- Db2 for i: *`schema.object`* and *`library/file(member)`*
- Db2 LUW: *`schema.object`*
- Db2 on z/OS: *`schema.object`*
- MySQL: *`database.object`*
- Oracle Database (non-CDB databases): *`schema.object`*
- SQL Server: *`schema.object`*
- Teradata: *`database.object`*

Three-part Names

Oracle GoldenGate supports three-part names for the following databases:

- Oracle container databases (CDB)

Three-part names are required to capture from a source Oracle container database because one Extract group can capture from more than one container. Thus, the name of the container, as well as the schema, must be specified for each object or objects in an Extract `TABLE` statement.

Specify a three-part Oracle CDB name as follows:

`container.schema.object`

For example: `PDBEAST.HR.EMP`

Applying Data from Multiple Containers or Catalogs

To apply data captured from multiple source containers or catalogs to a target Oracle container database, both three- and two-part names are required. In the `MAP` portion of the `MAP` statement, each source object must be associated with a container or catalog, just as it was in the `TABLE` statement. This enables you (and Replicat) to properly map data from multiple source containers or catalogs to the appropriate target objects. In the `TARGET` portion of the `MAP` statement, however, only two-part names are required. This is because Replicat can connect to only one target container or catalog at a time, and *`schema.owner`* is a sufficient qualifier. Multiple Replicat groups are required to support multiple target containers or catalogs. Specify the target container or catalog with the `TARGETDB` parameter.

Specifying a Default Container or Catalog

You can use the `SOURCECATALOG` parameter to specify a default catalog for any subsequent `TABLE`, `MAP`, (or Oracle `SEQUENCE`) specifications in the parameter file.

The following example shows the use of `SOURCECATALOG` to specify the default Oracle PDB named `pdbeast` for `region` and `jobs` objects, and the default PDB named `pdbwest` for `appraisal` objects. The objects in `pdbeast` are specified with a fully qualified three-part name, which does not require a default catalog to be specified.

```
TABLE pdbeast.hr.emp*;
SOURCECATALOG pdbeast
TABLE region.country*;
TABLE jobs.desg*;
SOURCECATALOG pdbwest
TABLE appraisal.sal*;
```

Specifying Case-Sensitive Database Object Names

Oracle GoldenGate supports case-sensitive names. Follow these rules when specifying case-sensitive objects.

- Specify object names from a case-sensitive database in the same case that is used to store them in the host database. Keep in mind that, in some database types, different levels of the database can have different case-sensitivity, such as case-sensitive schema but case-insensitive table. If the database requires quotes to enforce case-sensitivity, put quotes around each object that is case-sensitive in the qualified name.

  Correct: `TABLE "Sales"."ACCOUNT"`

  Incorrect: `TABLE "Sales.ACCOUNT"`
- Oracle GoldenGate converts case-insensitive names to the case in which they are stored when required for mapping purposes.

Table 8-10 provides an overview of the support for case-sensitivity in object names, per supported database. Refer to the database documentation for details on this type of support.

Table 8-10 Case Sensitivity of Object Names Per Database

Database	Requires quotes to enforce case-sensitivity?	Unquoted object name	Quoted object name
DB2	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in upper case	Case-sensitive, stores in mixed case
MySQL (Case-sensitive database)	No • Always case-sensitive, stores in mixed case • The names of columns, triggers, and procedures are case-insensitive	No effect	No effect

Table 8-10 (Cont.) Case Sensitivity of Object Names Per Database

Database	Requires quotes to enforce case-sensitivity?	Unquoted object name	Quoted object name
Oracle Database	Yes. Differentiates between case-sensitive and case-insensitive by use of quotes.	Case-insensitive, stores in upper case	Case-sensitive, stores in mixed case
SQL Server (Database created as case-sensitive)	No Always case-sensitive, stores in mixed case	No effect	No effect
SQL Server (Database created as case-insensitive)	No Always case-insensitive, stores in mixed case	No effect	No effect
Teradata	No Always case-insensitive, stores in mixed case	No effect	No effect

> **Note:**
>
> For all supported databases, passwords are always treated as case-sensitive regardless of whether the associated object name is quoted or unquoted.

Using Wildcards in Database Object Names

You can use wildcards for any part of a fully qualified object name, if supported for the specific database. These name parts can be the following: the container, database, or catalog name, the owner (schema or database name), and table or sequence name. For specifics on how object names and wildcards are supported, see the Oracle GoldenGate installation and configuration guide for that database.

Where appropriate, Oracle GoldenGate parameters permit the use of two wildcard types to specify multiple objects in one statement:

- A question mark (?) replaces one character. For example in a schema that contains tables named `TAB`*n*, where *n* is from 0 to 9, a wildcard specification of `HQ.TAB?` returns `HQ.TAB0`, `HQ.TAB1`, `HQ.TAB2`, and so on, up to `HQ.TAB9`, but no others. This wildcard is not supported for the DB2 LUW database nor for DEFGEN. This wildcard can only be used to specify source objects in a `TABLE` or `MAP` parameter. It cannot be used to specify target objects in the `TARGET` clause of `TABLE` or `MAP`.
- An asterisk (*) represents any number of characters (including zero sequence). For example, the specification of `HQ.T*` could return such objects as `HQ.TOTAL`, `HQ.T123`, and `HQ.T`. This wildcard is valid for all database types throughout all Oracle GoldenGate commands and parameters where a wildcard is allowed.
- In `TABLE` and `MAP` statements, you can combine the asterisk and question-mark wildcard characters in source object names only.

Rules for Using Wildcards for Source Objects

For source objects, you can use the asterisk alone or with a partial name. For example, the following source specifications are valid:

- `TABLE HQ.*;`
- `TABLE PDB*.HQ.*;`
- `MAP HQ.T_*;`
- `MAP HQ.T_*, TARGET HQ.*;`

The `TABLE`, `MAP` and `SEQUENCE` parameters take the case-sensitivity and locale of the database into account for wildcard resolution. For databases that are created as case-sensitive or case-insensitive, the wildcard matches the exact name and case. For example, if the database is case-sensitive, `SCHEMA.TABLE` is matched to `SCHEMA.TABLE`, `Schema.Table` is matched to `Schema.Table`, and so forth. If the database is case-insensitive, the matching is not case-sensitive.

For databases that can have both case-sensitive and case-insensitive object names in the same database instance, with the use of quote marks to enforce case-sensitivity, the wildcarding works differently. When used alone for a source name in a `TABLE` statement, an asterisk wildcard matches any character, whether or not the asterisk is within quotes. The following statements produce the same results:

```
TABLE hr.*;
TABLE hr."*";
```

Similarly, a question mark wildcard used alone matches any single character, whether or not it is within quotes. The following produce the same results:

```
TABLE hr.?;
TABLE hr."?";
```

If a question mark or asterisk wildcard is used with other characters, case-sensitivity is applied to the non-wildcard characters, but the wildcard matches both case-sensitive and case-insensitive names.

- The following `TABLE` statements capture any table name that begins with lower-case `abc`. The quoted name case is preserved and a case-sensitive match is applied. It captures table names that include `"abcA"` and `"abca"` because the wildcard matches both case-sensitive and case-insensitive characters.

  ```
  TABLE hr."abc*";
  TABLE hr."abc?";
  ```

- The following `TABLE` statements capture any table name that begins with upper-case `ABC`, because the partial name is case-insensitive (no quotes) and is stored in upper case by this database. However, because the wildcard matches both case-sensitive and case-insensitive characters, this example captures table names that include `ABCA` and `"ABCa"`.

  ```
  TABLE hr.abc*;
  TABLE hr.abc?;
  ```

Rules for Using Wildcards for Target Objects

When using wildcards in the `TARGET` clause of a `MAP` statement, the target objects must exist in the target database. (The exception is when DDL replication is being used, which allows new schemas and their objects to be replicated as they are created.)

For target objects, only an asterisk can be used. If an asterisk wildcard is used with a partial name, Replicat replaces the wildcard with the entire name of the corresponding source object. Therefore, specifications such as the following are *incorrect*:

```
TABLE HQ.T_*, TARGET RPT.T_*;
MAP HQ.T_*, TARGET RPT.T_*;
```

The preceding mappings produce incorrect results, because the wildcard in the target specification is replaced with `T_TEST` (the name of a source object), making the whole target name `T_T_TESTn`. The following illustrates the incorrect results:

- `HQ.T_TEST1` maps to `RPT.T_T_TEST1`
- `HQ.T_TEST2` maps to `RPT.T_T_TEST2`
- (The same pattern applies to all other `HQ.T_TESTn` mappings.)

The following examples show the correct use of asterisk wildcards.

```
MAP HQ.T_*, TARGET RPT.*;
```

The preceding example produces the following correct results:

- `HQ.T_TEST1` maps to `RPT.T_TEST1`
- `HQ.T_TEST2` maps to `RPT.T_TEST2`
- (The same pattern applies to all other `HQ.T_TESTn` mappings.)

Fallback Name Mapping

Oracle GoldenGate has a fallback mapping mechanism in the event that a source name cannot be mapped to a target name. If an exact match cannot be found on the target for a case-sensitive source object, Replicat tries to map the source name to the same name in upper or lower case (depending on the database type) on the target. Fallback name mapping is controlled by the `NAMEMATCH` parameters. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Wildcard Mapping from Pre-11.2.1 Trail Version

If Replicat is configured to read from a trail file that is a version prior to Oracle GoldenGate 11.2.1, the target mapping is made in the following manner to provide backward compatibility.

- Quoted object names are case-sensitive.
- Unquoted object names are case-insensitive.

The following maps a case-sensitive table name "`abc`" to target "`abc`". This only happens with a trail that was written by pre-11.2.1 Extract for SQL Server databases with a case-sensitive configuration. In this example, if the target database is Oracle Database or DB2 fallback name mapping is performed if the target database does not contain case-sensitive "`abc`" but does have table `ABC`.

```
MAP hq."abc", TARGET hq.*;
```

The following example maps a case-insensitive table name `abc` to target table name `ABC`. Previous releases of Oracle GoldenGate stored case-insensitive object names to the trail in upper case; thus the target table name is always upper cased. For case-insensitive name conversion, the comparison is in uppercase, A to Z characters only, in US-ASCII without taking locale into consideration.

```
MAP hq.abc, TARGET hq.*;
```

Asterisks or Question Marks as Literals in Object Names

If the name of an object itself includes an asterisk or a question mark, the entire name must be escaped and placed within double quotes, as in the following example:

```
TABLE HT."\?ABC";
```

How Wildcards are Resolved

By default, when an object name is wildcarded, the resolution for that object occurs when the first row from the source object is processed. (By contrast, when the name of an object is stated explicitly, its resolution occurs at process startup.) To change the rules for resolving wildcards, use the `WILDCARDRESOLVE` parameter. The default is `DYNAMIC`.

Excluding Objects from a Wildcard Specification

You can combine the use of wildcard object selection with explicit object exclusion by using the `EXCLUDEWILDCARDOBJECTSONLY`, `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `MAPEXCLUDE`, and `TABLEEXCLUDE` parameters.

Differentiating Case-Sensitive Column Names from Literals

By default, Oracle GoldenGate follows SQL-92 rules for specifying column names and literals. In Oracle GoldenGate parameter files, conversion functions, user exits, and commands, case-sensitive column names must be enclosed within double quotes if the database requires quotes around a name to support case-sensitivity. For example:

```
"columnA"
```

Case-sensitive column names in databases that do not require quotes to enforce case-sensitivity must be specified as they are stored in the database. For example:

```
ColumnA
```

Literals must be enclosed within single quotes. In the following example, `Product_Code` is a case-sensitive column name in an Oracle database, and the other strings are literals.

```
@CASE ("Product_Code", 'CAR', 'A car', 'TRUCK', 'A truck')
```

Performing Administrative Operations

This chapter contains instructions for making changes to applications, systems, and Oracle GoldenGate while the replication environment is active and processing data changes.

Shutting Down the System

When shutting down a system for maintenance and other procedures that affect Oracle GoldenGate, follow these steps to make certain that Extract has processed all of the transaction log records. Otherwise, you might lose synchronization data.

1. Stop all application and database activity that generates transactions that are processed by Oracle GoldenGate.
2. Run Admin Client and connect to the deployment using the `CONNECT` command.

3. Issue the `SEND EXTRACT` command with the `LOGEND` option. This command queries the Extract process to determine whether or not it is finished processing the records in the data source.

   ```
   SEND EXTRACT group LOGEND
   ```

4. Continue issuing the command until it returns a `YES` status. At that point, all transaction log data has been processed, and you can safely shut down Oracle GoldenGate and the system.

Changing Database Attributes

This section addresses administrative operations that are performed on database tables and structures.

Changing Database Metadata

This procedure is required to prevent Replicat errors when changing the following metadata of the source database:

- Database character set
- National character set
- Locale
- Timezone
- Object name case-sensitivity

If these changes are made without performing this procedure, the following error occurs:

```
2013-05-26 20:10:09  ERROR   OGG-05500  Detected database metadata mismatch
between current trail file ./dirdat/_p/v1000000003 and the previous sequence.
*DBTIMEZONE: [GMT]/[UTC].
```

This procedure stops Extract, and then creates a new trail file. The new database metadata is included in this new file with the transactions that started after the change.

1. Stop transaction activity on the source database. Do not make the metadata change to the database yet.
2. On the source system, issue the `SEND EXTRACT` command with the `LOGEND` option until it shows there is no more redo data to capture.

   ```
   SEND EXTRACT group LOGEND
   ```

3. Stop Extract.

   ```
   STOP EXTRACT group
   ```

4. On each target system, issue the `SEND REPLICAT` command with the `STATUS` option until it shows a status of "`At EOF`" to indicate that it finished processing all of the data in the trail. This must be done on all target systems until all Replicat processes return "`At EOF`."

   ```
   SEND REPLICAT group STATUS
   ```

5. Stop the data pumps and Replicat.

```
STOP EXTRACT group
STOP REPLICAT group
```

6. Change the database metadata.
7. On the source system, issue the `ALTER EXTRACT` command with the `ETROLLOVER` option for the primary Extract to roll over the local trail to the start of a new file.

```
ALTER EXTRACT group, ETROLLOVER
```

8. Start Extract.

```
START EXTRACT group
```

9. Reposition the Replicat processes to start at the new trail sequence number.

```
ALTER REPLICAT group, EXTSEQNO seqno, EXTRBA RBA
```

10. Start the data pumps.

```
START EXTRACT group
```

11. Start the Replicat processes.

```
START REPLICAT group
```

Adding Tables to the Oracle GoldenGate Configuration

This procedure assumes that the Oracle GoldenGate DDL support feature is not in use, or is not supported for, your database.

> **Note:**
>
> For Oracle and MySQL databases, you can enable the DDL support feature of Oracle GoldenGate to automatically capture and apply the DDL that adds new tables, instead of using this procedure. See the appropriate instructions for your database in this documentation.

Review these steps before starting. The process varies slightly, depending on whether or not the new tables satisfy wildcards in the `TABLE` parameter, and whether or not names or data definitions must be mapped on the target.

Prerequisites for Adding Tables to the Oracle GoldenGate Configuration

- This procedure assumes that the source and target tables are either empty or contain identical (already synchronized) data.
- You may be using the `DBLOGIN` and `ADD TRANDATA` commands. Before starting this procedure, see *Parameters and Functions Reference for Oracle GoldenGate* for the proper usage of these commands for your database.

To Add a Table to the Oracle GoldenGate Configuration

1. Stop user access to the new tables.
2. (*If new tables do not satisfy a wildcard*) If you are adding numerous tables that do not satisfy a wildcard, make a copy of the Extract and Replicat parameter files, and then add the new tables with `TABLE` and `MAP` statements. If you do not want to work with a copy, then edit the original parameter files after you are prompted to stop each process.
3. (*If new tables satisfy wildcards*) In the Extract and Replicat parameter files, make certain the `WILDCARDRESOLVE`parameter is not being used, unless it is set to the default of `DYNAMIC`.
4. (*If new tables do not satisfy a wildcard*) If the new tables *do not* satisfy a wildcard definition, stop Extract.

   ```
   STOP EXTRACT group
   ```

5. Add the new tables to the source and target databases.
6. If required for the source database, issue the `ADD TRANDATA` command in GGSCI for the new tables. Before using `ADD TRANDATA`, issue the `DBLOGIN` command.
7. Depending on whether the source and target definitins are identical or different, use either `ASSUMETARGETDEFS` or `SOURCEDEFS` in the Replicat parameter file. If `SOURCEDEFS` is needed, you can specify the template with the `DEF` option of the `MAP` parameter.
8. To register the new source definitions or new `MAP` statements, stop and then start Replicat.

   ```
   STOP REPLICAT group
   START REPLICAT group
   ```

9. Start Extract, if applicable.

   ```
   START EXTRACT group
   ```

10. Permit user access to the new tables.

Coordinating Table Attributes between Source and Target

Follow this procedure if you are changing an attribute of a source table that is in the Oracle GoldenGate configuration, such as adding or changing columns or partitions, or changing supplemental logging details (Oracle). It directs you how to make the same change to the target table without incurring replication latency.

> **Note:**
>
> See also Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables.

Note:

This procedure assumes that the Oracle GoldenGate DDL support feature is not in use, or is not supported for your database. For Oracle and MySQL databases, you can enable the DDL support feature of Oracle GoldenGate to propagate the DDL changes to the target, instead of using this procedure.

1. On the source and target systems, create a table, to be known as the *marker table*, that can be used for the purpose of generating a marker that denotes a stopping point in the transaction log. Just create two simple columns: one as a primary key and the other as a regular column. For example:

```
CREATE TABLE marker
(
id int NOT NULL,
column varchar(25) NOT NULL,
PRIMARY KEY (id)
);
```

2. Insert a row into the marker table on both the source and target systems.

```
INSERT INTO marker VALUES (1, 1);
COMMIT;
```

3. On the source system, run GGSCI.
4. Open the Extract parameter file for editing.

Caution:

Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

5. Add the marker table to the Extract parameter file in a `TABLE` statement.

```
TABLE marker;
```

6. Save and close the parameter file.
7. Add the marker table to the `TABLE` statement of the data pump, if one is being used.
8. Stop the Extract and data pump processes, and then restart them immediately to prevent capture lag.

```
STOP EXTRACT group
START EXTRACT group
STOP EXTRACT pump_group
START EXTRACT pump_group
```

9. On the target system, run GGSCI.
10. Open the Replicat parameter file for editing.

> **Caution:**
>
> Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted.

11. Add the marker table to the Replicat parameter file in a `MAP` statement, and use the `EVENTACTIONS` parameter as shown to stop Replicat and ignore operations on the marker table.

    ```
    MAP marker, TARGET marker, EVENTACTIONS (STOP, IGNORE);
    ```

12. Save and close the parameter file.
13. Stop, and then immediately restart, the Replicat process.

    ```
    STOP REPLICAT group
    START REPLICAT group
    ```

14. When you are ready to change the table attributes for both source and target tables, stop all user activity on them.
15. On the source system, perform an `UPDATE` operation to the marker table as the only operation in the transaction.

    ```
    UPDATE marker
    SET column=2,
    WHERE id=1;
    COMMIT;
    ```

16. On the target system, issue the following command until it shows that Replicat is stopped as a result of the `EVENTACTIONS` rule.

    ```
    STATUS REPLICAT group
    ```

17. Perform the DDL on the source and target tables, but do not yet allow user activity.
18. Start Replicat.

    ```
    START REPLICAT group
    ```

19. Allow user activity on the source and target tables.

Performing an ALTER TABLE to Add a Column on DB2 z/OS Tables

To add a fixed length column to a table that is in reordered row format and contains one or more variable length columns, one of the following will be required, depending on whether the table can be quiesced or not.

If the Table can be Quiesced

1. Allow Extract to finish capturing transactions that happened prior to the quiesce.
2. Alter the table to add the column.
3. Reorganize the tablespace.
4. Restart Extract.
5. Allow table activity to resume.

If the Table cannot be Quiesced

1. Stop Extract.
2. Remove the table from the `TABLE` statement in the parameter file.
3. Restart Extract.
4. Alter the table to add the column.
5. Reorganize the tablespace.
6. Stop Extract.
7. Add the table back to the `TABLE` statement.
8. Resynchronize the source and target tables.
9. Start Extract.
10. Allow table activity to resume.

Dropping and Recreating a Source Table

Dropping and recreating a source table requires caution when performed while Oracle GoldenGate is running.

1. Stop access to the table.
2. Allow Extract to process any remaining changes to that table from the transaction logs. To determine when Extract is finished, use the `INFO EXTRACT` command in GGSCI.

   ```
   INFO EXTRACT group
   ```

3. Stop Extract.

   ```
   STOP EXTRACT group
   ```

4. Drop and recreate the table.
5. If supported for this database, run the `ADD TRANDATA` command in GGSCI for the table.
6. If the recreate action changed the source table's definitions so that they are different from those of the target, run the DEFGEN utility for the source table to generate source definitions, and then replace the old definitions with the new definitions in the *existing* source definitions file on the target system.
7. Permit user access to the table.

Changing the ORACLE_SID

You can change the `ORACLE_SID` and `ORACLE_HOME` without having to change environment variables at the operating-system level. Depending on whether the change is for the source or target database, set the following parameters in the Extract or Replicat parameter files. Then, stop and restart Extract or Replicat for the parameters to take effect.

```
SETENV (ORACLE_HOME=location)
SETENV (ORACLE_SID='SID')
```

Purging Archive Logs

An Oracle archive log can be purged safely once Extract's read and write checkpoints are past the end of that log. Extract does not write a transaction to a trail until it has been committed, so

Extract must keep track of all open transactions. To do so, Extract requires access to the archive log where each open transaction started and all archive logs thereafter.

Extract reads the current archive log (the read checkpoint) for new transactions and also has a checkpoint (the recovery checkpoint) in the oldest archive log for which there is an uncommitted transaction.

Use the following command in GGSCI to determine Extract's checkpoint positions.

```
INFO EXTRACT group, SHOWCH
```

- The `Input Checkpoint` field shows where Extract began processing when it was started.
- The `Recovery Checkpoint` field shows the location of the oldest uncommitted transaction.
- The `Next Checkpoint` field shows the position in the redo log that Extract is reading.
- The `Output Checkpoint` field shows the position where Extract is writing.

You can write a shell script that purges all archive logs no longer needed by Extract by capturing the sequence number listed under the `Recovery Checkpoint` field. All archive logs prior to that one can be safely deleted.

Reorganizing a DB2 Table (z/OS Platform)

When using IBM's REORG utility to reorganize a DB2 table that has compressed tablespaces, specify the `KEEPDICTIONARY` option if the table is being processed by Oracle GoldenGate. This prevents the REORG utility from recreating the compression dictionary, which would cause log data that was written prior to the change not to be decompressed and cause Extract to terminate abnormally. As an alternative, ensure that all of the changes for the table have been extracted by Oracle GoldenGate before doing the reorganization, or else truncate the table.

Adding Process Groups to an Active Configuration

This section describes how to add process groups.

Before You Start

These instructions are for adding process groups to a configuration that is already active. The procedures should be performed by someone who has experience with Oracle GoldenGate. They involve stopping processes for a short period of time and reconfiguring parameter files. The person performing them must:

- Know the basic components of an Oracle GoldenGate configuration
- Understand Oracle GoldenGate parameters and commands
- Have access to GGSCI to create groups and parameter files
- Know which parameters to use in specific situations

Adding Another Extract Group to an Active Configuration

This procedure splits the workload of an existing Extract group into multiple Extract groups. It also provides instructions for including a data pump group (if applicable) and a Replicat group to propagate data that is captured by the new Extract group.

Steps are performed on the source and target systems.

1. Make certain the archived transaction logs are available in case the online logs recycle before you complete this procedure.
2. Choose a name for the new Extract group.
3. Decide whether or not to use a data pump.
4. On the source system, run GGSCI.
5. Create a parameter file for the new Extract group.

   ```
   EDIT PARAMS group
   ```

   > **Note:**
   >
   > You can copy the original parameter file to use for this group, but make certain to change the Extract group name and any other relevant parameters that apply to this new group.

6. In the parameter file, include:
   - `EXTRACT` parameter that specifies the new group.
   - Appropriate database login parameters.
   - Other appropriate Extract parameters for your configuration.
   - `EXTTRAIL` parameter that points to a local trail (if you will be adding a data pump) *or* a `RMTTRAIL` parameter (if you are not adding a data pump).
   - `RMTHOST` parameter if this Extract will write directly to a remote trail.
   - `TABLE` statement(s) (and `TABLEEXCLUDE`, if appropriate) for the tables that are to be processed by the new group.
7. Save and close the file.
8. Edit the original Extract parameter file(s) to remove the `TABLE` statements for the tables that are being moved to the new group or, if using wildcards, add the `TABLEEXCLUDE` parameter to exclude them from the wildcard specification.
9. (Oracle) If you are using Extract in integrated mode, register the new Extract group with the source database.

   ```
   REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])]
   ```

10. Lock the tables that were moved to the new group, and record the timestamp for the point when the locks were applied. For Oracle tables, you can run the following script, which also releases the lock after it is finished.

    ```
    -- temp_lock.sql
    -- use this script to temporary lock a table in order to
    -- get a timestamp

    lock table &schema . &table_name in EXCLUSIVE mode;
    SELECT TO_CHAR(sysdate,'MM/DD/YYYY HH24:MI:SS') "Date" FROM dual;
    commit;
    ```

11. Unlock the table(s) if you did not use the script in the previous step.
12. Stop the old Extract group(s) and any existing data pumps.

    ```
    STOP EXTRACT group
    ```

13. Add the new Extract group and configure it to start at the timestamp that you recorded.

```
ADD EXTRACT group, TRANLOG, BEGIN YYYY/MM/DD HH:MI:SS:CCCCCC
```

14. Add a trail for the new Extract group.

    ```
    ADD {EXTTRAIL | RMTTRAIL} trail, EXTRACT group
    ```

    Where:

    - `EXTTRAIL` creates a local trail. Use this option if you will be creating a data pump for use with the new Extract group. Specify the trail that is specified with `EXTTRAIL` in the parameter file. After creating the trail, go To Link a Local Data Pump to the New Extract Group .
    - `RMTTRAIL` creates a remote trail. Use this option if a data pump will not be used. Specify the trail that is specified with `RMTTRAIL` in the parameter file. After creating the trail, go To Link a Remote Replicat to the New Data Pump

    You can specify a relative or full path name. Examples:

    ```
    ADD RMTTRAIL dirdat/rt, EXTRACT primary
    ADD EXTTRAIL c:\ogg\dirdat\lt, EXTRACT primary
    ```

To Link a Local Data Pump to the New Extract Group

1. On the source system, add the data-pump Extract group using the `EXTTRAIL` trail as the data source.

   ```
   ADD EXTRACT pump, EXTTRAILSOURCE trail
   ```

   For example:

   ```
   ADD EXTRACT pump2, EXTTRAILSOURCE dirdat\lt
   ```

2. Create a parameter file for the data pump.

   ```
   EDIT PARAMS pump
   ```

3. In the parameter file, include the appropriate Extract parameters for your configuration, plus:
   - `RMTHOST` parameter to point to the target system.
   - `RMTTRAIL` parameter to point to a new remote trail (to be specified later).
   - `TABLE` parameter(s) for the tables that are to be processed by this data pump.
4. In GGSCI on the source system, add a remote trail for the data-pump. Use the trail name that you specified with `RMTTRAIL` in the parameter file.

   ```
   ADD RMTTRAIL trail, EXTRACT pump
   ```

   For example:

   ```
   ADD RMTTRAIL dirdat/rt, EXTRACT pump2
   ```

5. Follow the steps given below.

To Link a Remote Replicat to the New Data Pump

1. In GGSCI on the target system, add a Replicat group to read the remote trail. For `EXTTRAIL`, specify the same trail as in the `RMTTRAIL` Extract parameter and the `ADD RMTTRAIL` command.

   ```
   ADD REPLICAT group, EXTTRAIL trail
   ```

   For example:

```
ADD REPLICAT rep2, EXTTRAIL /home/ggs/dirdat/rt
```

2. Create a parameter file for this Replicat group. Use `MAP` statement(s) to specify the same tables that you specified for the new primary Extract and the data pump (if used).
3. On the source system, start the Extract groups and data pumps.

```
START EXTRACT group
START EXTRACT pump
```

4. On the target system, start the new Replicat group.

```
START REPLICAT group
```

Adding Another Data Pump to an Active Configuration

This procedure adds a data-pump Extract group to an active primary Extract group on the source system. It makes these changes:

- The primary Extract will write to a local trail.
- The data pump will write to a new remote trail after the data in the old trail is applied to the target.
- The old Replicat group will be replaced by a new one.

Steps are performed on the source and target systems.

1. On the source system, run GGSCI.
2. Add a local trail, using the name of the primary Extract group for *`group`*.

```
ADD EXTTRAIL trail, EXTRACT group
```

For example:

```
ADD EXTTRAIL dirdat\lt, EXTRACT primary
```

3. Open the parameter file of the primary Extract group, and replace the `RMTTRAIL` parameter with an `EXTTRAIL` parameter that points to the local trail that you created.

> **Caution:**
>
> Do not use the `VIEW PARAMS` or `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). View the parameter file from outside GGSCI if this is the case; otherwise, the contents may become corrupted..

Example `EXTTRAIL` parameter:

```
EXTTRAIL dirdat\lt
```

4. Remove the `RMTHOST` parameter.
5. Save and close the file.
6. Add a new data-pump Extract group, using the trail that you specified in step 2 as the data source.

```
ADD EXTRACT group, EXTTRAILSOURCE trail
```

For example:

```
ADD EXTRACT pump, EXTTRAILSOURCE dirdat\lt
```

7. Create a parameter file for the new data pump.

   ```
   EDIT PARAMS group
   ```

8. In the parameter file, include the appropriate Extract parameters for your configuration, plus:
   - `TABLE` parameter(s) for the tables that are to be processed by this data pump.
   - `RMTHOST` parameter to point to the target system.
   - `RMTTRAIL` parameter to point to a new remote trail (to be created later).

9. In GGSCI on the source system, add a remote trail for the data-pump. Use the trail name that is specified with `RMTTRAIL` in the data pump's parameter file, and specify the group name of the data pump for `EXTRACT`.

   ```
   ADD RMTTRAIL trail, EXTRACT group
   ```

   For example:

   ```
   ADD RMTTRAIL dirdat/rt, EXTRACT pump
   ```

   > **Note:**
   >
   > This command binds a trail name to an Extract group but does not actually create the trail. A trail file is created when processing starts.

10. On the target system, run GGSCI.

11. Add a new Replicat group and link it with the remote trail.

    ```
    ADD REPLICAT group, EXTTRAIL trail
    ```

    For example:

    ```
    ADD REPLICAT rep, EXTTRAIL dirdat/rt
    ```

12. Create a parameter file for this Replicat group. You can copy the parameter file from the original Replicat group, but make certain to change the `REPLICAT` parameter to the new group name.

13. On the source system, stop the primary Extract group, then start it again so that the parameter changes you made take effect.

    ```
    STOP EXTRACT group
    START EXTRACT group
    ```

14. On the source system, start the data pump.

    ```
    START EXTRACT group
    ```

15. On the target system, issue the `LAG REPLICAT` command for the old Replicat, and continue issuing it until it reports `At EOF, no more records to process.`

    ```
    LAG REPLICAT group
    ```

16. Stop the old Replicat group.

    ```
    STOP REPLICAT group
    ```

17. If using a checkpoint table for the old Replicat group, log into the database from GGSCI.

    ```
    DBLOGIN [SOURCEDB datasource] [{, USERIDALIAS alias | USERID user [,options]]
    ```

18. Delete the old Replicat group.

    ```
    DELETE REPLICAT group
    ```

19. Start the new Replicat group.

    ```
    START REPLICAT group
    ```

> **Note:**
>
> Do not delete the old remote trail, just in case it is needed later on for a support case or some other reason. You can move it to another location, if desired.

Adding Another Replicat Group to an Active Configuration

This procedure adds a new Replicat group to an existing Replicat group. The new Replicat reads from the same trail as the original Replicat.

Multiple Replicat groups may be required when Replicat is configured in classic mode, for the purpose of isolating transactions on certain tables or improving performance. Multiple Replicat groups usually are not required if using coordinated Replicat, because you can divide the workload among multiple processing threads within the same Replicat group. See Creating an Online Replicat Group for more information about Replicat modes.

Steps are performed on the source and target systems.

1. Choose a name for the new group.
2. On the target system, run GGSCI.
3. Create a parameter file for the new Replicat group.

    ```
    EDIT PARAMS group
    ```

> **Note:**
>
> You can copy the original parameter file to use for this group, but make certain to change the Replicat group name and any other relevant parameters that apply to this new group.

4. Add `MAP` statements (or edit copied ones) to specify the tables that you are adding or moving to this group. If this group will be a coordinated Replicat group, include the appropriate thread specifications.
5. Save and close the parameter file.
6. On the source system, run GGSCI.
7. Stop the Extract group.

    ```
    STOP EXTRACT group
    ```

8. Issue the `INFO REPLICAT` command for the old Replicat group, and continue issuing it until it reports `At EOF, no more records to process.`

```
INFO REPLICAT group
```

9. On the target system, edit the old Replicat parameter file to remove `MAP` statements that specified the tables that you moved to the new Replicat group. Keep only the `MAP` statements that this Replicat will continue to process.
10. Save and close the file.
11. Issue the `INFO REPLICAT` command for the old Replicat group, and continue issuing it until it reports `At EOF, no more records to process.`

    ```
    INFO REPLICAT group
    ```

12. Obtain the current Replicat checkpoint.

    ```
    INFO REPLICAT group
    ```

13. Stop the old Replicat group. If you are stopping a coordinated Replicat, make certain the stop is clean so that all threads stop at the same trail record.

    ```
    STOP REPLICAT group
    ```

14. Alter the new Replicat to position at the same trail sequence number and RBA as the old replicat group

    ```
    ALTER REPLICAT group, EXTSEQNO seqno, EXTRBA rba
    ```

    The *`seqno`* is the trail sequence number from the old group checkpoint and the *`rba`* is the trail record RBA number from the old group checkpoint.

15. Add the new Replicat group. For `EXTTRAIL`, specify the trail that this Replicat group is to read.

    ```
    ADD REPLICAT group, EXTTRAIL trail
    ```

    For example:

    ```
    ADD REPLICAT rep, EXTTRAIL dirdat/rt
    ```

16. Issue the `INFORM COMMAND` to alter the Replicat to the trail file sequence number and RBA displayed.

    ```
    INFORM COMMAND
    ```

17. On the source system, start the Extract group.

    ```
    START EXTRACT group
    ```

18. On the target system, start the old Replicat group.

    ```
    START REPLICAT group
    ```

19. Start the new Replicat group.

    ```
    START REPLICAT group
    ```

Changing the Size of Trail Files

You can change the size of trail files with the `MEGABYTES` option of either the `ALTER EXTTRAIL` or `ALTER RMTTRAIL` command, depending on whether the trail is local or remote. To change the file size, follow this procedure.

1. Issue one of the following commands, depending on the location of the trail, to view the path name of the trail you want to alter and the name of the associated Extract group. Use a wildcard to view all trails.

(Remote trail)

```
INFO RMTTRAIL *
```

(Local trail)

```
INFO EXTTRAIL *
```

2. Issue one of the following commands, depending on the location of the trail, to change the file size.

   (Remote trail)

   ```
   ALTER RMTTRAIL trail, EXTRACT group, MEGABYTES n
   ```

   (Local trail)

   ```
   ALTER EXTTRAIL trail, EXTRACT group, MEGABYTES n
   ```

3. Issue the following command to cause Extract to switch to the next file in the trail.

   ```
   SEND EXTRACT group, ROLLOVER
   ```

Switching Replicat from Non-Integrated Mode to Integrated Mode

Valid for Oracle only. For more information about Replicat modes for an Oracle database, see Choosing Capture and Apply Modes in .

This procedure switches an existing Replicat group from non-integrated to integrated mode.

> **Note:**
>
> Do not configure the switch between Replicat modes to occur immediately after Extract recovers from a failure or is repositioned to a different location in the transaction log.

1. Back up the Oracle GoldenGate working directories.
2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Replicat parameter file to a new name.
3. Grant the appropriate privileges to the Replicat user and perform the required configuration steps to support your business applications in integrated Replicat mode. See Assigning Credentials to Oracle GoldenGate in for information about configuring and running Replicat in integrated mode.
4. Run the command line (Admin Client).
5. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

6. Log into the target database from GGSCI.

   ```
   DBLOGIN USERIDALIAS alias
   ```

Where: *`alias`* is the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

7. Alter Replicat to integrated mode.

```
ALTER REPLICAT group, INTEGRATED
```

8. Replace the old parameter file with the new one, keeping the same name.
9. Start Replicat.

```
START REPLICAT group
```

10. Verify that Replicat is in integrated mode.

```
INFO REPLICAT group
```

When you start Replicat in integrated mode for the first time, the `START` command registers the Replicat group with the database and starts an inbound server to which Replicat attaches. When you convert a Replicat group to integrated mode, the use of the Oracle GoldenGate checkpoint table is discontinued and recovery information is maintained internally by the inbound server and by the checkpoint file going forward. You can retain the checkpoint table in the event that you decide to switch back to non-integrated mode.

Switching Replicat from Integrated Mode to Non-Integrated Mode

Valid for Oracle only. For more information about Replicat modes for an Oracle database, see About Integrated Replicat in .

You can, at any time, switch Replicat from integrated mode to non-integrated mode. This switch automatically unregisters the Replicat group from the target database, which removes the inbound server.

> **Note:**
>
> Do not configure the switch between Replicat modes to occur immediately after Extract recovers from a failure or is repositioned to a different location in the transaction log.

1. Back up the Oracle GoldenGate working directories.
2. While the Oracle GoldenGate processes continue to run in their current configuration, so that they keep up with current change activity, copy the Replicat parameter file to a new name.
3. Grant the appropriate privileges to the Replicat user and perform the required configuration steps to support your business applications in non-integrated Replicat mode. See Assigning Credentials to Oracle GoldenGatein for information about configuring and running Replicat in integrated mode.
4. Connect to the Oracle GoldenGate deployment as a system user.

5. Log into the target database from the command line.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   Where: *`alias`* is the alias of a user in the credential store who has the privileges granted through the Oracle `dbms_goldengate_auth.grant_admin_privilege` procedure.

6. Create a checkpoint table in the target database for the non-integrated Replicat to use to store its recovery checkpoints. If a checkpoint table was previously associated with this Replicat group and still exists, you can omit this step. See About Checkpoint Table for more information about options for using a checkpoint table.

   ```
   ADD CHECKPOINTTABLE [container.]table
   ```

7. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

8. Alter Replicat to non-integrated mode. For the `CHECKPOINTTABLE` argument, specify the checkpoint table that you created for this Replicat group.

   ```
   ALTER REPLICAT group, NONINTEGRATED, CHECKPOINTTABLE [container.]table
   ```

9. Replace the old parameter file with the new one, keeping the same name.

10. Start Replicat.

    ```
    START REPLICAT group
    ```

    After issuing this command, wait until there is some activity on the source database so that the switchover can be completed. (Replicat waits until its internal high-water mark is exceeded before removing the status of "switching from integrated mode.")

11. Verify that Replicat switched to non-integrated mode.

    ```
    INFO REPLICAT group
    ```

Switching Replicat to Coordinated Mode

Valid for all database types supported by Oracle GoldenGate.

This procedure upgrades a regular Replicat configuration (non-coordinated) to a coordinated configuration. This procedure assumes you are replacing a configuration that partitions data across multiple Extract and Replicat processes with a configuration that uses one Extract and one coordinated Replicat. The coordinated Replicat replaces the need for using multiple Replicat processes. A coordinated Replicat requires only one trail, so there is no need for multiple Extract processes or data pumps.

See Configuring Online Change Synchronization for more information about coordinated Replicat.

Procedure Overview

This procedure makes use of the `EVENTACTIONS` parameter with a `STOP` action, which enables all of the Replicat processes to stop at the same point in the trail. The `EVENTACTIONS` action is triggered by a transaction that contains an `INSERT` to a dummy table. The `INSERT` causes each process to finish processing everything up to, and including, the event transaction and then stop cleanly. An additional event action of `IGNORE` is specified for Replicat to prevent the multiple Replicat processes from attempting to insert the same record to the target. The result of this procedure is that all processes stop at the same point in the data stream: after completing the `INSERT` transaction to the dummy table.

After the processes stop, you move all of the `TABLE` statements to one primary Extract group. You move the same `TABLE` statements to the data pump that reads the trail of the Extract group that you retained. You move all of the `MAP` statements to a new coordinated Replicat group that reads the remote trail that is associated with the retained data pump. Once all of the `MAP` statements are in one parameter file, you edit them to add the thread specifications to support a coordinated Replicat. (This can be done ahead of time.) Then you drop the Replicat group and add it back in coordinated mode with the same name.

Performing the Switch to Coordinated Replicat

> **Note:**
>
> Do not create the Replicat group until prompted by these instructions.

1. Back up the current parameter files of all of the Extract groups, data pumps, and Replicat groups. You will be editing them.
2. Create a working directory outside the Oracle GoldenGate directory. You will use this directory to create and stage new versions of the parameter files. If needed, you can create a working directory on the source and target systems.
3. In the working directory, create a parameter file for a coordinated Replicat. Copy the `MAP` parameters from the active parameter files of all of the Replicat groups to this parameter file, and then add the thread specifications and any other parameters that support your required coordinated Replicat configuration
4. If using multiple primary Extract groups, select one to keep, and then save a copy of its current parameter file to the working directory.
5. Copy all of the `TABLE` statements from the other Extract groups to the new parameter file of the primary Extract that you are keeping.
6. In the working directory, save a copy of the parameter file of the data pump that is linked to the primary Extract that you are keeping.
7. Copy all of the `TABLE` statements from the other data pumps to the copied parameter file of the kept data pump.
8. In the source database, create a simple dummy table on which a simple `INSERT` statement can be performed. For this procedure, the name `schema.event` is used.
9. Create the same table on the target system, to avoid the need for additional configuration parameters.
10. Edit the active parameter files (not the copies) of all primary and data-pump Extract groups to add the following `EVENTACTIONS` parameter to each one.

    ```
    TABLE schema.event, EVENTACTIONS(STOP);
    ```

11. Edit the active parameter files (not the copies) of all of the Replicat groups to add the following `EVENTACTIONS` parameter to each one.

```
MAP schema.event, TARGET schema.event, EVENTACTIONS(IGNORE, STOP);
```

12. Stop the Oracle GoldenGate processes gracefully in the following order:
    - Stop all Replicat processes.
    - Stop all data pumps.
    - Stop all Extract processes.
13. Restart the Oracle GoldenGate processes in the following order so that the `EVENTACTIONS` parameters take effect:
    - Start all Extract processes.
    - Start all data pumps.
    - Start all Replicat processes.
14. On the source system, issue a transaction on the `schema.event` table that contains one `INSERT` statement. Make certain to commit the transaction.
15. In GGSCI, issue the `STATUS` command for all of the primary Extract and data pump processes on the source system, and issue the same command for all of the Replicat processes on the target system, until the commands show that all of the processes are `STOPPED`.

    ```
    STATUS EXTRACT *
    STATUS REPLICAT *
    ```

16. Replace the active parameter files of the primary Extract and data pump that you kept with the new parameter files from the working directory.
17. Delete the unneeded Extract and data pump groups and their parameter files.
18. Log into the target database by using the `DBLOGIN` command.
19. Delete all of the Replicat groups and their active parameter files.
20. Copy or move the new coordinated Replicat parameter file from the working directory to the Oracle GoldenGate directory.
21. In GGSCI, issue the `INFO EXTRACT` command for the data pump and make note of its write checkpoint position in the output (remote) trail.

    ```
    INFO EXTRACT pump, DETAIL
    ```

22. Add a new coordinated Replicat group with the following parameters.

    ```
    ADD REPLICAT group, EXTTRAIL trail, EXTSEQNO sequence_number, EXTRBA rba,
    COORDINATED MAXTHREADS number
    ```

    Where:

    - *`group`* is the name of the coordinated Replicat group. The name must match that of the new parameter file created for this group.
    - `EXTTRAIL` *`trail`* is the name of the trail that the data pump writes to.
    - `EXTSEQNO` *`sequence_number`* is the sequence number of the trail as shown in the write checkpoint returned by the `INFO EXTRACT` that you issued for the data pump.
    - `EXTRBA` *`rba`* is the relative byte address in the trail as shown in the write checkpoint returned by `INFO EXTRACT`. Together, these position Replicat to resume processing at the correct point in the trial.

- `MAXTHREADS` number specifies the maximum number of threads allowed for this group. This value should be appropriate for the number of threads that are specified in the parameter file.

23. Start the primary Extract group.
24. Start the data pump group.
25. Start the coordinated Replicat group.

Administering a Coordinated Replicat Configuration

This section contains instructions for coordinating threads and re-partitioning the workload among new or different threads. A coordinated Replicat should be stopped cleanly with the `STOP REPLICAT` command before making modifications to the partition specifications in `THREAD` or `THREADRANGE` clauses of the `MAP` statements. A clean stop ensures that all of the threads, which may be at different locations in the trail at any given point, all finish their work and arrive at a common trail location.

At startup, Replicat issues an error and abends if it detects that the last shutdown was not clean and the partitioning in the `MAP` statements was changed to contain a different number of threads (threads were added or removed). However, if the same threads are kept in the parameter file but simply rearranged among different `MAP` statements, Replicat issues a warning but does not abend. This can result in missing or duplicate records, because there is no way to ensure continuity of the thread-to-workload allocations from the previous run.

The following is an example of this condition.

Following is the original partitioning scheme:

```
MAP source, target, THREADRANGE(1-5);
MAP source1, target1, THREADRANGE(6-10);
```

The following re-partitioning of the original scheme produces only a warning:

```
MAP source, target, THREADRANGE(1-4);
MAP source1, target1, THREADRANGE(5-10);
```

This section provides instructions for cleanly shutting down Replicat before performing a re-partitioning, as well as instructions for attempting to recover Replicat continuity when a re-partitioning is performed after an unclean shutdown.

The following tasks can be performed for a Replicat group in coordinated mode.

Performing a Planned Re-partitioning of the Workload

A planned re-partitioning is when Replicat is allowed to shut down cleanly before it is started again with a new parameter file that contains updated thread partitioning. A clean shutdown enables all of the threads to arrive at a common checkpoint position in the trail. At that point, the new partitioning scheme can be applied in the next run. If Replicat does not shut down cleanly in this procedure, for example if there is an apply error, use the procedure in Synchronizing Threads After an Unclean Stop to re-synchronize the threads before you re-partition them.

1. Run GGSCI.
2. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

3. Open the parameter file for editing.

```
EDIT PARAMS group
```

4. Make the required changes to the `THREAD` or `THREADRANGE` specifications in the `MAP` statements.
5. Save and close the parameter file.
6. Start Replicat.

   ```
   START REPLICAT group
   ```

Recovering Replicat After an Unplanned Re-partitioning

An *unplanned re-partitioning* is when Replicat is not allowed to shut down cleanly before it is started again with a new parameter file that contains updated thread partitioning. In this scenario, some or all of the old threads were not able to finish their work and arrive at a common checkpoint. Upon restart, the coordinator thread attempts to apply the old partitioning scheme, and Replicat abends with an error. You can recover the coordinated Replicat group from this condition in one of the following ways:

- Use the auto-saved copy of the parameter file
- Reprocess from the low watermark with `HANDLECOLLISIONS`

Reprocessing From the Low Watermark with HANDLECOLLISIONS

In this procedure, you reposition all of the threads to the *low watermark* position. This is the earliest checkpoint position performed among all of the threads. To state it another way, the low watermark position is the last record processed by the slowest thread before the unclean stop. When you start Replicat, the threads reprocess the operations that they were processing before Replicat stopped, and the `HANDLECOLLISIONS` parameter handles any duplicate-record and missing-record errors that occur as the faster threads reprocess operations that they applied before the unclean stop.

1. Add the `HANDLECOLLISIONS` parameter to the Replicat parameter file. It is not necessary to use any `THREADS` options.
2. Issue the `INFO REPLICAT` command for the Replicat group as a whole (the coordinator thread). Make a record of the RBA of the checkpoint. This is the *low watermark* value. This output also shows you the active thread IDs under the `Group Name` column. Make a record of these, as well.

   ```
   INFO REPLICAT group

   GGSCI (slc03jgo) 3> info ra detailREPLICAT   RA       Last Started 2013-05-01
   14:15   Status ABENDEDCOORDINATED          Coordinator
   MAXTHREADS 15Checkpoint Lag       00:00:00 (updated 00:00:07 ago)Process
   ID           11445Log Read Checkpoint  File ./dirdat/withMaxTransOp/
   bg000000001                     2013-05-02 07:49:45.975662  RBA 44704Lowest Log BSN
   value: (requires database login)Active Threads: ID  Group Name PID   Status   Lag at
   Chkpt  Time Since Chkpt1   RA001     11454 ABENDED  00:00:00      00:00:01    2
   RA002     11455 ABENDED  00:00:00      00:00:04    3   RA003     11456 ABENDED
   00:00:00      00:00:01    5   RA005     11457 ABENDED  00:00:00      00:00:02    6
   RA006     11458 ABENDED  00:00:00      00:00:04    7   RA007     11459 ABENDED
   00:00:00      00:00:04
   ```

3. Issue the `INFO REPLICAT` command for each processing thread ID and record the RBA position of each thread. Make a note of the *highest* RBA. This is the *high watermark* of the Replicat group.

```
INFO REPLICAT threadID

 info ra002
REPLICAT   RA002    Last Started 2013-05-01 14:15   Status
ABENDEDCOORDINATED          Replicat Thread                  Thread 2Checkpoint
Lag       00:00:00 (updated 00:00:06 ago)Process ID           11455
Log Read Checkpoint  File ./dirdat/withMaxTransOp/bg000000001
2013-05-02 07:49:15.837271  RBA 45603
```

4. Issue the `ALTER REPLICAT` command for the coordinator thread (Replicat as a whole, without any thread ID) and position to the *low watermark* RBA that you recorded.

   ```
   ALTER REPLICAT group EXTRBA low_watermark_rba
   ```

5. Start Replicat.

   ```
   START REPLICAT group
   ```

6. Issue the basic `INFO REPLICAT` command until it shows an RBA that is higher than the *high watermark* that you recorded. `HANDLECOLLISIONS` handles any collisions that occur due to previously applied transactions.

   ```
   INFO REPLICAT group
   ```

7. Stop Replicat.

   ```
   STOP REPLICAT group
   ```

8. Remove or comment out the `HANDLECOLLISIONS` parameter.

9. Start Replicat.

   ```
   START REPLICAT group
   ```

Using the Auto-Saved Parameter File

A copy of the original parameter file is saved whenever the parameter file is edited before shutting down Replicat cleanly. You can revert to this parameter file and then resynchronize the threads so that they all catch up to the thread that had the most recent checkpoint position. Once the threads are synchronized, you can switch to the new parameter file and then start Replicat.

1. Save the new parameter file to a different name, and then rename the saved original parameter file to the correct name (same as the group name). The saved parameter file has a `.backup` suffix and is stored in the `dirprm` subdirectory of the Oracle GoldenGate installation directory.

2. Issue the following command to synchronize the Replicat threads to the maximum checkpoint position. This command automatically starts Replicat and executes the threads until they reach the maximum checkpoint position.

   ```
   SYNCHRONIZE REPLICAT group
   ```

3. Issue the `STATUS REPLICAT` command until it shows that Replicat stopped cleanly.

   ```
   STATUS REPLICAT group
   ```

4. Save the original parameter file to a different name, and then rename the new parameter file to the group name.

5. Start Replicat.

   ```
   START REPLICAT group
   ```

Synchronizing Threads After an Unclean Stop

When a Replicat group stops in an unclean manner, not all of the threads will reach a common checkpoint position in the trail. Unclean stops can be caused by issuing `STOP REPLICAT` with the `!` option, issuing the `KILL REPLICAT` command, or by transient errors related to Replicat, the database, or other local processes. You can restore the threads to the same position in the trail after an unclean stop and then start Replicat again from the correct checkpoint position.

In this procedure, the restore position is the *high watermark*. This is the most recent checkpoint position performed among all of the threads (the last record processed by the fastest thread before the unclean stop). Before starting Replicat, you can make changes to the parameter file, such as to repartition the workload among different or new threads. The repartitioning takes effect in a seamless manner after you start Replicat, because the threads can start from a synchronized state.

1. Run GGSCI.
2. Synchronize the Replicat threads to the maximum checkpoint position. Replicat performs the synchronization and then stops.

   ```
   SYNCHRONIZE REPLICAT group
   ```

3. (Optional) To re-partition the workload among different or new threads, open the parameter file for editing and then make the required changes to the `THREAD` or `THREADRANGE` specifications in the `MAP` statements.

   ```
   EDIT PARAMS group
   ```

4. Save and close the parameter file.
5. Start Replicat.

   ```
   START REPLICAT group
   ```

Restarting a Primary Extract after System Failure or Corruption

This procedure enables Oracle GoldenGate to recover from certain conditions, such as a file system corruption or a system failure, that corrupt the Extract checkpoint file, trail, or both, and which prevent Extract from being able to start. It enables you to establish a safe starting point in the transaction log for the primary Extract after the system has been restored. It also shows you how to reposition downstream data pumps and Replicat to read from the correct Extract write position in the trails, and to filter out any transactions that Replicat already applied to the target.

Details of This Procedure

Extract passes a *log begin sequence number*, or *LOGBSN*, to the trail files. The BSN is the native database sequence number that identifies the oldest uncommitted transaction that is held in Extract memory. For example, the BSN in an Oracle installation would be the Oracle system change number (SCN). Each trail file contains the lowest `LOGBSN` value for all of the transactions in that trail file. Once you know the `LOGBSN` value, you can reposition Extract at the correct read position to ensure that the appropriate transactions are re-generated to the trail and propagated to Replicat.

> **Note:**
>
> In an Oracle RAC environment, the lowest SCN of all of the threads is transmitted to Replicat. Transactions that may already have been committed by Replicat are handled as duplicates at startup. However, any thread that has been idle past a certain threshold will not be considered for the BSN value, to avoid Extract having to read too far back in the log stream when restarted.

The bounded recovery checkpoint is not taken into account when calculating the `LOGBSN`. The failure that affected the Extract checkpoint file may also involve a loss of the persisted bounded recovery data files and bounded recovery checkpoint information.

Performing the Recovery

Follow these steps in the order shown to recover the Oracle GoldenGate processes.

1. In GGSCI on the target system, issue the `DBLOGIN` command.

   ```
   DBLOGIN {USERID Replicat_user | USERIDALIAS alias_of_Replicat_user}
   ```

2. On the target, obtain the `LOGBSN` value by issuing the `INFO REPLICAT` command with the `DETAIL` option.

   ```
   INFO REPLICAT group, DETAIL
   ```

   The BSN is included in the output as a line similar to the following:

   ```
   Current Log BSN value: 1151679
   ```

3. Issue the following command in GGSCI to reposition the primary Extract to the `LOGBSN` start position.

   - (Integrated capture mode)

     ```
     ALTER EXTRACT group SCN 1151679
     ALTER EXTRACT group ETROLLOVER
     ```

   > **Note:**
   >
   > There is a limit on how far back Extract can go in the transaction stream, when in integrated mode. If the required SCN is no longer available, the `ALTER EXTRACT` command fails.

4. Issue the following command in GGSCI to the primary Extract to view the new sequence number of the Extract *`Write Checkpoint`*. This command shows the trail and RBA where Extract will begin to write new data. Because a rollover was issued, the start point is at the beginning (RBA 0) of the new trail file, in this example file number 7.

   ```
   INFO EXTRACT group SHOWCH
   Sequence #: 7
   RBA: 0
   ```

5. Issue the following command in GGSCI to reposition the downstream data pump and start a new output trail file.

```
ALTER EXTRACT pump EXTSEQNO 7
ALTER EXTRACT pump EXTRBA 0
ALTER EXTRACT pump ETROLLOVER
```

6. Issue the following command in GGSCI to the data pump Extract to view the new sequence number of the data pump Write Checkpoint, in this example trail number 9.

```
INFO EXTRACT pump SHOWCH
Sequence #: 9
RBA: 0
```

7. Reposition Replicat to start reading the trail at the new Write Checkpoint of the data pump.

```
ALTER REPLICAT group EXTSEQNO 9
ALTER REPLICAT group EXTRBA 0
```

8. Start the primary Extract and the data pump.

```
START EXTRACT group
START REPLICAT group
```

9. Issue the following command in GGSCI to start Replicat. If Replicat is operating in integrated mode (Oracle targets only), you do not need the `FILTERDUPTRANSACTIONS` option. Integrated Replicat handles duplicate transactions transparently.

```
START REPLICAT group[, FILTERDUPTRANSACTIONS]
```

Note:

The `LOGBSN` gives you the information needed to set Extract back in time to reprocess transactions. Some filtering by Replicat is necessary because Extract will likely re-generate a small amount of data that was already applied by Replicat. `FILTERDUPTRANSACTIONS` directs Replicat to find and filter duplicates at the beginning of the run.

Using Automatic Trail File Recovery

The trail recovery process has the ability to, in some cases, automatically rebuild trail files that are corrupt or missing by Oracle GoldenGate. When an Extract pump restarts, if the last trail that the pump was writing to is missing, then the Extract pump attempts to rebuild the missing trail file on the target system. This is done automatically using the checkpoint information for the process and the last valid trail file. The Replicat process automatically skips over any duplicate data in the trail files that have been rebuilt by the new trail recovery feature. This recovery will occur as long as there is at least 1 target trail from this sequence and that the trail files still exist on the source where the Extract pump is reading them.

This process can also be used to rebuild corrupt or invalid trail files on the target. Simply delete the corrupt trail file, and any trail files after that, and then restart the Extract pump. With this new behavior, Oracle recommends that `PURGEOLDEXTRACTS MINKEEP` rules are properly configured to ensure that there are trail files from the source that can be used to rebuild the target environment. This feature requires that Oracle GoldenGate release 12.1.2.1.8 or greater is used on both the source and target servers. Do *not* attempt to start the Replicat with `NOFILTERDUPTRANSACTIONS` because it will override Replicat's default behavior and may cause transactions that have already been applied to the target database to be applied again.

Customizing Oracle GoldenGate Processing

This chapter describes how to customize Oracle GoldenGate processing.

Executing Commands, Stored Procedures, and Queries with SQLEXEC

The `SQLEXEC` parameter of Oracle GoldenGate enables Extract and Replicat to communicate with the database to do the following:

- Execute a database command, stored procedure, or SQL query to perform a database function, return results (`SELECT` statements) or perform DML (`INSERT`, `UPDATE`, `DELETE`) operations.
- Retrieve output parameters from a procedure for input to a `FILTER` or `COLMAP` clause.

> **Note:**
>
> `SQLEXEC` provides minimal globalization support. To use `SQLEXEC` in the capture parameter file of the source capture, make sure that the client character set in the source `.prm` file is either the same or a superset of the source database character set.

Performing Processing with SQLEXEC

`SQLEXEC` extends the functionality of both Oracle GoldenGate and the database by allowing Oracle GoldenGate to use the native SQL of the database to execute custom processing instructions.

- Stored procedures and queries can be used to select or insert data into the database, to aggregate data, to denormalize or normalize data, or to perform any other function that requires database operations as input. Oracle GoldenGate supports stored procedures that accept input and those that produce output.
- Database commands can be issued to perform database functions required to facilitate Oracle GoldenGate processing, such as disabling triggers on target tables and then enabling them again.

Using SQLEXEC

The `SQLEXEC` parameter can be used as follows:

- as a clause of a `TABLE` or `MAP` statement
- as a standalone parameter at the root level of the Extract or Replicat parameter file.

Executing SQLEXEC within a TABLE or MAP Statement

When used within a `TABLE` or `MAP` statement, `SQLEXEC` can pass and accept parameters. It can be used for procedures and queries, but not for database commands.

Syntax

This syntax executes a procedure within a `TABLE` or `MAP` statement.

```
SQLEXEC (SPNAME sp_name,
[ID logical_name,]
{PARAMS param_spec | NOPARAMS})
```

Argument	Description
`SPNAME`	Required keyword that begins a clause to execute a stored procedure.
`sp_name`	Specifies the name of the stored procedure to execute.
`ID logical_name`	Defines a logical name for the procedure. Use this option to execute the procedure multiple times within a `TABLE` or `MAP` statement. Not required when executing a procedure only once.
`PARAMS param_spec | NOPARAMS`	Specifies whether or not the procedure accepts parameters. One of these options must be used (see Using Input and Output Parameters).

Syntax

This syntax executes a query within a `TABLE` or `MAP` statement.

```
SQLEXEC (ID logical_name, QUERY ' query ',
{PARAMS param_spec | NOPARAMS})
```

Argument	Description
`ID logical_name`	Defines a logical name for the query. A logical name is required in order to extract values from the query results. `ID logical_name` references the column values returned by the query.
`QUERY ' sql_query '`	Specifies the SQL query syntax to execute against the database. It can either return results with a `SELECT` statement or change the database with an `INSERT`, `UPDATE`, or `DELETE` statement. The query must be within single quotes and must be contained all on one line. Specify case-sensitive object names the way they are stored in the database, such as within quotes for Oracle case-sensitive names. `SQLEXEC 'SELECT "col1" from "schema"."table"'`
`PARAMS param_spec | NOPARAMS`	Defines whether or not the query accepts parameters. One of these options must be used (see Using Input and Output Parameters).

If you want to execute a query on a table residing on a different database than the current database, then the different database name has to be specified with the table. The delimiter between the database name and the tablename should be a colon (`:`). The following are some example use cases:

```
select col1 from db1:tab1
select col2 from db2:schema2.tab2
select col3 from tab3
select col3 from schema4.tab4
```

Executing SQLEXEC as a Standalone Statement

When used as a standalone parameter statement in the Extract or Replicat parameter file, `SQLEXEC` can execute a stored procedure, query, or database command. As such, it need not be tied to any specific table and can be used to perform general SQL operations. For example, if the Oracle GoldenGate database user account is configured to time-out when idle, you could use `SQLEXEC` to execute a query at a defined interval, so that Oracle GoldenGate does not appear idle. As another example, you could use `SQLEXEC` to issue an essential database command, such as to disable target triggers. A standalone `SQLEXEC` statement cannot accept input parameters or return output parameters.

Parameter syntax	Purpose
`SQLEXEC 'call procedure_name()'`	Execute a stored procedure
`SQLEXEC 'sql_query'`	Execute a query
`SQLEXEC 'database_command'`	Execute a database command

Argument	Description
`'call procedure_name ()'`	Specifies the name of a stored procedure to execute. The statement must be enclosed within single quotes. Example: `SQLEXEC 'call prc_job_count ()'`
`'sql_query'`	Specifies the name of a query to execute. The query must be contained all on one line and enclosed within single quotes. Specify case-sensitive object names the way they are stored in the database, such as within double quotes for Oracle object names that are case-sensitive. `SQLEXEC 'SELECT "col1" from "schema"."table"'`
`'database_command'`	Specifies a database command to execute. Must be a valid command for the database.

`SQLEXEC` provides options to control processing behavior, memory usage, and error handling. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Using Input and Output Parameters

Oracle GoldenGate provides options for passing input and output values to and from a procedure or query that is executed with `SQLEXEC` within a `TABLE` or `MAP` statement.

Passing Values to Input Parameters

To pass data values to input parameters within a stored procedure or query, use the `PARAMS` option of `SQLEXEC`.

Syntax

```
PARAMS ([OPTIONAL | REQUIRED] param = {source_column | function}
[, ...] )
```

Where:

- `OPTIONAL` indicates that a parameter value is not required for the SQL to execute. If a required source column is missing from the database operation, or if a column-conversion function cannot complete successfully because a source column is missing, the SQL executes anyway.
- `REQUIRED` indicates that a parameter value must be present. If the parameter value is not present, the SQL will not be executed.
- *`param`* is one of the following:
  - For a stored procedure, it is the name of any parameter in the procedure that can accept input, such as a column in a lookup table.
  - For an Oracle query, it is the name of any input parameter in the query excluding the leading colon. For example, `:param1` would be specified as `param1` in the `PARAMS` clause.
  - For a non-Oracle query, it is `p`*`n`*, where *`n`* is the number of the parameter within the statement, starting from 1. For example, in a query with two parameters, the *`param`* entries are `p1` and `p2`.
- `{`*`source_column`* | *`function`*} is the column or Oracle GoldenGate conversion function that provides input to the procedure.

Passing Values to Output Parameters

To pass values from a stored procedure or query as input to a `FILTER` or `COLMAP` clause, use the following syntax:

Syntax

```
{procedure_name | logical_name}.parameter
```

Where:

- *`procedure_name`* is the actual name of the stored procedure. Use this argument only if executing a procedure one time during the life of the current Oracle GoldenGate process.
- *`logical_name`* is the logical name specified with the `ID` option of `SQLEXEC`. Use this argument if executing a query or a stored procedure that will be executed multiple times.
- *`parameter`* is either the name of the parameter or `RETURN_VALUE`, if extracting returned values.

SQLEXEC Examples Using Parameters

These examples use stored procedures and queries with input and output parameters.

> **Note:**
>
> Additional `SQLEXEC` options are available for use when a procedure or query includes parametes. See the full `SQLEXEC` documentation in *Parameters and Functions Reference for Oracle GoldenGate*.

Example 8-27 SQLEXEC with a Stored Procedure

This example uses `SQLEXEC` to run a stored procedure named `LOOKUP` that performs a query to return a description based on a code. It then maps the results to a target column named `NEWACCT_VAL`.

```
CREATE OR REPLACE PROCEDURE LOOKUP
(CODE_PARAM IN VARCHAR2, DESC_PARAM OUT VARCHAR2)
BEGIN
    SELECT DESC_COL
    INTO DESC_PARAM
    FROM LOOKUP_TABLE
    WHERE CODE_COL = CODE_PARAM
END;
```

Contents of `MAP` statement:

```
MAP sales.account, TARGET sales.newacct, &
  SQLEXEC (SPNAME lookup, PARAMS (code_param = account_code)), &
    COLMAP (newacct_id = account_id, newacct_val = lookup.desc_param);
```

`SQLEXEC` executes the `LOOKUP` stored procedure. Within the `SQLEXEC` clause, the `PARAMS (code_param = account_code)` statement identifies `code_param` as the procedure parameter to accept input from the `account_code` column in the `account` table.

Replicat executes the `LOOKUP` stored procedure prior to executing the column map, so that the `COLMAP` clause can extract and map the results to the `newacct_val` column.

Example 8-28 SQLEXEC with a Query

This example implements the same logic as used in the previous example, but it executes a SQL query instead of a stored procedure and uses the `@GETVAL` function in the column map.

A query must be on one line. To split an Oracle GoldenGate parameter statement into multiple lines, an ampersand (`&`) line terminator is required.

Query for an Oracle database:

```
MAP sales.account, TARGET sales.newacct, &
SQLEXEC (ID lookup, &
QUERY 'select desc_col desc_param from lookup_table where code_col = :code_param', &
PARAMS (code_param = account_code)), &
COLMAP (newacct_id = account_id, newacct_val = &
@getval (lookup.desc_param));
```

Query for a non-Oracle database:

```
MAP sales.account, TARGET sales.newacct, &
SQLEXEC (ID lookup, &
QUERY 'select desc_col desc_param from lookup_table where code_col = ?', &
PARAMS (p1 = account_code)), &
COLMAP (newacct_id = account_id, newacct_val = &
@getval (lookup.desc_param));
```

Handling SQLEXEC Errors

There are two types of error conditions to consider when implementing `SQLEXEC`:

- The column map requires a column that is missing from the source database operation. This can occur for an update operation if the database only logs the values of columns that changed, rather than all of the column values. By default, when a required column is missing, or when an Oracle GoldenGate column-conversion function results in a "column missing" condition, the stored procedure does not execute. Subsequent attempts to extract an output parameter from the stored procedure results in a "column missing condition" in the `COLMAP` or `FILTER` clause.

- The database generates an error.

Handling Missing Column Values

Use the `@COLTEST` function to test the results of the parameter that was passed, and then map an alternative value for the column to compensate for missing values, if desired. Otherwise, to ensure that column values are available, you can use the `FETCHCOLS` or `FETCHCOLSEXCEPT` option of the `TABLE` parameter to fetch the values from the database if they are not present in the log. As an alternative to fetching columns, you can enable supplemental logging for those columns.

Handling Database Errors

Use the `ERROR` option in the `SQLEXEC` clause to direct Oracle GoldenGate to respond in one of the following ways:

Table 8-11 ERROR Options

Action	Description
`IGNORE`	Causes Oracle GoldenGate to ignore all errors associated with the stored procedure or query and continue processing. Any resulting parameter extraction results in a "column missing" condition. This is the default.
`REPORT`	Ensures that all errors associated with the stored procedure or query are reported to the discard file. The report is useful for tracing the cause of the error. It includes both an error description and the value of the parameters passed to and from the procedure or query. Oracle GoldenGate continues processing after reporting the error.
`RAISE`	Handles errors according to rules set by a `REPERROR` parameter specified in the Replicat parameter file. Oracle GoldenGate continues processing other stored procedures or queries associated with the current `TABLE` or `MAP` statement before processing the error.
`FINAL`	Performs in a similar way to `RAISE` except that when an error associated with a procedure or query is encountered, any remaining stored procedures and queries are bypassed. Error processing is called immediately after the error.
`FATAL`	Causes Oracle GoldenGate to abend immediately upon encountering an error associated with a procedure or query.

Additional SQLEXEC Guidelines

Observe the following `SQLEXEC` guidelines:

- Up to 20 stored procedures or queries can be executed per `TABLE` or `MAP` entry. They execute in the order listed in the parameter statement.
- A database login by the Oracle GoldenGate user must precede the `SQLEXEC` clause. Use the `SOURCEDB` and/or `USERID` or `USERIDALIAS` parameter in the Extract parameter file or the `TARGETDB` and/or `USERID` or `USERIDALIAS` parameter in the Replicat parameter file, as needed for the database type and configured authentication method.

- The SQL is executed by the Oracle GoldenGate user. This user must have the privilege to execute stored procedures and call RDBM-supplied procedures.
- Database operations within a stored procedure or query are committed in same context as the original transaction.
- Do not use `SQLEXEC` to update the value of a primary key column. If `SQLEXEC` is used to update the value of a key column, then the Replicat process will not be able to perform a subsequent update or delete operation, because the original key value will be unavailable. If a key value must be changed, you can map the original key value to another column and then specify that column with the `KEYCOLS` option of the `TABLE` or `MAP` parameter.
- For DB2, Oracle GoldenGate uses the ODBC `SQLExecDirect` function to execute a SQL statement dynamically. This means that the connected database server must be able to prepare the statement dynamically. ODBC prepares the SQL statement every time it is executed (at the requested interval). Typically, this does not present a problem to Oracle GoldenGate users. See the IBM DB2 documentation for more information.
- Do not use `SQLEXEC` for objects being processing by a data-pump Extract in pass-through mode.
- All object names in a `SQLEXEC` statement must be fully qualified with their two-part or three-part names, as appropriate for the database.
- All objects that are affected by a `SQLEXEC` stored procedure or query must exist with the correct structures prior to the execution of the SQL. Consequently, DDL on these objects that affects structure (such as `CREATE` or `ALTER`) must happen before the `SQLEXEC` executes.
- All objects affected by a standalone `SQLEXEC` statement must exist before the Oracle GoldenGate processes start. Because of this, DDL support must be disabled for those objects; otherwise, DDL operations could change the structure or delete the object before the `SQLEXEC` procedure or query executes on it.

Using Oracle GoldenGate Macros to Simplify and Automate Work

You can use Oracle GoldenGate macros in parameter files to configure and reuse parameters, commands, and conversion functions. reducing the amount of text you must enter to do common tasks. A macro is a built-in automation tool that enables you to call a stored set of processing steps from within the Oracle GoldenGate parameter file. A macro can consist of a simple set of frequently used parameter statements to a complex series of parameter substitutions, calculations, or conversions. You can call other macros from a macro. You can store commonly used macros in a library, and then call the library rather than call the macros individually.

Oracle GoldenGate macros work with the following parameter files:

- DEFGEN
- Extract
- Replicat

Do not use macros to manipulate data for tables that are being processed by a data-pump Extract in pass-through mode.

There are two steps to using macros:

Defining a Macro

Calling a Macro

Defining a Macro

To define an Oracle GoldenGate macro, use the `MACRO` parameter in the parameter file. `MACRO` defines any input parameters that are needed and it defines the work that the macro performs.

Syntax

```
MACRO #macro_name
PARAMS (#p1, #p2 [, ...])
BEGIN
macro_body
END;
```

Table 8-12 Macro Definition Arguments

Argument	Description
`MACRO`	Required. Indicates the start of an Oracle GoldenGate macro definition.
`#macro_name`	The name of the macro. Macro and parameter names must begin with a macro character. The default macro character is the pound (#) character, as in `#macro1` and `#param1`. A macro or parameter name can be one word consisting of letters and numbers, or both. Special characters, such as the underscore character (_) or hyphen `(-)`, can be used. Some examples of macro names are: `#mymacro`, `#macro1`, `#macro_1`, `#macro-1`, `#macro$`. Some examples of parameter names are `#sourcecol`, `#s`, `#col1`, and `#col_1`. To avoid parsing errors, the macro character cannot be used as the first character of a macro name. For example, `##macro` is invalid. If needed, you can change the macro character by using the `MACROCHAR` parameter. See Reference for Oracle GoldenGate for Windows and UNIX. Macro and parameter names are not case-sensitive. Macro or parameter names within quotation marks are ignored.
`PARAMS (#p1, #p2)`	Optional definition of input parameters. Specify a comma-separated list of parameter names and enclose it within parentheses. Each parameter must be referenced in the macro body where you want input values to be substituted. You can list each parameter on a separate line to improve readability (making certain to use the open and close parentheses to enclose the parameter list). See Calling a Macro that Contains Parameters for more information.
`BEGIN`	Begins the macro body. Must be specified before the macro body.

Table 8-12 (Cont.) Macro Definition Arguments

Argument	Description

`macro_body`	The macro body. The body is a syntax statement that defines the function that is to be performed by the macro. A macro body can include any of the following types of statements.
	• Simple parameter statements, as in:
	`COL1 = COL2`
	• Complex parameter statements with parameter substitution as in:
	`MAP #o.#t, TARGET #o.#t, KEYCOLS (#k), COLMAP (USEDEFAULTS);`
	• Invocations of other macros, as in:
	`#colmap (COL1, #sourcecol)`

`END;`	Ends the macro definition. The semicolon is required to complete the definition.

The following is an example of a macro definition that includes parameters. In this case, the macro simplifies the task of object and column mapping by supplying the base syntax of the `MAP` statement with input parameters that resolve to the names of the owners, the tables, and the `KEYCOLS` columns.

```
MACRO #macro1
PARAMS ( #o, #t, #k )
BEGIN
MAP #o.#t, TARGET #o.#t, KEYCOLS (#k), COLMAP (USEDEFAULTS);
END;
```

The following is an example of a macro that does not define parameters. It executes a frequently used set of parameters.

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

Calling a Macro

This section shows you how to call a macro. (To define a macro, see Defining a Macro).

To call a macro, use the following syntax where you want the macro to run within the parameter file.

Syntax

```
[target =] macro_name (val[, ...])

[target =] macro_name (val | {val, val, ...}[, ...])
```

Table 8-13 Syntax Elements for Calling a Macro

Argument	Description

`target` `=`	Optional. Specifies the target to which the results of the macro are assigned or mapped. For example, `target` can be used to specify a target column in a `COLMAP` statement. In the following call to the `#make_date` macro, the column `DATECOL1` is the target and will be mapped to the macro results.
	`DATECOL1 = #make_date (YR1, MO1, DAY1)`
	Without a target, the syntax to call `#make_date` is:
	`#make_date (YR1, MO1, DAY1)`

`macro_name`	The name of the macro that is being called, for example: `#make_date`.
`(` *`val`*`[, ...])`	The parameter input values. This component is required whether or not the macro defines parameters. If the macro defines parameters, specify a comma-separated list of input values, in the order that corresponds to the parameter definitions in the `MACRO` parameter, and enclose the list within parentheses. If the macro does not define parameters, specify the open and close parentheses with nothing between them (). For more information about this syntax, see the following: Calling a Macro that Contains Parameters. Calling a Macro without Input Parameters.
`(` *`val`* `| {`*`val`*`,` *`val`*`, ...} )[, ...]`	The parameter input values. This component is required whether or not the macro defines parameters. If the macro defines parameters, specify a comma-separated list of input values, in the order that corresponds to the parameter definitions in the `MACRO` parameter, and enclose the list within parentheses. To pass multiple values to one parameter, separate them with commas and enclose the list within curly brackets. If the macro does not define parameters, specify the open and close parentheses with nothing between them (). For more information about this syntax, see the following: Calling a Macro that Contains Parameters. Calling a Macro without Input Parameters.

Calling a Macro that Contains Parameters

To call a macro that contains parameters, the call statement must supply the input values that are to be substituted for those parameters when the macro runs. See the syntax in Table 8-13.

Valid input for a macro parameter is any of the following, preceded by the macro character (default is #):

- A single value in plain or quoted text, such as: `#macro (#name, #address, #phone)` or `#macro (#"name", #"address", #"phone")`.
- A comma-separated list of values enclosed within curly brackets, such as: `#macro1 (SCOTT, DEPT, {DEPTNO1, DEPTNO2, DEPTNO3})`. The ability to substitute a block of values for any given parameter add flexibility to the macro definition and its usability in the Oracle GoldenGate configuration.
- Calls to other macros, such as: `#macro (#mycalc (col2, 100), #total)`. In this example, the `#mycalc` macro is called with the input values of `col2` and `100`.

Oracle GoldenGate substitutes parameter values within the macro body according to the following rules.

1. The macro processor reads through the macro body looking for instances of parameter names specified in the `PARAMS` statement.
2. For each occurrence of the parameter name, the corresponding parameter value specified during the call is substituted.
3. If a parameter name does not appear in the `PARAMS` statement, the macro processor evaluates whether or not the item is, instead, a call to another macro. (See Calling Other Macros from a Macro.) If the call succeeds, the nested macro is executed. If it fails, the whole macro fails.

Example 8-29   Using Parameters to Populate a MAP Statement

The following macro definition specifies three parameter that must be resolved. The parameters substitute for the names of the table owner (parameter `#o`), the table (parameter `#t`), and the `KEYCOLS` columns (parameter `#k`) in a `MAP` statement.

```
MACRO #macro1  PARAMS ( #o, #t, #k )  BEGIN  MAP #o.#t, TARGET #o.#t, KEYCOLS (#k),
COLMAP (USEDEFAULTS); END;
```

Assuming a table in the `MAP` statement requires only one `KEYCOLS` column, the following syntax can be used to call `#macro1`. In this syntax, the `#k` parameter can be resolved with only one value.

```
#macro1 (SCOTT, DEPT, DEPTNO1)
```

To call the macro for a table that requires two `KEYCOLS` columns, the curly brackets are used as follows to enclose both of the required values for the column names:

```
#macro1 (SCOTT, DEPT, {DEPTNO1, DEPTNO2})
```

The `DEPTNO1` and `DEPTNO2` values are passed as one argument to resolve the `#t` parameter. Tables with three or more `KEYCOLS` can also be handled in this manner, using additional values inside the curly brackets.

Example 8-30   Using a Macro to Perform Conversion

In this example, a macro defines the parameters `#year`, `#month`, and `#day` to convert a proprietary date format.

```
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month, 'DD',
#day)
END;
```

The macro is called in the `COLMAP` clause:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcol1 = sourcecol1,
datecol1 = #make_date(YR1, MO1, DAY1),
datecol2 = #make_date(YR2, MO2, DAY2)
);
```

The macro expands as follows:

```
MAP sales.acct_tab, TARGET sales.account,
COLMAP
(
targcol1 = sourcecol1,
datecol1 = @DATE ('YYYY-MM-DD', 'CC', @IF (YR1 < 50, 20, 19),'YY', YR1, 'MM', MO1, 'DD',
DAY1),
datecol2 = @DATE ('YYYY-MM-DD', 'CC', @IF (YR2 < 50, 20, 19),'YY', YR2, 'MM', MO2, 'DD',
DAY2)
);
```

Calling a Macro without Input Parameters

To call a macro without input parameters, the call statement must supply the open and close parentheses, but without any input values: `#macro ()`.

The following macro is defined without input parameters. The body contains frequently used parameters.

```
MACRO #option_defaults
BEGIN
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
END;
```

This macro is called as follows:

```
#option_defaults ()
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

#option_defaults ()
MAP owner.srctab2, TARGET owner.targtab2;
```

The macro expands as follows:

```
GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
IGNOREUPDATES
MAP owner.srctab, TARGET owner.targtab;

GETINSERTS
GETUPDATES
GETDELETES
INSERTDELETES
MAP owner.srctab2, TARGET owner.targtab2;
```

Calling Other Macros from a Macro

To call other macros from a macro, create a macro definition similar to the following. In this example, the `#make_date` macro is nested within the `#assign_date` macro, and it is called when `#assign_date` runs.

The nested macro must define all, or a subset of, the same parameters that are defined in the base macro. In other words, the input values when the base macro is called must resolve to the parameters in both macros.

The following defines `#assign_date`:

```
MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

The following defines `#make_date`. This macro creates a date format that includes a four-digit year, after first determining whether the two-digit input date should be prefixed with a century value of 19 or 20. Notice that the `PARAMS` statement of `#make_date` contains a subset of the parameters in the `#assign_date` macro.

```
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month, 'DD',
#day)
END;
```

The following syntax calls `#assign_date`:

```
#assign_date (COL1, YEAR, MONTH, DAY)
```

The macro expands to the following given the preceding input values and the embedded `#make_date` macro:

```
COL1 = @DATE ('YYYY-MM-DD', 'CC', @IF (YEAR < 50, 20, 19),'YY', YEAR, 'MM', MONTH, 'DD',
DAY)
```

Creating Macro Libraries

You can create a macro library that contains one or more macros. By using a macro library, you can define a macro once and then use it within many parameter files.

To Create a Macro Library

1. Open a new file in a text editor.
2. Use commented lines to describe the library, if needed.
3. Using the syntax described in Defining a Macro, enter the syntax for each macro.
4. Save the file in the `dirprm` sub-directory of the Oracle GoldenGate directory as:

   `filename``.mac`

   Where:

   `filename` is the name of the file. The `.mac` extension defines the file as a macro library.

The following sample library named `datelib` contains two macros, `#make_date` and `#assign_date`.

```
-- datelib macro library
--
MACRO #make_date
PARAMS (#year, #month, #day)
BEGIN
@DATE ('YYYY-MM-DD', 'CC', @IF (#year < 50, 20, 19), 'YY', #year, 'MM', #month, 'DD',
#day)
END;

MACRO #assign_date
PARAMS (#target_col, #year, #month, #day)
```

```
BEGIN
#target_col = #make_date (#year, #month, #day)
END;
```

To use a macro library, use the `INCLUDE` parameter at the beginning of a parameter file, as shown in the following sample Replicat parameter file.

```
INCLUDE /ggs/dirprm/datelib.mac
REPLICAT rep
ASSUMETARGETDEFS
USERIDALIAS ogg
MAP fin.acct_tab, TARGET fin.account;
```

When including a long macro library in a parameter file, you can use the `NOLIST` parameter to suppress the listing of each macro in the Extract or Replicat report file. Listing can be turned on and off by placing the `LIST` and `NOLIST` parameters anywhere within the parameter file or within the macro library file. In the following example, `NOLIST` suppresses the listing of each macro in the `hugelib` macro library. Specifying `LIST` after the `INCLUDE` statement restores normal listing to the report file.

```
NOLIST
INCLUDE /ggs/dirprm/hugelib.mac
LIST
INCLUDE /ggs/dirprm/mdatelib.mac
REPLICAT REP
```

Tracing Macro Expansion

You can trace macro expansion with the `CMDTRACE` parameter. With `CMDTRACE` enabled, macro expansion steps are shown in the Extract or Replicat report file.

Syntax

```
CMDTRACE [ON | OFF | DETAIL]
```

Where:

- `ON` enables tracing.
- `OFF` disables tracing.
- `DETAIL` produces a verbose display of macro expansion.

In the following example, tracing is enabled before `#testmac` is called, then disabled after the macro's execution.

```
REPLICAT REP
MACRO #testmac
BEGIN
COL1 = COL2,
COL3 = COL4,
END;
...
CMDTRACE ON
MAP test.table1, TARGET test.table2,
COLMAP (#testmac);
CMDTRACE OFF
```

Using User Exits to Extend Oracle GoldenGate Capabilities

User exits are custom routines that you write in C programming code and call during Extract or Replicat processing. User exits extend and customize the functionality of the Extract and Replicat processes with minimal complexity and risk. With user exits, you can respond to database events when they occur, without altering production programs.

When to Implement User Exits

You can employ user exits as an alternative to, or in conjunction with, the column-conversion functions that are available within Oracle GoldenGate. User exits can be a better alternative to the built-in functions because a user exit processes data only once (when the data is extracted) rather than twice (once when the data is extracted and once to perform the transformation).

The following are some ways in which you can implement user exits:

- Perform arithmetic operations, date conversions, or table lookups while mapping from one table to another.
- Implement record archival functions offline.
- Respond to unusual database events in custom ways, for example by sending an e-mail message or a page based on an output value.
- Accumulate totals and gather statistics.
- Manipulate a record.
- Repair invalid data.
- Calculate the net difference in a record before and after an update.
- Accept or reject records for extraction or replication based on complex criteria.
- Normalize a database during conversion.

Making Oracle GoldenGate Record Information Available to the Routine

The basis for most user exit processing is the `EXIT_CALL_PROCESS_RECORD` function. For Extract, this function is called just before a record buffer is output to the trail. For Replicat, it is called just before a record is applied to the target. If source-target mapping is specified in the parameter file, the `EXIT_CALL_PROCESS_RECORD` event takes place after the mapping is performed.

When `EXIT_CALL_PROCESS_RECORD` is called, the record buffer and other record information are available to it through callback routines. The user exit can map, transform, clean, or perform any other operation with the data record. When it is finished, the user exit can return a status indicating whether the record should be processed or ignored by Extract or Replicat.

Creating User Exits

The following instructions help you to create user exits on Windows and UNIX systems. For more information about the parameters and functions that are described in these instructions, see Reference for Oracle GoldenGate for Windows and UNIX.

> **Note:**
>
> User exits are case-sensitive for database object names. Names are returned exactly as they are defined in the hosting database. Object names must be fully qualified.

To Create User Exits

1. In C code, create either a shared object (UNIX systems) or a DLL (Windows) and create or export a routine to be called from Extract or Replicat. This routine is the communication point between Oracle GoldenGate and your routines. Name the routine whatever you want. The routine must accept the following Oracle GoldenGate user exit parameters:
   - `EXIT_CALL_TYPE`: Indicates when, during processing, the routine is called.
   - `EXIT_CALL_RESULT`: Provides a response to the routine.
   - `EXIT_PARAMS`: Supplies information to the routine. This function enables you to use the `EXITPARAM` option of the `TABLE` or `MAP` statement to pass a parameter that is a literal string to the user exit. This is only valid during the exit call to process a specific record. This function also enables you to pass parameters specified with the `PARAMS` option of the `CUSEREXIT` parameter at the exit call startup.
2. In the source code, include the `usrdecs.h` file. The `usrdecs.h` file is the include file for the user exit API. It contains type definitions, return status values, callback function codes, and a number of other definitions. The `usrdecs.h` file is installed within the Oracle GoldenGate directory. Do not modify this file.
3. Include Oracle GoldenGate callback routines in the user exit when applicable. Callback routines retrieve record and application context information, and they modify the contents of data records. To implement a callback routine, use the `ERCALLBACK` function in the shared object. The user callback routine behaves differently based on the function code that is passed to the callback routine.

   ```
   ERCALLBACK (function_code, buffer, result_code);
   ```

   Where:

   - *`function_code`* is the function to be executed by the callback routine.
   - *`buffer`* is a void pointer to a buffer containing a predefined structure associated with the specified function code.
   - *`result_code`* is the status of the function that is executed by the callback routine. The result code that is returned by the callback routine indicates whether or not the callback function was successful.
   - On Windows systems, Extract and Replicat export the `ERCALLBACK` function that is to be called from the user exit routine. The user exit must explicitly load the callback function at run-time using the appropriate Windows API calls.
4. Include the `CUSEREXIT` parameter in your Extract or Replicat parameter file. This parameter accepts the name of the shared object or DLL and the name of the exported routine that is to be called from Extract or Replicat. You can specify the full path of the shared object or DLL or let the operating system's standard search strategy locate the shared object.

   ```
   CUSEREXIT {DLL | shared_object} routine
   [, INCLUDEUPDATEBEFORES]
   [, PARAMS 'startup_string']
   ```

Where:

- `DLL` is a Windows DLL and `shared_object` is a UNIX shared object that contains the user exit function.
- `INCLUDEUPDATEBEFORES` gets before images for `UPDATE` operations.
- `PARAMS 'startup_string'` supplies a startup string, such as a startup parameter.

Example 8-31 Example of Base Syntax, UNIX

```
CUSEREXIT eruserexit.so MyUserExit
```

Example 8-32 Example Base Syntax, Windows

```
CUSEREXIT eruserexit.dll MyUserExit
```

Supporting Character-set Conversion in User Exits

To maintain data integrity, a user exit needs to understand the character set of the character-type data that it exchanges with an Oracle GoldenGate process. Oracle GoldenGate user exit logic provides globalization support for:

- character-based database metadata, such as the names of catalogs, schemas, tables, and columns
- the values of character-type columns, such as `CHAR`, `VARCHAR2`, `CLOB`, `NCHAR`, `NVARCHAR2`, and `NCLOB`, as well as string-based numbers, date-time, and intervals.

Properly converting between character sets allows column data to be compared, manipulated, converted, and mapped properly from one type of database and character set to another. Most of this processing is performed when the `EXIT_CALL_PROCESS_RECORD` call type is called and the record buffer and other record information is made available through callback routines.

The user exit has its own session character set. This is defined by the `GET_SESSION_CHARSET` and `SET_SESSION_CHARSET` callback functions. The caller process provides conversion between character sets if the character set of the user exit is different from the hosting context of the process.

To enable this support in user exits, there is the `GET_DATABASE_METADATA` callback function code. This function enables the user exit to get database metadata, such as the locale and the character set of the character-type data that it exchanges with the process that calls it (Extract, data pump, Replicat). It also returns how the database treats the case-sensitivity of object names, how it treats quoted and unquoted names, and how it stores object names.

For more information about these components, see Reference for Oracle GoldenGate for Windows and UNIX.

Using Macros to Check Name Metadata

The object name that is passed by the user exit API is the exact name that is encoded in the user-exit session character set, and exactly the same name that is retrieved from the database. If the user exit compares the object name with a literal string, the user exit must retrieve the database locale and then normalize the string so that it is compared with the object name in the same encoding.

Oracle GoldenGate provides the following macros that can be called by the user exit to check the metadata of database object names. For example, a macro can be used to check whether a quoted table name is case-sensitive and whether it is stored as mixed-case in the database server. These macros are defined in the `usrdecs.h` file.

Table 8-14 Macros for metadata checking

Macro	What it verifies
`supportsMixedCaseIdentifiers( nameMeta, DbObjType )`	Whether the database treats a mixed-case unquoted name of a specified data type as case-sensitive and stores the name in mixed case.
`supportsMixedCaseQuotedIdentifiers( nameMeta, DBObjType )`	Whether the database treats the mixed-case quoted name of a specified data type as case-sensitive and stores the name in mixed case.
`storesLowerCaseIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in lower case.
`storesLowerCaseQuotedIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in lower case.
`storesMixedCaseIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in mixed case.
`storesMixedCaseQuotedIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in mixed case.
`storesUpperCaseIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case unquoted name of a specified data type as case-insensitive and stores the name in upper case.
`storesUpperCaseQuotedIdentifiers( nameMeta, DbObjType )`	Whether the database treats the mixed-case quoted name of a specified data type as case-insensitive and stores the name in upper case.

Describing the Character Format

The input parameter `column_value_mode` describes the character format of the data that is being processed and is used in several of the function codes. The following table describes the meaning of the `EXIT_FN_RAW_FORMAT`, `EXIT_FN_CHAR_FORMAT`, and `EXIT_FN_CNVTED_SESS_FORMAT` format codes, per data type.

Table 8-15 column_value_mode_matrix Meanings

Data Type	EXIT_FN_RAW_FORMAT	EXIT_FN_CHAR_FORMAT	EXIT_FN_CNVTED_SESS_FORMAT
`CHAR` `"abc"`	2-byte null indicator + 2-byte length info + column value 0000 0004 61 62 63 20	"abc" encoded in `ASCII` or `EBCDIC`. `NULL` terminated. Tailing spaces are trimmed.	"abc" encoded in user exit session character set. `NOT NULL` terminated. Tailing spaces are trimmed by default unless the `GLOBALS` parameter `NOTRIMSPACES` is specified.

Table 8-15 (Cont.) column_value_mode_matrix Meanings

Data Type	EXIT_FN_RAW_FORMAT	EXIT_FN_CHAR_FORMAT	EXIT_FN_CNVTED_SESS_FORMAT

`NCHAR`
`0061 0062 0063 0020`
	2-byte null indicator +
	2-byte length info +
	column value.
	0000 0008 00 61 0062 0063 0020
		"abc" (encoded in `UTF8`) or truncated at the first byte, depending on whether `NCHAR` is treated as UTF-8.
		`NULL` terminated.
		Trailing spaces are trimmed.
			"abc" encoded in user exit session character set.
			`NOT NULL` terminated.
			Tailing spaces are trimmed by default unless the `GLOBALS` parameter `NOTRIMSPACES` is specified.

`VARCHAR2` `"abc"`	2-byte null indicator + 2-byte length info + column value	"abc" encoded in `ASCII` or `EBCDIC`. `NULL` terminated. No trimming.	"abc" encoded in user exit session character set. `NOT NULL` terminated. No trimming.

`NVARCHAR2`
`0061 0062 0063 0020`
	2-byte null indicator +
	2-byte length info +
	column value
		"abc" (encoded in `UTF8`) or truncated at the first byte, depending on whether `NVARCHAR2` is treated as `UTF-8`.
		`NULL` terminated.
		No trimming.
			"abc"encoded in user exit session character set.
			`NOT NULL` terminated.
			No trimming.

`CLOB`	2-byte null indicator + 2-byte length info + column value	Similar to `VARCHAR2`, but only output up to 4K bytes. `NULL` Terminated. No trimming.	Similar to `VARCHAR2`, but only output data requested in user exit session character set. `NOT NULL` terminated. No trimming.
`NCLOB`	2-byte null indicator + 2-byte length info + column value	Similar to `NVARCHAR2`, but only output up to 4K bytes. `NULL` terminated. No trimming.	Similar to `NVARCHAR2`, but only output data requested in user exit session character set. `NOT NULL` terminated. No trimming.

`NUMBER`
`123.89`
	2-byte null indicator +
	2-byte length info +
	column value
		"123.89" encoded in `ASCII` or `EBCDIC`.
		`NULL` terminated.
			"123.89" encoded in user exit session character set.
			`NOT NULL` terminated.

`DATE`
`31-May-11`
	2-byte null indicator +
	2-byte length info +
	column value
		"2011-05-31" encoded in `ASCII` or `EBCDIC`.
		`NULL` terminated.
			"2011-05-31" encoded in user exit session character set.
			`NOT NULL` terminated.

`TIMESTAMP`
`31-May-11 12.00.00 AM`
	2-byte null indicator +
	2-byte length info +
	column value
		"2011-05-31 12.00.00 AM" encoded in `ASCII` or `EBCDIC`.
		`NULL` terminated.
			"2011-05-31 12.00.00 AM" encoded in user exit session character set.
			`NOT NULL` terminated.

`Interval Year to Month or Interval Day to Second`	2-byte null indicator + 2-byte length info + column value	NA	NA
`RAW`	2-byte null indicator + 2-byte length info + column value	2-byte null indicator + 2-byte length info + column value	2-byte null indicator + 2-byte length info + column value

Upgrading User Exits

The `usrdecs.h` file is versioned to allow backward compatibility with existing user exits when enhancements or upgrades, such as new functions or structural changes, are added to a new Oracle GoldenGate release. The version of the `usrdecs.h` file is printed in the report file at the startup of Replicat or Extract.

To use new user exit functionality, you must recompile your routines to include the new `usrdecs` file. Routines that do not use new features do not need to be recompiled.

Viewing Examples of How to Use the User Exit Functions

Oracle GoldenGate installs the following sample user exit files into the `UserExitExamples` directory of the Oracle GoldenGate installation directory:

- `exitdemo.c` shows how to initialize the user exit, issue callbacks at given exit points, and modify data. It also demonstrates how to retrieve the fully qualified table name or a specific metadata part, such as the name of the catalog or container, or the schema, or just the unqualified table name. In addition, this demo shows how to process DDL data. The demo is not specific to any database type.
- `exitdemo_utf16.c` shows how to use UTF16-encoded data (both metadata and column data) in the callback structures for information exchanged between the user exit and the caller process.
- `exitdemo_more_recs.c` shows an example of how to use the same input record multiple times to generate several target records.
- `exitdemo_lob.c` shows an example of how to get read access to LOB data.
- `exitdemo_pk_befores.c` shows how to access the before and after image portions of a primary key update record, as well as the before images of regular updates (non-key updates). It also shows how to get target row values with `SQLEXEC` in the Replicat parameter file as a means for conflict detection. The resulting fetched values from the target are mapped as the target record when it enters the user exit.

Each directory contains the `*.c` files as well as makefiles and a `readme.txt` file.

Using the Oracle GoldenGate Event Marker System to Raise Database Events

Oracle GoldenGate provides an event marker system, also known as the event marker infrastructure (EMI), which enables the Oracle GoldenGate processes to take a defined action based on an *event record* in the transaction log or in the trail (depending on the data source of the process). The event record is a record that satisfies a specific filter criterion for which you want an action to occur. You can use this system to customize Oracle GoldenGate processing based on database events.

For example, you can use the event marker system to start, suspend, or stop a process, to perform a transformation, or to report statistics. The event marker system can be put to use for purposes such as:

- To establish a synchronization point at which `SQLEXEC` or user exit functions can be performed
- To execute a shell command that executes a data validation script or sends an email

- To activate tracing when a specific account number is detected
- To capture lag history
- To stop or suspend a process to run reports or batch processes at the end of the day

The event marker feature is supported for the replication of data changes, but not for initial loads.

The system requires the following input components:

1. The *event record* that triggers the action can be specified with `FILTER`, `WHERE`, or `SQLEXEC` in a `TABLE` or `MAP` statement. Alternatively, a special `TABLE` statement in a Replicat parameter file enables you to perform `EVENTACTIONS` actions without mapping a source table to a target table.
2. In the `TABLE` or `MAP` statement where you specify the event record, include the `EVENTACTIONS` parameter with the appropriate option to specify the action that is to be taken by the process.

You can combine `EVENTACTIONS` options, as shown in the following examples.

The following causes the process to issue a checkpoint, log an informational message, and ignore the entire transaction (without processing any of it), plus generate a report.

```
EVENTACTIONS (CP BEFORE, REPORT, LOG, IGNORE TRANSACTION)
```

The following writes the event record to the discard file and ignores the entire transaction.

```
EVENTACTIONS (DISCARD, IGNORE TRANS)
```

The following logs an informational message and gracefully stop the process.

```
EVENTACTIONS (LOG INFO, STOP)
```

The following rolls over the trail file and does not write the event record to the new file.

```
EVENTACTIONS (ROLLOVER, IGNORE)
```

For syntax details and additional usage instructions, see *Parameters and Functions Reference for Oracle GoldenGate*.

Case Studies in the Usage of the Event Marker System

These examples highlight some use cases for the event marker system.

Trigger End-of-day Processing

This example specifies the capture of operations that are performed on a special table named `event_table` in the source database. This table exists solely for the purpose of receiving inserts at a predetermined time, for example at 5:00 P.M. every day. When Replicat receives the transaction record for this operation, it stops gracefully to allow operators to start end-of-day processing jobs. By using the insert on the `event_table` table every day, the operators know that Replicat has applied all committed transactions up to 5:00. `IGNORE` causes Replicat to ignore the event record itself, because it has no purpose in the target database. `LOG INFO` causes Replicat to log an informational message about the operation.

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

Simplify Transition from Initial Load to Change Synchronization

Event actions and event tables can be used to help with the transition from an initial load to ongoing change replication. For example, suppose an existing, populated source table must be added to the Oracle GoldenGate configuration. This table must be created on the target, and then the two must be synchronized by using an export/import. This example assumes that an event table named `source.event_table` exists in the source database and is specified in a Replicat `TABLE` statement.

```
TABLE source.event_table, EVENTACTIONS (IGNORE, LOG INFO, STOP);
```

To allow users to continue working with the new source table, it is added to the Extract parameter file, but not to the Replicat parameter file. Extract begins capturing data from this table to the trail, where it is stored.

At the point where the source and target are read-consistent after the export, an event record is inserted into the event table on the source, which propagates to the target. When Replicat receives the event record (marking the read-consistent point), the process stops as directed by `EVENTACTIONS STOP`. This allows the new table to be added to the Replicat `MAP` statement. Replicat can be positioned to start replication from the timestamp of the event record, eliminating the need to use the `HANDLECOLLISIONS` parameter. Operations in the trail from before the event record can be ignored because it is known that they were applied in the export.

The event record itself is ignored by Replicat, but an informational message is logged.

Stop Processing When Data Anomalies are Encountered

This example uses `ABORT` to stop Replicat immediately with a fatal error if an anomaly is detected in a bank record, where the customer withdraws more money than the account contains. In this case, the source table is mapped to a target table in a Replicat `MAP` statement for actual replication to the target. A `TABLE` statement is also used for the source table, so that the `ABORT` action stops Replicat before it applies the anomaly to the target database. `ABORT` takes precedence over processing the record.

```
MAP source.account, TARGET target.account;
TABLE source.account, FILTER (withdrawal > balance), EVENTACTIONS (ABORT);
```

Trace a Specific Order Number

The following example enables Replicat tracing only for an order transaction that contains an insert operation for a specific order number (`order_no = 1`). The trace information is written to the `order_1.trc` trace file. The `MAP` parameter specifies the mapping of the source table to the target table.

```
MAP sales.order, TARGET rpt.order;
TABLE source.order,
FILTER (@GETENV ('GGHEADER', 'OPTYPE') = 'INSERT' AND order_no = 1), &
EVENTACTIONS (TRACE order_1.trc TRANSACTION);
```

Execute a Batch Process

In this example, a batch process executes once a month to clear the source database of accumulated data. At the beginning of the transaction, typically a batch transaction, a record is written to a special `job` table to indicate that the batch job is starting. `TRANSACTION` is used with `IGNORE` to specify that the entire transaction must be ignored by Extract, because the target system does not need to reflect the deleted records. By ignoring the work on the Extract side, unnecessary trail and network overhead is eliminated.

```
TABLE source.job, FILTER (@streq (job_type = 'HOUSEKEEPING')=1), &
EVENTACTIONS (IGNORE TRANSACTION);
```

> **Note:**
>
> If a logical batch delete were to be composed of multiple smaller batches, each smaller batch would require an insert into the job table as the first record in the transaction.

Propagate Only a SQL Statement without the Resultant Operations

This example shows how different `EVENTACTIONS` clauses can be used in combination on the source and target to replicate just a SQL statement rather than the operations that result from that statement. In this case, it is an `INSERT INTO...SELECT` transaction. Such a transaction could generate millions of rows that would need to be propagated, but with this method, all that is propagated is the initial SQL statement to reduce trail and network overhead. The `SELECT`s are all performed on the target. This configuration requires perfectly synchronized source and target tables in order to maintain data integrity.

Extract:

```
TABLE source.statement, EVENTACTIONS (IGNORE TRANS INCLUDEEVENT);
```

Replicat:

```
TABLE source.statement, SQLEXEC (execute SQL statement), &
EVENTACTIONS (INFO, IGNORE);
```

To use this configuration, a `statement` table is populated with the first operation in the transaction, that being the `INSERT INTO...SELECT`, which becomes the event record.

> **Note:**
>
> For large SQL statements, the statement can be written to multiple columns in the table. For example, eight `VARCHAR (4000)` columns could be used to store SQL statements up to 32 KB in length.

Because of the `IGNORE TRANS INCLUDEEVENT`, Extract ignores all of the subsequent inserts that are associated with the `SELECT` portion of the statement, but writes the event record that contains the SQL text to the trail. Using a `TABLE` statement, Replicat passes the event record to a `SQLEXEC` statement that concatenates the SQL text columns, if necessary, and executes the `INSERT INTO...SELECT` statement using the target tables as the input for the `SELECT` sub-query.

Committing Other Transactions Before Starting a Long-running Transaction

This use of `EVENTACTIONS` ensures that all open transactions that are being processed by Replicat get committed to the target before the start of a long running transaction. It forces Replicat to write a checkpoint before beginning work on the large transaction. Forcing a checkpoint constrains any potential recovery to just the long running transaction. Because a

Replicat checkpoint implies a commit to the database, it frees any outstanding locks and makes the pending changes visible to other sessions.

```
TABLE source.batch_table, EVENTACTIONS (CHECKPOINT BEFORE);
```

Execute a Shell Script to Validate Data

This example executes a shell script that runs another script that validates data after Replicat applies the last transaction in a test run. On the source, an event record is written to an event table named `source.event`. The record inserts the value `COMPARE` into the `event_type` column of the event table, and this record gets replicated at the end of the other test data. In the `TABLE` statement in the Replicat parameter file, the `FILTER` clause qualifies the record and then triggers the shell script `compare_db.sh` to run as specified by `SHELL` in the `EVENTACTIONS` clause. After that, Replicat stops immediately as specified by `FORCESTOP`.

Extract:

```
TABLE src.*;
TABLE test.event;
```

Replicat:

```
MAP src.*, TARGET targ.*;
MAP test.event, TARGET test.event, FILTER (@streq (event_type, 'COMPARE')=1), &
EVENTACTIONS (SHELL 'compare_db.sh', FORCESTOP);
```

Oracle GoldenGate Globalization Support

This chapter describes Oracle GoldenGate globalization support, which enables the processing of data in its native language encoding.

Preserving the Character Set

In order to process the data in its native language encoding, Oracle GoldenGate takes into consideration the character set of the database and the operating system locale, if applicable.

Character Set of Database Structural Metadata

Oracle GoldenGate processes catalog, schema, table and column names in their native language as determined by the character set encoding of the source and target databases. This processing is extended to the parameter files and command interpreter, where they are processed according to the operating system locale. These objects appear in their localized format throughout the client interface, on the console, and in files.

Character Set of Character-type Data

The Oracle GoldenGate apply process (Replicat) supports the conversion of data from one character set to another when the data is contained in character column types. Character-set conversion support is limited to column-to-column mapping as performed with the `COLMAP` or `USEDEFAULTS` clauses of a `TABLE` or `MAP` statement. It is not supported by the column-conversion functions, by `SQLEXEC`, or by the `TOKENS` feature.

See Mapping and Manipulating Data for more information about character sets, conversion between them, and data mapping.

Character Set of Database Connection

The Extract and Replicat processes use a session character set when connecting to the database. For Oracle Databases, the session character set is set to the same as the database character set by both Extract and Replicat. For MySQL, the session character set is taken from the `SESSIONCHARSET` option of `SOURCEDB` and `TARGETDB`, or from the `SESSIONCHARSET` parameter set globally in the `GLOBALS` file. For other database types, it is obtained programmatically. In addition, Oracle GoldenGate processes use a session character set for communication and data transfer between Oracle GoldenGate and the database, such as for SQL queries, fetches, and applying data.

Character Set of Text Input and Output

Oracle GoldenGate supports text input and output in the default character set of the host operating system for the following:

- Console
- Command-line input and output
- `FORMATASCII`, `FORMATSQL`, `FORMATXML` parameters, text files such as parameter files, data-definitions files, error log, process reports, discard files, and other human-readable files that are used by Oracle GoldenGate users to configure, run, and monitor the Oracle GoldenGate environment.

In the event that the platform does not support a required character set as the default in the operating system, you can use the following parameters to specify a character set:

- `CHARSET` parameter to specify a character set to be used by processes to read their parameter files.
- `CHARSET` option of the `DEFSFILE` parameter to generate a data-definitions file in a specific character set.

The GGSCI command console always operates in the character set of the local operating system for both keyboard and `OBEY` file input and console output.

Using Unicode and Native Characters

Oracle GoldenGate supports the use of an escape sequence to represent characters in Unicode or in the native character encoding of the Windows, UNIX, and Linux operating systems. You can use an escape sequence if the operating system does not support the required character, or for any other purpose when needed. For more information about this support, see Support for Escape Sequences.

Using Oracle GoldenGate Parameter Files

Most Oracle GoldenGate functionality is controlled by means of parameters specified in parameter files. A parameter file is a plain text file that is read by an associated Oracle GoldenGate process. Oracle GoldenGate uses two types of parameter files: a `GLOBALS` file and runtime parameter files.

Globalization Support for Parameter Files

Oracle GoldenGate creates parameter files in the default character set of the local operating system. In the event that the local platform does not support a required character set as the default in the operating system, you can use the `CHARSET` parameter either globally or per-process to specify a character set for parameter files.

To avoid issues caused by character-set incompatibilities, create or edit a parameter file on the server where the associated process will be running. Avoid creating it on one system (such as your Windows laptop) and then transferring the file to the UNIX server where Oracle GoldenGate is installed and where the operating system character set is different. Oracle GoldenGate provides some tools to help with character set incompatibilities if you must create the parameter file on a different system:

- You can use the `CHARSET` parameter to specify a compatible character set for the parameter file. This parameter must be placed on the first line of the parameter file and allows you to write the file in the specified character set. After the file is transferred to the other system, do not edit the file on that system.
- You can use Unicode notation to substitute for characters that are not compatible with the character set of the operating system where the file will be used. See Support for Escape Sequences for more information about Unicode notation.

See *Parameters and Functions Reference for Oracle GoldenGate* for more information about the `CHARSET` parameter.

Working with the GLOBALS File

The `GLOBALS` file stores parameters that relate to the Oracle GoldenGate instance as a whole. This is in contrast to runtime parameters, which are coupled with a specific process such as Extract. The parameters in the `GLOBALS` file apply to all processes in the Oracle GoldenGate instance, but can be overridden by specific process parameters. A `GLOBALS` parameter file may or may not be required for your Oracle GoldenGate environment.

> **Note:**
>
> The `GLOBALS` file is specific to Classic Architecture.

When used, a `GLOBALS` file must exist before starting any Oracle GoldenGate processes, including GGSCI. The GGSCI program reads the `GLOBALS` file and passes the parameters to processes that need them.

To Create a GLOBALS File

1. From the Oracle GoldenGate installation location, run GGSCI and enter the following command, or open a file in a text editor.

   ```
   EDIT PARAMS ./GLOBALS
   ```

   > **Note:**
   >
   > The `./` portion of this command must be used, because the `GLOBALS` file must reside at the root of the Oracle GoldenGate installation file.

2. In the file, enter the `GLOBALS` parameters, one per line.

3. Save the file. If you used a text editor, save the file as `GLOBALS` (uppercase, without a file extension) at the root of the Oracle GoldenGate installation directory. If you created the file correctly in GGSCI, the file is saved that way automatically. Do not move this file.
4. Exit GGSCI. You must start from a new GGSCI session before issuing commands or starting processes that reference the `GLOBALS` file.

Working with Runtime Parameters

Runtime parameters give you control over the various aspects of Oracle GoldenGate synchronization, such as:

- Data selection, mapping, transformation, and replication
- DDL and sequence selection, mapping, and replication (where supported)
- Error resolution
- Logging
- Status and error reporting
- System resource usage
- Startup and runtime behavior

There can only be one manager process for each Oracle GoldenGate installation. It is configured using the `mgr.prm` parameter file. Although you can have multiple Extracts and Replicats running in a single installation, each one can only be associated by a single parameter file. For Extracts and Replicats, they are identified by their case-insensitive name. For example, an Extract called `EXT_DEMO`, would have **1** associated parameter file called `EXT_DEMO.prm`. See Simplifying the Creation of Parameter Files for more information about simplifying the use of parameter files.

There are two types of parameters: global (not to be confused with `GLOBALS` parameters) and object-specific:

- Global parameters apply to all database objects that are specified in a parameter file. Some global parameters affect process behavior, while others affect such things as memory utilization and so forth. `USERIDALIAS` in Example 8-33 and Example 8-35 is an example of a global parameter. In most cases, a global parameter can appear anywhere in the file before the parameters that specify database objects, such as the `TABLE` and `MAP` statements. A global parameter should be listed only once in the file. When listed more than once, only the *last* instance is active, and all other instances are ignored.
- Object-specific parameters enable you to apply different processing rules for different sets of database objects. `GETINSERTS` and `IGNOREINSERTS` in Example 8-35 are examples of object-specific parameters. Each precedes a `MAP` statement that specifies the objects to be affected. Object-specific parameters take effect in the order that each one is listed in the file.

Example 8-33and Example 8-35 are examples of basic parameter files for Extract and Replicat. Comments are preceded by double hyphens.

The preceding example reflects a case-insensitive Oracle database, where the object names are specified in the `TABLE` statements in capitals. For a case-insensitive Oracle database, it makes no difference how the names are entered in the parameter file (upper, lower, mixed case). For other databases, the case of the object names may matter. See Specifying Object Names in Oracle GoldenGate Input for more information about specifying object names.

Note the use of single and double quote marks in the Replicat example. For databases that require quote marks to enforce case-sensitive object names, such as Oracle, you must enclose case-sensitive object names within double quotes in the parameter file as well. For other case-sensitive databases, specify the names as they are stored in the database. For more information about specifying names and literals, see Specifying Object Names in Oracle GoldenGate Input.

Example 8-33 Sample Extract Parameter File

```
-- Extract group name
EXTRACT capt
-- Extract database user login, with alias to credentials in the credential store.
USERIDALIAS ogg1
-- Remote host to where captured data is sent in encrypted format:
RMTHOSTOPTIONS sysb, MGRPORT 7809, ENCRYPT AES192 KEYNAME mykey
-- Encryption specification for trail data
ENCRYPTTRAIL AES192
-- Remote trail on the remote host
RMTTRAIL /ggs/dirdat/aa

With these lines:
-- Encryption specification for trail data
ENCRYPTTRAIL AES192
-- Local trail on the remote host
EXTTRAIL ./dirdat/aa
```

Example 8-34 Sample Extract Pump Parameter File

```
-- Extract Pump group name
EXTRACT pmp
-- Remote host to where captured data is sent in encrypted format:
RMTHOSTOPTIONS sysb, MGRPORT 7809, ENCRYPT AES192 KEYNAME mykey
-- Encryption specification for trail data
ENCRYPTTRAIL AES192
-- Remote trail on the remote host
RMTTRAIL /ggs/dirdat/bb
-- TABLE statements that identify data to capture.
TABLE FIN.*;
TABLE SALES.*;
```

Example 8-35 Sample Replicat Parameter File

```
-- Replicat group name
REPLICAT deliv
-- Replicat database user login, with alias to credentials in the credential store
USERIDALIAS ogg2
-- Error handling rules
REPERROR DEFAULT, ABEND
-- Ignore INSERT operations
IGNOREINSERTS
-- MAP statement to map source objects to target objects and
-- specify column mapping
MAP "fin"."accTAB", TARGET "fin"."accTAB",
COLMAP ("Account" = "Acct",
"Balance" = "Bal",
"Branch" = "Branch");
-- Get INSERT operations
GETINSERTS
-- MAP statement to map source objects to target objects and
-- filter to apply only the 'NY' branch data.
```

```
MAP "fin"."teller", TARGET "fin"."tellTAB",
WHERE ("Branch" = 'NY');
```

Creating a Parameter File

Oracle recommends using GGSCI when writing the parameter file in the character set of the operating system, but if using the `CHARSET` parameter and writing the file in a different character set, use a text editor instead of GGSCI.

Topics:

Creating a Parameter File in GGSCI and Admin Client

To create a parameter file, use the `EDIT PARAMS` command within the command line interface through GGSCI or Admin Client user interface or use a text editor directly. When you use the command line interface, you are using a standard text editor, but your parameter file is saved automatically with the correct file name and in the correct directory.

When you create a parameter file with `EDIT PARAMS`, it is saved to the `dirprm` sub-directory of the Oracle GoldenGate directory. You can create a parameter file in a directory other than `dirprm`, but you also must specify the full path name with the `PARAMS` option of the `ADD EXTRACT` or `ADD REPLICAT` command when you create your process groups. Once paired with an Extract or Replicat group, a parameter file must remain in its original location for Oracle GoldenGate to operate properly once processing has started.

The `EDIT PARAMS` command launches the following text editors within the GGSCI or Admin Client interface:

- Notepad on Microsoft Windows systems
- The vi editor on UNIX and Linux systems. DB2 for i only supports vi when connected with SSH or xterm. For more information, see Creating a Parameter File with a Text Editor.

> **Note:**
>
> You can change the default editor through the GGSCI or Admin Client interface by using the `SET EDITOR` command.

1. From the directory where Oracle GoldenGate is installed, run GGSCI or Admin Client.
2. In GGSCI or Admin Client, issue the following command to open the default text editor.

   ```
   EDIT PARAMS group_name
   ```

   Where:

   `group_name` is either `mgr` (for the Manager process) or the name of the Extract or Replicat group for which the file is being created. The name of an Extract or Replicat parameter file must match that of the process group.

   The following creates or edits the parameter file for an Extract group named `extora`.

   ```
   EDIT PARAMS extora
   ```

   The following creates or edits the parameter file for the Manager process.

   ```
   EDIT PARAMS MGR
   ```

3. Using the editing functions of the text editor, enter as many comment lines as you want to describe this file, making certain that each comment line is preceded with two hyphens (--).
4. On non-commented lines, enter the Oracle GoldenGate parameters, starting a new line for each parameter statement.

   Oracle GoldenGate parameters have the following syntax:

   ```
   PARAMETER_NAME argument [, option] [&]
   ```

   Where:

   - `PARAMETER_NAME` is the name of the parameter.
   - `argument` is a required argument for the parameter. Some parameters take arguments, but others do not. Commas between arguments are optional.

     ```
     EXTRACT myext
     USERIDALIAS ogg1
     ENCRYPT AES192 KEYNAME mykey
     ENCRYPTTRAIL AES 192
     EXTTRAIL ./dirdat/c1, PURGE
     CUSEREXIT userexit.dll MyUserExit, INCLUDEUPDATEBEFORES, &
       PARAMS "init.properties"
     TABLE myschema.mytable;
     ```

   - `[, option]` is an optional argument.
   - `[&]` is required at the end of each line in a multi-line parameter statement, as in the `CUSEREXIT` parameter statement in the previous example. The exceptions are the following, which can accept, but do not require, the ampersand because they terminate with a semicolon:
     - `MAP`
     - `TABLE`
     - `SEQUENCE`
     - `FILE`
     - `QUERY`

   > **Note:**
   >
   > The `RMTHOST` and `RMTHOSTOPTIONS` parameters can be specified together; the `RMTHOST` parameter is *not* required for `RMTHOSTOPTIONS` if the dynamic IP assignment is properly configured. When `RMTHOSTOPTIONS` is used, the `MGRPORT` option is ignored.

5. Save and close the file.

Creating a Parameter File with a Text Editor

You can create a parameter file outside GGSCI or Admin Client by using a text editor, but make certain to:

- Save the parameter file with the name of the Extract or Replicat group that owns it, or save it with the name `mgr` if the Manager process owns it. Use the `.prm` file extension. For example: `extfin.prm` and `mgr.prm`.

- Save the parameter file in the `dirprm` directory of the Oracle GoldenGate installation directory.
- For DB2 for i systems, you can edit parameter files from a 5250 terminal using SEU or EDTF. If you use SEU, you must copy the file using the `CPYTOSTMF` command, specify an encoding of CCSID 1208, and line endings of *LF. If editing with EDTF from F15 (services) ensure that you change the CCSID of the file to 1208 and the EOL option to *LF.

  Alternatively, you can use the `Rfile` command from the IBM Portable Application Solutions Environment for i.

Validating a Parameter File

The `checkprm` validation native command is run from the command line and gives an assessment of the specified parameter file, with a configurable application and running environment. It can provide either a simple `PASS`/`FAIL` or with optional details about how the values of each parameter are stored and interpreted.

The `CHECKPRM` executable file can be found in the Oracle GoldenGate installation directory for Classic Architecture and in the `/bin` directory of Microservices Architecture. The input to `checkprm` is case insensitive. If a value string contains spaces, it does not need to be quoted because `checkprm` can recognize meaningful values. If no mode is specified to `checkprm`, then all parameters applicable to any mode of the component will be accepted.

The output of `checkprm` is assembled with four possible sections:

- help messages
- pre-validation error
- validation result
- parameter details

A pre-validation error is typically an error that prevents a normal parameter validation from executing, such as missing options or an inaccessible parameter file. If an option value is specified incorrectly, a list of possible inputs for that option is provided. If the result is `FAIL`, each error is in the final result message. If the result is `PASS`, a message that some of the parameters are subject to further runtime validation. The parameter detailed output contains the validation context, the values read from GLOBALS (if it is present), and the specified parameters. The parameter and options are printed with proper indentation to illustrate these relationships.

Table 8-16 describes all of the arguments that you can use with the `checkprm` commands. When you use `checkprm` and do not use any of these arguments, then `checkprm` attempts to automatically detect Extract or Replicat and the platform and database of the Oracle GoldenGate installation.

Table 8-16 checkprm Arguments

Argument	Purpose & Behavior
`None`	Displays usage information
`-v`	Displays banner. Cannot be combined with other options.
`? | help`	Displays detailed usage information, include all possible values of each option. Cannot be combine with other options.

Table 8-16 (Cont.) checkprm Arguments

Argument	Purpose & Behavior
`parameter_file`	Specifies the name of the parameter file, has to be the first argument if a validation is requested. You must specify the absolute path to the parameter file. For example, `CHECKPRM ./dirprm/myext.prm`.

`-COMPONENT | -C`	Specifies the running component (application) that this parameter file is validated for. This option can be omitted for Extract or Replicat because automatic detection is attempted. Valid values include:
	`CACHEFILEDUMP COBGEN CONVCHK CONVPRM DDLCOB DEFGEN EMSCLNT EXTRACT GGCMD GGSCI KEYGEN LOGDUMP`
	`MGR OGGERR REPLICAT RETRACE`
	`REVERSE SERVER GLOBALS`
	There is no default for this option.

`-MODE | -M`	Specifies the mode of the running application if applicable. This option is optional, only applicable to Extract or Replicat. If no mode is specified, the validation is performed for all Extract or Replicat modes. Valid input of this option includes: • Integrated Extract • Initial Load Extract • Remote Task Extract • Data Pump Extract • Passive Extract • Classic Replicat • Coordinated Replicat • Integrated Replicat • Parallel Integrated Replicat • Parallel Nonintegrated Replicat • Special Run Replicat • Remote Task When key in the value for this option, the application name is optional, as long as it matches the va lue of component. For example, "`Data Pump Extract`" is equivalent to "`Data Pump`" if the component is Extract. However, it is invalid if the component is Replicat.

`-PLATFORM | -P`	Specifies the platform the application is supposed to run on. The default value is the platform that this `checkprm` executable is running on.
	The possible values are:
	`AIX HP-OSS HPUX-IT HPUX-PA`
	`Linux OS400 ZOS Solaris SPARC`
	`Solaris x86 Windows x64 All`

Table 8-16 (Cont.) checkprm Arguments

Argument	Purpose & Behavior

`-DATABASE | -D`	Specifies the database the application is built against. The default value is the database for your Oracle GoldenGate installation.
	The database options are (case insensitive):
	`Generic`
	`Oracle 8`
	`Oracle 9i`
	`Oracle 10g`
	`Oracle 11g`
	`Oracle 12c`
	`Oracle 18c`
	`Oracle 19c`
	`Oracle 21c`
	`Sybase`
	`DB2LUW 9.5`
	`DB2LUW 9.7`
	`DB2LUW 10.5`
	`DB2LUW 10.1`
	`DB2LUW 11.1`
	`DB2`
	`Remote Teradata`
	`Timesten`
	`Timesten 7`
	`Timesten 11.2.1`
	`MySQL`
	`Ctree8`
	`Ctree9`
	`DB2 for i`
	`Remote MSSQL`
	`MSSQL CDC`
	`Informix`
	`Informix1150`
	`Informix1170`
	`Informix1210`
	`Ingres SQL/MX`
	`DB2 z/OS`
	`PostgreSQL`

`-VERBOSE | -V`	Directs `checkprm` to print out detailed parameter information, to demonstrate how the values are read and interpreted. It must be the last option specified in a validation.

Following are some use examples:

```
checkprm ?
checkprm  ./dirprm/ext1.prm -C extract -m data pump -p Linux -v
checkprm  ./dirprm/ext1.prm -m integrated
checkprm  ./dirprm/rep1.prm -m integrated
checkprm  ./dirprm/mgr.prm -C mgr -v
checkprm GLOBALS -c GLOBALS
```

Verifying Using `CHECKPARAMS` Parameter

An alternative to using the recommended `checkprm` utility, is to check the syntax of parameters in an Extract or Replicat parameter file for accuracy using the `CHECKPARAMS` parameter. This process can be used with Extract or Replicat.

To Verify Parameter Syntax

1. Include the `CHECKPARAMS` parameter in the parameter file.
2. Start the associated process by issuing the `START EXTRACT` or `START REPLICAT` command.

   ```
   START {EXTRACT | REPLICAT} group_name
   ```

   The process audits the syntax, writes the results to the report file or the screen, and then stops.
3. Do either of the following:
   - If the syntax is correct, remove the `CHECKPARAMS` parameter before starting the process to process data.
   - If the syntax is wrong, correct it based on the findings in the report. You can run another test to verify the changes, if desired. Remove `CHECKPARAMS` before starting the process to process data.

For more information about `CHECKPARAMS`, see *Parameters and Functions Reference for Oracle GoldenGate*.

Viewing a Parameter File

You can view a parameter file directly from the command shell of the operating system, or you can view it from the GGSCI or Admin Client command line interface, using the `VIEW PARAMS` command.

```
VIEW PARAMS group_name
```

Where:

`group_name` is either `mgr` (for Manager) or the name of the Extract or Replicat group that is associated with the parameter file.

> **Caution:**
>
> Do not use `VIEW PARAMS` to view an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). The contents may become corrupted. View the parameter file from outside the command line interface.

If the parameter file was created in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory, specify the full path name as shown in the following example.

```
VIEW PARAMS c:\lpparms\replp.prm
```

Changing a Parameter File

An Oracle GoldenGate process must be stopped before changing its parameter file, and then started again after saving the parameter file. Changing parameter settings while a process is running can have unexpected results, especially if you are adding tables or changing mapping or filtering rules.

Caution:

Do not use the `EDIT PARAMS` command to view or edit an existing parameter file that is in a character set other than that of the local operating system (such as one where the `CHARSET` option was used to specify a different character set). The contents may become corrupted. View the parameter file from outside Admin Client or GGSCI.

To Change Parameters:

1. Stop the process by issuing the following command in Admin Client or GGSCI. To stop Manager in a Windows cluster, use the Cluster Administrator.

```
STOP {EXTRACT | REPLICAT | MANAGER} group_name
```

2. Open the parameter file by using a text editor or the `EDIT PARAMS` command in Admin Client GGSCI.

```
EDIT PARAMS mgr
```

3. Make the edits, and then save the file.
4. Start the process by issuing the following command in Admin Client GGSCI. Use the Cluster Administrator if starting Manager in a Windows cluster.

```
START {EXTRACT | REPLICAT | MANAGER} group_name
```

When an Extract process or Replicat process is restarted, it picks up right where it left off. You do not need to queisce database activity prior to bouncing the Oracle GoldenGate process to replace a parameter file.

Simplifying the Creation of Parameter Files

You can reduce the number of times that a parameter must be specified by using the following time-saving tools.

Using Macros

You can use macros to automate multiple uses of a parameter statement. See Using Oracle GoldenGate Macros to Simplify and Automate Work.

Using OBEY

You can create a library of text files that contain frequently used parameter settings, and then you can call any of those files from the active parameter file by means of the `OBEY` parameter. The syntax for `OBEY` is:

```
OBEY file_name
```

Where:

`file_name` is the relative or full path name of the file.

Upon encountering an `OBEY` parameter in the active parameter file, Oracle GoldenGate processes the parameters from the referenced file and then returns to the active file to process any remaining parameters. `OBEY` is not supported for the `GLOBALS` parameter file.

If using the `CHARSET` parameter in a parameter file that includes an `OBEY` parameter, the referenced parameter file does not inherit the `CHARSET` character set. The `CHARSET` character set is used to read wildcarded object names in the referenced file, but you must use an escape sequence (`\uX`) for all other multibyte specifications in the referenced file.

See *Parameters and Functions Reference for Oracle GoldenGate* for more information about `OBEY`.

See *Parameters and Functions Reference for Oracle GoldenGate* for more information about `CHARSET`.

Using Parameter Substitution

You can use parameter substitution to assign values to Oracle GoldenGate parameters automatically at run time, instead of assigning static values when you create the parameter file. That way, if values change from run to run, you can avoid having to edit the parameter file or maintain multiple files with different settings. You can simply export the required value at runtime. Parameter substitution can be used for any Oracle GoldenGate process.

To Use Parameter Substitution

1. For each parameter for which substitution is to occur, declare a runtime parameter instead of a value, and precede the runtime parameter name with a question mark (?) as shown in the following example.

   ```
   SOURCEISFILE
   EXTFILE ?EXTFILE
   MAP scott?TABNAME, TARGET tiger ACCOUNT_TARG;
   ```

2. Before starting the Oracle GoldenGate process, use the shell of the operating system to pass the runtime values by means of an environment variable, as shown in #unique_923/unique_923_Connect_42_I1047805 and #unique_923/unique_923_Connect_42_I1047809.

Example 8-36 Parameter substitution on Windows

```
C:\GGS> set EXTFILE=C:\ggs\extfile
C:\GGS> set TABNAME=PROD.ACCOUNTS
C:\GGS> replicat paramfile c:\ggs\dirprm\parmfl
```

Example 8-37   Parameter substitution on UNIX (Korn shell)

```
$ EXTFILE=/ggs/extfile
$ export EXTFILE
$ TABNAME=PROD.ACCOUNTS
$ export TABNAME
$ replicat paramfile ggs/dirprm/parmfl
```

UNIX is case-sensitive, so the parameter declaration in the parameter file must be the same case as the shell variable assignments.

Using Wildcards

For parameters that accept object names, you can use asterisk (*) and question mark (?) wildcards. The use of wildcards reduces the work of specifying numerous object names or all objects within a given schema. For more information about using wildcards, see Using Wildcards in Database Object Names.

Getting Information about Oracle GoldenGate Parameters

You can use the `INFO PARAM` command to view a parameter's definition information from GGSCI. The name provided in the command line can be a parameter, or an option, but it must be a full name that is part of the names concatenated together using a period (.) as the delimiter. For example:

```
INFO PARAM RMTHOST
RMTHOST.STREAMING
INFO PARAM RMTHOST.STREAMING
```

Using the `GETPARAMINFO`, you can query the runtime parameter values of a running instance, including Extract, Replicat, and Manager. This command is similar to using `checkprm -v`, see Validating a Parameter File. The default behavior is to display all that has ever been queried by the application, parameters and their current values. If a particular parameter name is specified, then the output is filtered by that name. Optionally, the output can be redirect to a file specified by the `-FILE` option. For example:

```
SEND ext1pmp GETPARAMINFO
```

Configure Bi-Directional Replication

In a bidirectional configuration, there are Extract and Replicat processes on both the source and target systems to support the replication of transactional changes on each system to the other system. To support this configuration, each Extract must be able to filter the transactions applied by the local Replicat, so that they are not recaptured and sent back to their source in a continuous loop. Additionally, tables whose key columns are `AUTO_INCREMENT` columns must be set so that there is no conflict between the values on each system.

1. Configure Oracle GoldenGate for high availability or active-active replication.
2. To filter out Replicat operations in a bi-directional configuration so that the applied operations are not captured and looped back to the source again, take the following steps on each MySQL database:
    - Configure each Replicat process to use a checkpoint table. Replicat writes a checkpoint to this table at the end of each transaction. You can use one global checkpoint table or one per Replicat process.

- Specify the name of the checkpoint table with the `FILTERTABLE` option of the `TRANLOGOPTIONS` parameter in the Extract parameter file. The Extract process will ignore transactions that end with an operation to the specified table, which should only be those of Replicat.

  > **Note:**
  >
  > Although optional for other supported databases as a means of enhancing recovery, the use of a checkpoint table is required for MySQL when using bidirectional replication (and likewise, will enhance recovery).

  If using a parallel Replicat in a bidirectional replication, then multiple filter tables are supported using the `TRANLOGOPTIONS FILTERTABLE` option. Multiple filter tables allow the `TRANLOGOPTIONS FILTERTABLE` to be specified multiple times with different table names or wildcards.

  You can include single or multiple `TRANLOGOPTIONS FILTERTABLE` entries in the Extract parameter file. In the following example, multiple `TRANLOGOPTIONS FILTERTABLE`entries are included in the Extract parameter file with explicit object names and wildcards.

  ```
  TRANLOGOPTIONS FILTERTABLE ggs.chkpt2
  TRANLOGOPTIONS FILTERTABLE ggs.chkpt_RABC_*
  ```

3. If replicating data for tables that have `AUTO_INCREMENT` columns, edit the MySQL server and `auto_increment_offset` parameters to avoid discrepancies that could be caused by the bi-directional operations. The following illustrates these parameters, assuming two servers: **ServerA** and **ServerB**.

   ServerA:

   ```
   auto-increment-increment = 2
   auto-increment-offset = 1
   ```

   ServerB:

   ```
   auto-increment-increment = 2
   auto-increment-offset = 2
   ```

Other Oracle GoldenGate Parameters for MySQL

The following parameters may be of use in MySQL installations, and might be required if non-default settings are used for the MySQL database. Other Oracle GoldenGate parameters will be required in addition to these, depending on your intended business use and configuration.

Parameter	Description
`DBOPTIONS` with `CONNECTIONPORT` *`port_number`*	Required to specify to the VAM the TCP/IP connection port number of the MySQL instance to which an Oracle GoldenGate process must connect if MySQL is not running on the default of 3306. `DBOPTIONS CONNECTIONPORT 3307`

Parameter	Description
`DBOPTIONS` with `HOST host_id`	Specifies the DNS name or IP address of the system hosting MySQL to which Replicat must connect.
`DBOPTIONS` with `ALLOWLOBDATATRUNCATE`	Prevents Replicat from abending when replicated LOB data is too large for a target MySQL `CHAR`, `VARCHAR`, `BINARY` or `VARBINARY` column.

`SOURCEDB with USERID` and `PASSWORD`	Specifies database connection information consisting of the database, user name and password to use by an Oracle GoldenGate process that connects to a MySQL database. If MySQL is not running on the default port of 3306, you must specify a complete connection string that includes the port number: `SOURCEDB` *`dbname@hostname:port`*, `USERID` *`user`*, `PASSWORD` *`password`*.Example:
	`SOURCEDB mydb@mymachine:3307, USERID myuser, PASSWORD mypassword`
	If you are not running the MySQL database on port 3306, you must also specify the connection port of the MySQL database in the `DBLOGIN` command when issuing commands that affect the database through GGSCI:
	`DBLOGIN SOURCEDB dbname@hostname:port, USERID user, PASSWORD password`
	For example:
	`GGSCI> DBLOGIN SOURCEDB mydb@mymachine:3307, USERID myuser, PASSWORD mypassword`

`SQLEXEC`	To enable Replicat to bypass the MySQL connection timeout, configure the following command in a `SQLEXEC` statement in the Replicat parameter file. `SQLEXEC "select CURRENT_TIME();" EVERY n MINUTES` **Where**: *`n`* is the maximum interval after which you want Replicat to reconnect. The recommended connection timeout 31536000 seconds (365 days).

Parameter	Description
Global variable `sql_mode`	For heartbeattable to work in MySQL 5.7 , MySQL global variable `sql_mode` should not have `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`. In the following example `sql_mode` includes `NO_ZERO_IN_DATE,NO_ZERO_DATE` values:

```
mysql> show variables like
'%sql_mode%';+---------------+-
+
| Variable_name |
Value




|
+---------------
+-------------------------------------------------
+
| sql_mode      |
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DA
TE,NO_ZERO_DATE,ERROR_FOR_
DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBST
ITUTION |
+---------------
+-------------------------------------------------
----------------------------------------------------
-------------
+
1 row in set (0.00 sec)
```

These values must be removed by issuing the following command:

```
mysql> Set global
sql_mode='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,ERR
OR_FOR_DIVISION_BY_ZERO,NO
_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show variables like '%sql_mode%';
+---------------
+---------------------------------------
--------------------------------------------------+
| Variable_name |
Value

                                                 |
+---------------
+---------------------------------------
--------------------------------------------------+
| sql_mode      |
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,ERROR_FOR_DIV
ISION_BY_ZERO,NO_AUTO_CREA
```

Parameter	Description

```
TE_USER,NO_ENGINE_SUBSTITUTION |
+---------------+-----------------------------------
-------------------------------------------------+
1 row in set (0.01 sec)
```

9

Performance

Learn about monitoring and tuning of Oracle GoldenGate Classic Architecture processes.

Monitoring Oracle GoldenGate Processing

See Monitor in the *Oracle GoldenGate Microservices* guide.

Using the Information Commands

You can view information about Oracle GoldenGate processes from the Oracle GoldenGate MA web interface or use the command line interface to monitor various processes.

See Monitoring Performance, Monitoring Trails, , Monitoring Distribution Path Information Statistics in the *Step by Step Data Replication Using Oracle GoldenGate Microservices* guide.

To learn about command syntax, usage, and examples, see the *Command Line Interface Reference for Oracle GoldenGate* .

Table 9-1 Commands to View Process Information

Command	What it shows
`INFO {EXTRACT | REPLICAT}` *`group`* `[DETAIL]`	Run status, checkpoints, approximate lag, and environmental information.
`INFO MANAGER`	Run status and port number
`INFO ALL`	`INFO` output for all Oracle GoldenGate processes on the system
`STATS {EXTRACT | REPLICAT}` *`group`*	Statistics on processing volume, such as number of operations performed.
`STATUS {EXTRACT | REPLICAT}` *`group`*	Run status (starting, running, stopped, abended)
`STATUS MANAGER`	Run status
`LAG {EXTRACT | REPLICAT}` *`group`*	Latency between last record processed and timestamp in the data source
`INFO {EXTTRAIL | RMTTRAIL}` *`trail`*	Name of associated process, position of last data processed, maximum file size
`SEND MANAGER`	Run status, information about child processes, port information, trail purge settings
`SEND` {`EXTRACT` | `REPLICAT`} group	Depending on the process and selected options, returns information about memory pool, lag, TCP statistics, long-running transactions, process status, recovery progress, and more.
`VIEW REPORT` *`group`*	Contents of the discard file or process report
`VIEW GGSEVT`	Contents of the Oracle GoldenGate error log

Table 9-1 (Cont.) Commands to View Process Information

Command	What it shows

`COMMAND ER wildcard`	Information dependent on the `COMMAND` type:
	`INFO`
	`LAG`
	`SEND`
	`STATS`
	`STATUS`
	`wildcard` is a wildcard specification for the process groups to be affected, for example:
	`INFO ER ext*`
	`STATS ER *`

`INFO PARAM`	Queries for and displays static information.
`GETPARAMINFO`	Displays currently-running parameter values.
`INFO DISTPATH`	Returns information about distribution paths. Before you run this command, ensure that the Distribution Service is running for that deployment.
`INFO EXTTRAIL`	Retrieves configuration information for a local trail. It shows the name of the trail, the Extract that writes to it, the position of the last data processed, and the assigned maximum file size.
`INFO RMTTRAIL`	Retrieves configuration information for a remote trail. It shows the name of the trail, the Extract that writes to it, the position of the last data processed, and the assigned maximum file size.
`INFO ER`	Retrieves information on multiple Extract and Replicat groups as a unit.
`INFO CHECKPOINTTABLE`	Confirms the existence of a checkpoint table and view the date and time that it was created.
`INFO CREDENTIALS`	Retrieves a list of credentials.
`INFO ENCRYPTIONPROFILE`	Returns information about the encryption profiles available with the Service Manager.
`INFO HEARTBEATTABLE`	Displays information about the heartbeat tables configured in the database.
`INFO AUTHORIZATIONPROFILE`	Lists all the authorization profiles in a deployment or information on a specific authorization profile for a specific deployment.
`INFO MASTERKEY`	Displays the contents of a currently open master-key wallet. If a wallet store does not exist, a new wallet store file is created. This wallet store file is then used to host different encrypted keys as they are created.
`INFO PROFILE`	Returns information about managed process profiles.

Table 9-1 (Cont.) Commands to View Process Information

Command	What it shows
`INFO RECVPATH`	Returns information about a target-initiated distribution path in the Receiver Service. Before you run this command, ensure that the Receiver Service is running.
`INFO SCHEMATRANDATA`	Valid for Oracle database only. Determine whether Oracle schema-level supplemental logging is enabled for the specified schema or if any instantiation information is available. Use the `DBLOGIN` command to establish a database connection before using this command.
`INFO TRACETABLE`	Verifies the existence of the specified trace table in the local instance of the database
`INFO TRANDATA`	Displays different outputs depending on the database.
`STATS DISTPATH | RECVPATH`	Get the statistics for the distribution path (DISTPATH) or receiver path (RECVPATH).
`STATS ER`	Retrieve statistics on multiple Extract and Replicat groups as a unit. Use it with wildcards to affect every Extract and Replicat group that satisfies the wildcard.
`STATUS ER`	Checks the status of multiple Extract and Replicat groups as a unit
`STATUS DEPLOYMENT`	View the status of the specified deployment.
`STATUS PMSRVR`	
`STATUS SERVICE`	Displays the status of specified Oracle GoldenGate services.

Monitoring an Extract Recovery

If Extract abends when a long-running transaction is open, it can seem to take a long time to recover when it is started again. To recover its processing state, Extract must search back through the online and archived logs (if necessary) to find the first log record for that long-running transaction. The farther back in time that the transaction started, the longer the recovery takes, in general, and Extract can appear to be stalled.

To confirm that Extract is recovering properly, use the `SEND EXTRACT` command with the `STATUS` option. One of the following status notations appears, and you can follow the progress as Extract changes its log read position over the course of the recovery.

- `In recovery[1]` – Extract is recovering to its checkpoint in the transaction log. Meaning that it is reading from either:

  a) reading from BR checkpoint files and then archived/online logs,

  or

  b) reading from Recovery Checkpoint in archived/online log.

- `In recovery[2]` – Extract is recovering from its checkpoint to the end of the trail. Meaning that a recovery marker is appended to the output trail when the last transaction was not completely written then rewriting the transaction.

- `Recovery complete` – The recovery is finished, and normal processing will resume.

Monitoring Lag

Lag statistics show you how well the Oracle GoldenGate processes are keeping pace with the amount of data that is being generated by the business applications. With this information, you can diagnose suspected problems and tune the performance of the Oracle GoldenGate processes to minimize the latency between the source and target databases. See Tuning the Performance of Oracle GoldenGate for help with tuning Oracle GoldenGate to minimize lag.

About Lag

For Extract, lag is the difference, in seconds, between the time that a record was processed by Extract (based on the system clock) and the timestamp of that record in the data source.

For Replicat, lag is the difference, in seconds, between the time that the last record was processed by Replicat (based on the system clock) and the timestamp of the record in the trail.

To view lag statistics, use either the `LAG` or `SEND` command in GGSCI. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

> **Note:**
>
> The `INFO` command also returns a lag statistic, but this statistic is taken from the last record that was checkpointed, not the current record that is being processed. It is less accurate than `LAG` or `INFO`.

Controlling How Lag is Reported

Use the `LAGREPORTMINUTES` or `LAGREPORTHOURS` parameter to specify the interval at which Manager checks for Extract and Replicat lag. See *Parameters and Functions Reference for Oracle GoldenGate*.

Use the `LAGCRITICALSECONDS`, `LAGCRITICALMINUTES`, or `LAGCRITICALHOURS` parameter to specify a lag threshold that is considered critical, and to force a warning message to the error log when the threshold is reached. This parameter affects Extract and Replicat processes on the local system. See *Parameters and Functions Reference for Oracle GoldenGate*.

Use the `LAGINFOSECONDS`, `LAGINFOMINUTES`, or `LAGINFOHOURS` parameter to specify a lag threshold; if lag exceeds the specified value, Oracle GoldenGate reports lag information to the error log. If the lag exceeds the value specified with the `LAGCRITICAL` parameter, Manager reports the lag as critical; otherwise, it reports the lag as an informational message. A value of zero (0) forces a message at the frequency specified with the `LAGREPORTMINUTES` or `LAGREPORTHOURS` parameter. See *Parameters and Functions Reference for Oracle GoldenGate*.

Using Automatic Heartbeat Tables to Monitor

You can use the default automatic heartbeat table functionality to monitor end-to-end replication lag. Automatic heartbeats are sent from each source database into the replication streams, by updating the records in a *heartbeat seed table* and a *heartbeat table*, and constructing a `heartbeat history table`. Each of the replication processes in the replication path process these heartbeat records and update the information in them. These heartbeat records are inserted or updated into the heartbeat table at the target databases.

The heartbeat tables contain the following information:

- Source database
- Destination database
- Information about the outgoing replication streams:
  - Names of the Extract, pump/Distribution Service, and or Replicat processes in the path
  - Timestamps when heartbeat records were processed by the replication processes.
- Information about the incoming replication streams:
  - Names of the Extract, pump/Distribution Service, and or replicat processes in the path
  - Timestamps when heartbeat records were processed by the replication processes.

Using the information in the heartbeat table and the heartbeat history table, the current and historical lags in each of the replication can be computed.

Replicat can track the current restart position of Extract with automatic heartbeat tables (LOGBSN). This allows regenerating the trail files from the source database, if required and minimizes the redo log retention period of the source database. In addition, by tracking the most recent Extract restart position, the tombstone tables for automatic Conflict Detection and Resolution (ACDR) tables can be purged more frequently.

In a bidirectional configuration, the heartbeat table has as many entries as the number of replication paths to neighbors that the database has and in a unidirectional setup, the table at the source is empty. The outgoing columns have the timestamps and the outgoing path, the local Extract and the downstream processes. The incoming columns have the timestamps and path of the upstream processes and local Replicat.

In a unidirectional configuration, the target database will populate only the incoming columns in the heartbeat table.

> **Note:**
>
> The Automatic Heartbeat functionality is not supported on MySQL version 5.5.

Understanding Heartbeat Table End-To-End Replication Flow

The end-to-end replication process for heartbeat tables relies on using the Oracle GoldenGate trail format. The process is as follows:

Add a heartbeat table to each of your databases with the `ADD HEARTBEATTABLE` command. Add the heartbeat table to all source and target instances and then restart existing Oracle GoldenGate processes to enable heartbeat functionality. Depending on the database, you may or may not be required to create or enable a job to populate the heartbeat table data. See the following sample:

```
GGSCI>DBLOGIN {[SOURCEDB data_source] |[, database@host:port] |USERID {/ |
userid}[, PASSWORD password]
[algorithm ENCRYPTKEY {keyname | DEFAULT}] |USERIDALIAS alias [DOMAIN domain]|
[SYSDBA | SQLID sqlid][SESSIONCHARSET character_set]}
GGSCI>ADD HEARTBEATTABLE
```

(Optional) For Oracle Databases, you must ensure that the Oracle `DBMS_SCHEDULER` is operating correctly as the heartbeat update relies on it. You can query the `DBMS_SCHEDULER` by issuing:

```
select START_DATE, LAST_START_DATE, NEXT_RUN_DATE
from dba_scheduler_jobs
```

Where `job_name ='GG_UPDATE_HEARTBEATS';`
Then look for valid entries for `NEXT_RUN_DATE`, which is the next time the scheduler will run. If this is a timestamp in the past, then no job will run and you must correct it.
A common reason for the scheduler not working is when the parameter `job_queue_processes` is set too low (typically zero). Increase the number of `job_queue_processes` configured in the database with the `ALTER SYSTEM SET JOB_QUEUE_PROCESSES = ##;` command where `##` is the number of job queue processes.

Run an Extract, which on receiving the logical change records (LCR) checks the value in the `OUTGOING_EXTRACT` column.

- If the Extract name matches this value, the `OUTGOING_EXTRACT_TS` column is updated and the record is entered in the trail.
- If the Extract name does not match then the LCR is discarded.
- If the `OUTGOING_EXTRACT` value is `NULL`, it is populated along with `OUTGOING_EXTRACT_TS` and the record is entered in the trail.

The Pump or Distribution server on reading the record, checks the value in the `OUTGOING_ROUTING_PATH` column. This column has a list of distribution paths.
If the value is `NULL`, the column is updated with the current group name (and path if this is a Distribution server),`"*"`, update the `OUTGOING_ROUTING_TS` column, and the record is written into its target trail file.
If the value has a `"*"` in the list, then replace it with *`group name[:pathname],"*"'`*, update the `OUTGOING_ROUTING_TS` column, and the record is written into its target trail file. When the value does not have a asterisk (*) in the list and the pump name is in the list, then the record is sent to the path specified in the relevant *`group name[:pathname],"*"'`* pair in the list. If the pump name is not in the list, the record is discarded.
Run a Replicat, which on receiving the record checks the value in the `OUTGOING_REPLICAT` column.

- If the Replicat name matches the value, the row in the heartbeat table is updated and the record is inserted into the history table.
- If the Replicat name does not match, the record is discarded.
- If the value is `NULL`, the row in the heartbeat and heartbeat history tables are updated with an implicit invocation of the Replicat column mapping.

  Automatic Replicat Column Mapping:

  ```
  REMOTE_DATABASE          = LOCAL_DATABASE
  INCOMING_EXTRACT          = OUTGOING_EXTRACT
  INCOMING_ROUTING_PATH       = OUTGOING_ROUTING_PATH with "*"    removed
  INCOMING_REPLICAT         = @GETENV ("GGENVIRONMENT", "GROUPNAME")
  ```

```
INCOMING_HEARTBEAT_TS     = HEARTBEAT_TIMESTAMP
INCOMING_EXTRACT_TS     = OUTGOING_EXTRACT_TS
INCOMING_ROUTING_TS     = OUTGOING_ROUTING_TS
INCOMING_REPLICAT_TS           = @DATE ('UYYYY-MM-DD 
HH:MI:SS.FFFFFF','JTSLCT',@GETENV ('JULIANTIMESTAMP'))
LOCAL_DATABASE     = REMOTE_DATABASE
OUTGOING_EXTRACT     = INCOMING_EXTRACT
OUTGOING_ROUTING_PATH     = INCOMING_ROUTING_PATH
OUTGOING_HEARTBEAT_TS      = INCOMING_HEARTBEAT_TS
OUTGOING_REPLICAT     = INCOMING_REPLICAT
OUTGOING_HEARTBEAT_TS     = INCOMING_HEARTBEAT_TS
```

There is just one column for `OUTGOING_ROUTING_TS`. If a record passes through multiple pump before being applied by a Replicat, each pump will overwrite the `OUTGOING_ROUTING_TS` column so that the pumps lag that is calculated is not specific to a single pump and refers to the lag across all the pumps specified in `PUMP_PATH`.

Additional Considerations:

Computing lags as the heartbeat flows through the system relies on the clocks of the source and target systems to be set up correctly. It is possible that the lag can be negative if the target system is ahead of the source system. The lag is shown as a negative number so that you are aware of their clock discrepancy and can take actions to fix it.

The timestamp that flows through the system is in UTC. There is no time zone associated with the timestamp so when viewing the heartbeat tables, the lag can be viewed quickly even if different components are in different time zones. You can write any view you want on top of the underlying tables; UTC is recommended.

All the heartbeat entries are written to the trail in UTF-8.

The outgoing and incoming paths together uniquely determine a row. Meaning that if you have two rows with same outgoing path and a different incoming path, then it is considered two unique entries.

Heartbeat Table Details

The `GG_HEARTBEAT` table displays timestamp information of the end-to-end replication time and the timing information at the different components primary and secondary Extract and Replicat.

In a unidirectional environment, only the target database contains information about the replication lag. That is the time when a record is generated at the source database and becomes visible to clients at the target database.

Note:

The automatic heartbeat tables don’t populate the OUTGOING_% columns with data, when both the source and remote databases have the same name. To change the database name, use the utility `DBNEWID`. For details, see the `DBNEWID` Utility.

Column	Data Type	Description
`LOCAL_DATABASE`	`VARCHAR2`	Local database where the replication time from the remote database is measured.
`CURRENT_LOCAL_TS 00:00:00`	DATETIME	

Column	Data Type	Description
`HEARTBEAT_TIMESTAMP`	`TIMESTAMP(6)`	The point in time when a timestamp is generated at the remote database.
`REMOTE_DATABASE`	`VARCHAR2`	Remote database where the timestamp is generated
`INCOMING_EXTRACT`	`VARCHAR2`	Name of the primary Extract (capture) at the remote database
`INCOMING_ROUTING_PATH`	`VARCHAR2`	Name of the secondary Extract (pump) at the remote database
`INCOMING_REPLICAT`	`VARCHAR2`	Name of the Replicat on the local database.
`INCOMING_HEARTBEAT_TS`	`TIMESTAMP(6)`	Final timestamp when the information is inserted into the `GG_HEARTBEAT` table at the local database.
`INCOMING_EXTRACT_TS`	`TIMESTAMP(6)`	Timestamp of the generated timestamp is processed by the primary Extract at the remote database.
`INCOMING_ROUTING_TS`	`TIMESTAMP(6)`	Timestamp of the generated timestamp is processed by the secondary Extract at the remote database.
`INCOMING_REPLICAT_TS`	`TIMESTAMP(6)`	Timestamp of the generated timestamp is processed by Replicat at the local database.
`OUTGOING_EXTRACT`	`VARCHAR2`	Bidirectional/N-way replication: Name of the primary Extract on the local database.
`OUTGOING_ROUTING_PATH`	`VARCHAR2`	Bidirectional/N-way replication: Name of the secondary Extract on the local database.
`OUTGOING_REPLICAT`	`VARCHAR2`	Bidirectional/N-way replication: Name of the Replicat on the remote database.
`OUTGOING_HEARTBEAT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Final timestamp when the information is inserted into the table at the remote database.
`OUTGOING_EXTRACT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by the primary Extract on the local database.
`OUTGOING_ROUTING_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by the secondary Extract on the local database.

Column	Data Type	Description
`OUTGOING_REPLICAT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp of the generated timestamp is processed by Replicat on the remote database.
`INCOMING_REPLICAT_LW_CSN`	`VARCHAR2`	
`INCOMING_EXTRACT_HEARTBEAT_CSN`	`VARCHAR2`	
`INCOMING_EXTRACT_RESTART_CSN`	`VARCHAR2`	
`INCOMING_EXTRACT_RESTART_TS`	`TIMESTAMP(6)`	

The `GG_HEARTBEAT_HISTORY` table displays historical timestamp information of the end-to-end replication time and the timing information at the different components primary and secondary Extract and Replicat.

In a unidirectional environment, only the destination database contains information about the replication lag.

Timestamps are managed in UTC time zone. That is the time when a record is generated at the source database and becomes visible to clients at the target database.

Column	Data Type	Description
`LOCAL_DATABASE`	`VARCHAR2`	Local database where the end-to-end lag is measured.
`HEARTBEAT_RECEIVED_TS`	`TIMESTAMP(6)`	Point in time when a timestamp from the remote database receives at the local database.
`REMOTE_DATABASE`	`VARCHAR2`	Remote database where the timestamp is generated.
`INCOMING_EXTRACT`	`VARCHAR2`	Name of the primary Extract on the remote database.
`INCOMING_ROUTING_PATH`	`VARCHAR2`	Name of the secondary Extract of the remote database.
`INCOMING_REPLICAT`	`VARCHAR2`	Name of the Replicat on the local database.
`INCOMING_HEARTBEAT_TS`	`TIMESTAMP(6)`	Final timestamp when the information is inserted into the `GG_HEARTBEAT_HISTORY` table on the local database.
`INCOMING_EXTRACT_TS`	`TIMESTAMP(6)`	Timestamp when the generated timestamp is processed by the primary Extract on the remote database.
`INCOMING_ROUTING_TS`	`TIMESTAMP(6)`	Timestamp when the generated timestamp is processed by the secondary Extract on the remote database.

Column	Data Type	Description
`INCOMING_REPLICAT_TS`	`TIMESTAMP(6)`	Timestamp when the generated timestamp is processed by Replicat on the local database.
`OUTGOING_EXTRACT`	`VARCHAR2`	Bidirectional/N-way replication: Name of the primary Extract from the local database.
`OUTGOING_ROUTING_PATH`	`VARCHAR2`	Bidirectional/N-way replication: Name of the secondary Extract from the local database.
`OUTGOING_REPLICAT`	`VARCHAR2`	Bidirectional/N-way replication: Name of the Replicat on the remote database.
`OUTGOING_HEARTBEAT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Final timestamp when the information is persistently inserted into the table of the remote database.
`OUTGOING_EXTRACT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by the primary Extract on the local database.
`OUTGOING_ROUTING_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by the secondary Extract on the local database.
`OUTGOING_REPLICAT_TS`	`TIMESTAMP(6)`	Bidirectional/N-way replication: Timestamp when the generated timestamp is processed by Replicat on the remote database.
`REPLICAT_LOW_WATERMARK_CSN`	`String`	This column is populated by Replicat when it processes this heartbeat record. It populates this column with its current low watermark (LWM) when it processes this record. This allows us to choose a LOGBSN from a heartbeat record which is as of the Replicat LWM.
`SOURCE_EXTRACT_HEARTBEAT_CSN`	`String`	This column is populated by Extract and contains the source commit SCN for the heartbeat transaction in the source database. The heartbeat job on the source database cannot populate this value as it will not know the commit SCN apriori.

Column	Data Type	Description
`SOURCE_EXTRACT_RESTART_CSN`	`String`	This column will be populated by Extract and will contain the current LOGBSN when Extract processes this particular heartbeat record. The heartbeat job on the source database will not populate this value.
`SOURCE_EXTRACT_RESTART_CSN_TS`	`TIMESTAMP`	This column will be populated by Extract and will contain the redo timestamp in UTC that corresponds to the current LOGBSN when Extract processes this particular heartbeat record. The heartbeat job on the source database will not populate this value.

The `GG_LAG` view displays information about the replication lag between the local and remote databases.

In a unidirectional environment, only the destination database contains information about the replication lag. The lag is measured in seconds.

Column	Data Type	Description
`LOCAL_DATABASE`	`VARCHAR2`	Local database where the end-to-end replication lag from the remote database is measured.
`CURRENT_LOCAL_TS`	`TIMESTAMP(6)`	Current timestamp of the local database.
`REMOTE_DATABASE`	`VARCHAR2`	Remote database where the timestamp is generated.
`INCOMING_HEARTBEAT_AGE`	`NUMBER`	The age of the most recent heartbeat received from the remote database.
`INCOMING_PATH`	`VARCHAR2`	Replication path from the remote database to the local database with Extract and Replicat components.
`INCOMING_LAG`	`NUMBER`	Replication lag from the remote database to the local database. This is the time where the heartbeat where generated at the remote database minus the time where the information was persistently inserted into the table at the local database.
`OUTGOING_HEARTBEAT_AGE`	`NUMBER`	The age of the most recent heartbeat from the local database to the remote database.
`OUTGOING_PATH`	`VARCHAR2`	Replication Path from Local database to the remote database with Extract and Replicat components

Column	Data Type	Description
`OUTGOING_LAG`	`NUMBER`	Replication Lag from the local database to the remote database. This is the time where the heartbeat where generated at the local database minus the time where the information was persistently inserted into the table at the remote database.
`REMOTE_EXTRACT_RESTART_CSN`	`String`	Source Extract restart position.
`REMOTE_DATABASE` `DB_UNIQUE_NAME`	`String`	Remote database unique name is displayed. If no unique name exists, then the DB_NAME value is displayed.
`REMOTE_EXTRACT_RESTART_CSN_TIME`	`Timestamp`	Timestamp associated with source Extract redo position.
`REMOTE_DB_OLDEST_OPEN_TXN_AGE`	`Timestamp`	Age of the oldest open transaction at the source database that Extract is currently processing. This column can be calculated as `SYSTIMESTAMP - REMOTE_EXTRACT_RESTART_TIME`.
`LOCAL_REPLICAT_LWM_CSN`	`String`	Low watermark CSN of the local Replicat when it processed the heartbeat.

The `GG_LAG_HISTORY` view displays the history information about the replication lag history between the local and remote databases.

In a unidirectional environment, only the destination database contains information about the replication lag.

The unit of the lag units is in seconds.

Column	Data Type	Description
`LOCAL_DATABASE`	`VARCHAR2`	Local database where the end-to-end replication lag from the remote database is measured.
`HEARTBEAT_RECEIVED_TS`	`TIMESTAMP(6)`	Point in time when a timestamp from the remote database receives on the local database.
`REMOTE_DATABASE`	`VARCHAR2`	Remote database where the timestamp is generated.
`DB_NAME`	String	Remote database name.

Column	Data Type	Description
`DB_UNIQUE_NAME`	String	Remote database unique name. If the database unique name doesn't exist, then the `DB_NAME` and `DB_UNIQUE_NAME` will be same. In a switchover to standby scenario, the `db_unique_name` will change but the `db_name` and replication path remain the same
`INCOMING_HEARTBEAT_AGE`	`NUMBER`	The age of the heartbeat table.
`INCOMING_PATH`	`VARCHAR2`	Replication path from the remote database to local database with Extract and Replicat components.
`INCOMING_LAG`	`NUMBER`	Replication lag from the remote database to the local database. This is the time where the heartbeat was generated at the remote database minus the time where the information was persistently inserted into the table on the local database.
`OUTGOING_HEARTBEAT_AGE`	`NUMBER`	
`OUTGOING_PATH`	`VARCHAR2`	Replication path from local database to the remote database with Extract and Replicat components.
`OUTGOING_LAG`	`NUMBER`	Replication lag from the local database to the remote database. This is the time where the heartbeat was generated at the local database minus the time where the information was persistently inserted into the table on the remote database.
`REMOTE_EXTRACT_RESTART_CSN`	`String`	Source Extract restart position.
`REMOTE_EXTRACT_RESTART_CSN_TIME`	`TIMESTAMP`	Timestamp associated with source Extract redo position.
`REMOTE_DB_OLDEST_OPEN_TXN_AGE`	`TIMESTAMP`	Age of the oldest open transaction at the source database that Extract is currently processing. This column can be calculated as: `SYSTIMESTAMP - REMOTE_EXTRACT_RESTART_TIME`
`LOCAL_REPLICAT_LWM_CSN`	`String`	Low watermark CSN of the local Replicat when it processed the heartbeat.
`INCOMING_EXTRACT_LAG`		
`INCOMING_ROUTINE_LAG`		
`INCOMING_REPLICAT_READ_LAG`		

Column	Data Type	Description
`INCOMING_REPICAT_LAG`		
`OUTGOING_EXTRACT_LAG`		
`OUTGOING_ROUTINE_LAG`		
`OUTGOING_REPLICAT_READ_LAG`		
`OUTGOING_REPLICAT_LAG`		

Updating Heartbeat Tables

The `HEARTBEAT_TIMESTAMP` column in the heartbeat seed table must be updated periodically by a database job. The default heartbeat interval is 1 minute and this interval can be specified or overridden using from the command line or the Administration Service web interface.

For Oracle Database, the database job is created automatically. For all other supported databases, you must create background jobs to update the heartbeat timestamp using the database specific scheduler functionality.

See `ADD HEARTBEATTABLE`, `ALTER HEARTBEATTABLE` for details on updating the heartbeat table.

Purging the Heartbeat History Tables

The heartbeat history table is purged periodically using a job. The default interval is 30 days and this interval can be specified or overridden using a command line inteface such as Admin Client or GGSCI or the Administration Service web interface.

For Oracle Database, the database job is created automatically. For all other supported databases, you must create background jobs to purge the heartbeat history table using the database specific scheduler functionality.

Best Practice

Oracle recommends that you:

- Use the same heartbeat frequency on all the databases to makes diagnosis easier.
- Adjust the retention period if space is an issue.
- Retain the default heartbeat table frequency; the frequency set to be 30 to 60 seconds gives the best results for most workloads.
- Use lag history statistics to collect lag and age information.

Using the Automatic Heartbeat Commands

You can use the heartbeat table commands to control the Oracle GoldenGate automatic heartbeat functionality as follows.

Command	Description
`ADD HEARTBEATTABLE`	Creates the heartbeat tables required for automatic heartbeat functionality including the `LOGBSN` columns.
`ALTER HEARTBEATTABLE`	Alters existing heartbeat objects.

Command	Description
`ALTER HEARTBEATTABLE UPGRADE`	Alters the heartbeat tables to add the `LOGBSN` columns to the heartbeat tables. This is optional.
`DELETE HEARTBEATTABLE`	Deletes existing heartbeat objects.
`DELETE HEARTBEATENTRY`	Deletes entries in the heartbeat table.
`INFO HEARTBEATTABLE`	Displays heartbeat table information.

For more information, see the Reference for Oracle GoldenGate for Windows and UNIX.

Monitoring Processing Volume

The `STATS` commands show you the amount of data that is being processed by an Oracle GoldenGate process, and how fast it is being moved through the Oracle GoldenGate system. With this information, you can diagnose suspected problems and tune the performance of the Oracle GoldenGate processes. These commands provide a variety of options to select and filter the output.

The `STATS` commands are: `STATS EXTRACT`, `STATS REPLICAT`, or `STATS ER` command.

You can send interim statistics to the report file at any time with the `SEND EXTRACT` or `SEND REPLICAT` command with the `REPORT` option.

Using the Error Log

Use the Oracle GoldenGate error log to view:

- a history of GGSCI commands
- Oracle GoldenGate processes that started and stopped
- processing that was performed
- errors that occurred
- informational and warning messages

Because the error log shows events as they occurred in sequence, it is a good tool for detecting the cause (or causes) of an error. For example, you might discover that:

- someone stopped a process
- a process failed to make a TCP/IP or database connection
- a process could not open a file

To view the error log, use any of the following:

- Standard shell command to view the `ggserr.log` file within the root Oracle GoldenGate directory
- Oracle GoldenGate Director or Oracle GoldenGate Monitor
- `VIEW GGSEVT` command in GGSCI.

You can control the `ggserr.log` file behavior to:

- Roll over the file when it reaches a maximum size, which is the default to avoid disk space issues.

- All messages are appended to the file by all processes without regard to disk space.
- Disable the file.
- Route messages to another destination, such as the system log.

This behavior is controlled and described in the `ogg-ggserr.xml` file in one of the following locations:

Microservices Architecture
`$OGG_HOME/etc/conf/logging/`

Classic Architecture
`diretc/logging/`

Using the Process Report

Use the process report to view (depending on the process):

- parameters in use
- table and column mapping
- database information
- runtime messages and errors
- runtime statistics for the number of operations processed

Every Extract, Replicat, and Manager process generates a report file. The report can help you diagnose problems that occurred during the run, such as invalid mapping syntax, SQL errors, and connection errors.

To view a process report, use any of the following:

- standard shell command for viewing a text file
- Oracle GoldenGate Monitor
- `VIEW REPORT` command in GGSCI.
- To view information if a process abends without generating a report, use the following command to run the process from the command shell of the operating system (not GGSCI) to send the information to the terminal.

  ```
  process paramfile path.prm
  ```

  Where:

  - The value for *`process`* is either `extract` or `replicat`.
  - The value for *`path`*`.prm` is the fully qualified name of the parameter file, for example:

    ```
    replicat paramfile /ogg/dirdat/repora.prm
    ```

By default, reports have a file extension of `.rpt`, for example `EXTORA.rpt`. The default location is the `dirrpt` sub-directory of the Oracle GoldenGate directory. However, these properties can be changed when the group is created. Once created, a report file must remain in its original location for Oracle GoldenGate to operate properly after processing has started.

To determine the name and location of a process report, use the `INFO EXTRACT`, `INFO REPLICAT`, or `INFO MANAGER` command in GGSCI.

Scheduling Runtime Statistics in the Process Report

By default, runtime statistics are written to the report once, at the end of each run. For long or continuous runs, you can use optional parameters to view these statistics on a regular basis, without waiting for the end of the run.

To set a schedule for reporting runtime statistics, use the `REPORT` parameter in the Extract or Replicat parameter file to specify a day and time to generate runtime statistics in the report. See `REPORT`.

To send runtime statistics to the report on demand, use the `SEND EXTRACT` or `SEND REPLICAT` command with the `REPORT` option to view current runtime statistics when needed.

Viewing Record Counts in the Process Report

Use the `REPORTCOUNT` parameter to report a count of transaction records that Extract or Replicat processed since startup. Each transaction record represents a logical database operation that was performed within a transaction that was captured by Oracle GoldenGate. The record count is printed to the report file and to the screen. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Preventing SQL Errors from Filling the Replicat Report File

Use the `WARNRATE` parameter to set a threshold for the number of SQL errors that can be tolerated on any target table before being reported to the process report and to the error log. The errors are reported as a warning. If your environment can tolerate a large number of these errors, increasing `WARNRATE` helps to minimize the size of those files. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Using the Discard File

By default, a discard file is generated whenever a process is started with the `START` command through GGSCI. The discard file captures information about Oracle GoldenGate operations that failed. This information can help you resolve data errors, such as those that involve invalid column mapping.

The discard file reports such information as:

- The database error message
- The sequence number of the data source or trail file
- The relative byte address of the record in the data source or trail file
- The details of the discarded operation, such as column values of a DML statement or the text of a DDL statement.

To view the discard file, use a text editor or use the `VIEW REPORT` command in GGSCI. See *Parameters and Functions Reference for Oracle GoldenGate*.

The default discard file has the following properties:

- The file is named after the process that creates it, with a default extension of `.dsc`. Example: `finance.dsc`.
- The file is created in the `dirrpt` sub-directory of the Oracle GoldenGate installation directory.

- The maximum file size is 50 megabytes.
- At startup, if a discard file exists, it is purged before new data is written.

You can change these properties by using the `DISCARDFILE` parameter. You can disable the use of a discard file by using the `NODISCARDFILE` parameter. See *Parameters and Functions Reference for Oracle GoldenGate*.

If a proces is started from the command line of the operating system, it does not generate a discard file by default. You can use the `DISCARDFILE` parameter to specify the use of a discard file and its properties.

Once created, a discard file must remain in its original location for Oracle GoldenGate to operate properly after processing has started.

Maintaining the Discard and Report Files

By default, discard files and report files are aged the same way. A new discard or report file is created at the start of a new process run. Old files are aged by appending a sequence number from 0 (the most recent) to 9 (the oldest) to their names.

If the active report or discard file reaches its maximum file size before the end of a run (or over a continuous run), the process abends unless there is an aging schedule in effect. Use the `DISCARDROLLOVER` and `REPORTROLLOVER` parameters to set aging schedules for the discard and report files respectively. These parameters set instructions for rolling over the files at regular intervals, in addition to when the process starts. Not only does this control the size of the files and prevent process outages, but it also provides a predictable set of archives that can be included in your archiving routine. For more information, see the following documentation:

- `DISCARDROLLOVER`
- `REPORTROLLOVER`

No process ever has more than ten aged reports or discard files and one active report or discard file. After the tenth aged file, the oldest is deleted when a new report is created. It is recommended that you establish an archiving schedule for aged reports and discard files in case they are needed to resolve a service request.

Table 9-2 Current Extract and Manager Reports Plus Aged Reports

Permissions	X	Date	Report
-rw-rw-rw-	`1 ggs ggs`	`1193 Oct 11 14:59`	`MGR.rpt`
-rw-rw-rw-	`1 ggs ggs`	`3996 Oct 5  14:02`	`MGR0.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`4384 Oct 5  14:02`	`TCUST.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`1011 Sep 27 14:10`	`TCUST0.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`3184 Sep 27 14:10`	`TCUST1.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`2655 Sep 27 14:06`	`TCUST2.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`2655 Sep 27 14:04`	`TCUST3.rpt`

Table 9-2 (Cont.) Current Extract and Manager Reports Plus Aged Reports

Permissions	X	Date	Report
`-rw-rw-rw-`	`1 ggs ggs`	`2744 Sep 27 13:56`	`TCUST4.rpt`
`-rw-rw-rw-`	`1 ggs ggs`	`3571 Aug 29 14:27`	`TCUST5.rpt`

Reconciling Time Differences

To account for time differences between source and target systems, use the `TCPSOURCETIMER` parameter in the Extract parameter file. This parameter adjusts the timestamps of replicated records for reporting purposes, making it easier to interpret synchronization lag. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Getting Help with Performance Tuning

See Tuning the Performance of Oracle GoldenGate for help with tuning the performance of Oracle GoldenGate.

Tuning the Performance of Oracle GoldenGate

This chapter contains suggestions for improving the performance of Oracle GoldenGate components.

Using Multiple Process Groups

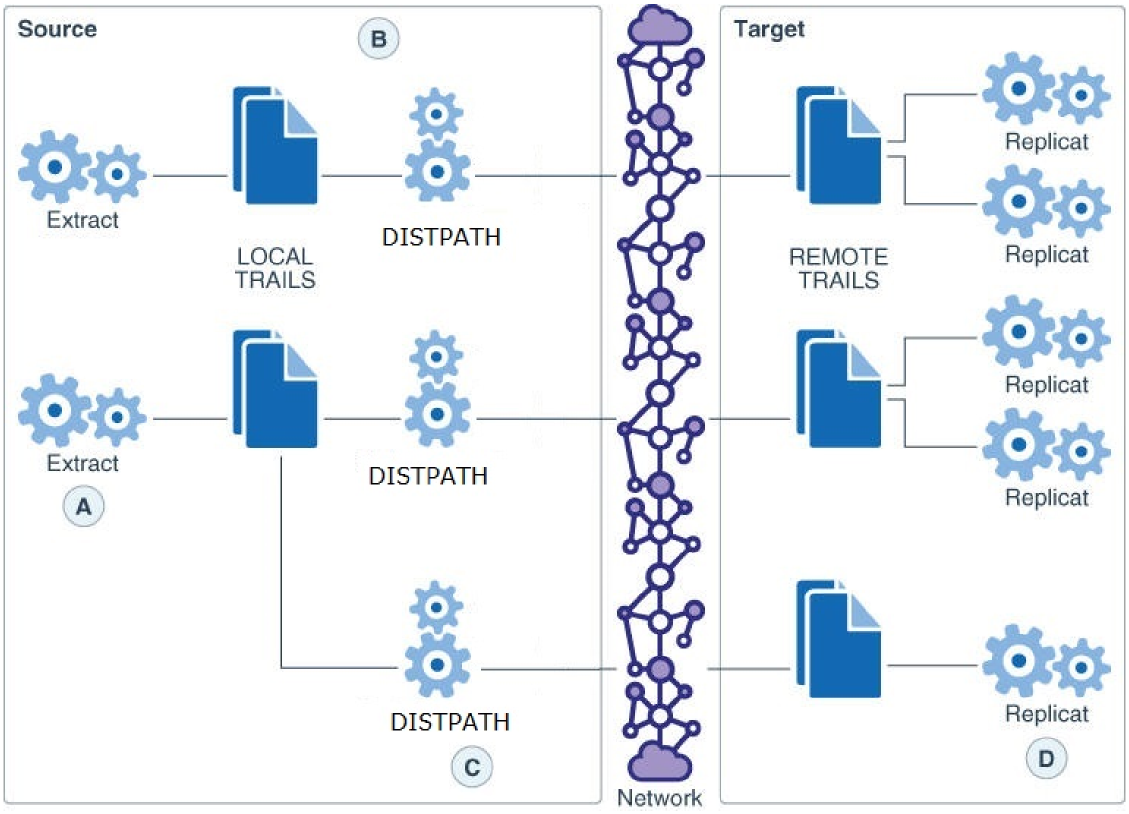
Typically, only one Extract group is required to efficiently capture from a database. However, depending on the redo (transactional) values, or the data and operation types, you may find that you are required to add one or more Extract group to the configuration.

Similarly, only one Replicat group is typically needed to apply data to a target database if using Replicat in coordinated mode. (See About Coordinated Replicat Mode for more information.) However, even in some cases when using Replicat in coordinated mode, you may be required to use multiple Replicat groups. If you are using Replicat in classic mode and your applications generate a high transaction volume, you probably will need to use parallel Replicat groups.

Because each Oracle GoldenGate component — Extract, data pump, trail, Replicat — is an independent module, you can combine them in ways that suit your needs. You can use multiple trails and parallel Extract and Replicat processes (with or without data pumps) to handle large transaction volume, improve performance, eliminate bottlenecks, reduce latency, or isolate the processing of specific data.

The diagram below shows some of the ways that you can configure Oracle GoldenGate to improve throughput speed and overcome network bandwidth issues.

Figure 9-1 Load-balancing configurations that improve performance



The image labels imply the following:

- **A**: Parallel Extracts divide the load. For example, by schema or to isolate tables that generate fetches.
- **B**: A data pump with local trail can be used for filtering, conversion, and network false tolerance.
- **C**: Multiple data pumps work around network per-process bandwidth limitations to enable TCP/IP throughput. Divide the TABLE parameter statements among them.
- **D**: Parallel Replicats increase throughput to the database. Any trail can be read by one or more Replicats. Divide MAP statements among them.

Considerations for Using Multiple Process Groups

Before configuring multiple processing groups, review the following considerations to ensure that your configuration produces the desired results and maintains data integrity.

Maintaining Data Integrity

Not all workloads can be partitioned across multiple groups and still preserve the original transaction atomicity. You must determine whether the objects in one group will ever have dependencies on objects in any other group, transactional or otherwise. For example, tables for which the workload routinely updates the primary key cannot easily be partitioned in this manner. DDL replication (if supported for the database) is not viable in this mode, nor is the use of some `SQLEXEC` or `EVENTACTIONS` features that base their actions on a specific record.

If your tables do not have any foreign- key dependencies or updates to primary keys, you may be able to use multiple processes. Keep related DML together in the same process stream to ensure data integrity.

Number of Groups

The number of concurrent Extract and Replicat process groups that can run on a system depends on how much system memory is available. Each Extract and classic Replicat process needs approximately 25-55 MB of memory or more, depending on the size of the transactions and the number of concurrent transactions. The Oracle GoldenGate command interface fully supports up to 5,000 concurrent Extract and Replicat groups (combined) per instance of Oracle GoldenGate. At the supported level, all groups can be controlled and viewed in full with commands such as the `INFO` and `STATUS` commands.

Beyond the supported level, group information is not displayed and errors may occur. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at a more manageable level, such as 100 or below, in order to manage the environment effectively. The maximum number of groups is controlled by the `MAXGROUPS` parameter, which has a default value of 1000.

For Windows Server environments, the number of process groups that can be run are tightly coupled to the ‘non-interactive’ Windows desktop heap memory settings. The default settings for Windows desktop heap may be enough to run very small numbers of process groups, but as you approach larger amounts of process groups, more than 60 or so, you will either need to adjust the ‘non-interactive’ value of the SharedSection field in the registry, based on this information from Microsoft (Windows desktop heap memory), or increase the number of Oracle GoldenGate homes and spread the total number of desired process groups across these homes.

> **Note:**
>
> For more information on modifying the Windows Desktop Heap memory, review the following Oracle Knowledge Base document (Doc ID 2056225.1).

Memory

The system must have sufficient swap space for each Oracle GoldenGate Extract and Replicat process that will be running. To determine the required swap space:

1. Start up one Extract or Replicat.
2. Run GGSCI.
3. View the report file and find the line `PROCESS VM AVAIL FROM OS (min)`.
4. Round up the value to the next full gigabyte if needed. For example, round up 1.76GB to 2 GB.
5. Multiply that value by the number of Extract and Replicat processes that will be running. The result is the maximum amount of swap space that could be required

See the `CACHEMGR` parameter in *Parameters and Functions Reference for Oracle GoldenGate* for more information about how memory is managed.

Isolating Processing-Intensive Tables

You can use multiple process groups to support certain kinds of tables that tend to interfere with normal processing and cause latency to build on the target. For example:

- Extract may need to perform a fetch from the database because of the data type of the column, because of parameter specifications, or to perform SQL procedures. When data must be fetched from the database, it affects the performance of Extract. You can get fetch statistics from the `STATS EXTRACT` command if you include the `STATOPTIONS REPORTFETCH` parameter in the Extract parameter file. You can then isolate those tables into their own Extract groups, assuming that transactional integrity can be maintained.
- In its classic mode, Replicat process can be a source of performance bottlenecks because it is a single-threaded process that applies operations one at a time by using regular SQL. Even with `BATCHSQL` enabled (see *Parameters and Functions Reference for Oracle GoldenGate*) Replicat may take longer to process tables that have large or long-running transactions, heavy volume, a very large number of columns that change, and LOB data. You can then isolate those tables into their own Replicat groups, assuming that transactional integrity can be maintained.

Using Parallel Replicat Groups on a Target System

This section contains instructions for creating a configuration that pairs one Extract group with multiple Replicat groups. Although it is possible for multiple Replicat processes to read a single trail (no more than three of them to avoid disk contention) it is recommended that you pair each Replicat with its own trail and corresponding Extract process.

For detailed instructions on configuring change synchronization, see Configuring Online Change Synchronization.

To Create the Extract Group

> **Note:**
>
> This configuration includes Extract data-pumps.

1. On the source, use the `ADD EXTRACT` command to create a primary Extract group.
2. On the source, use the `ADD EXTTRAIL` command to specify as many local trails as the number of Replicat groups that you will be creating. All trails must be associated with the primary Extract group.
3. On the source create a data-pump Extract group.
4. On the source, use the `ADD RMTTRAIL` command to specify as many remote trails as the number of Replicat groups that you will be creating. All trails must be associated with the data-pump Extract group.
5. On the source, use the `EDIT PARAMS` command to create Extract parameter files, one for the primary Extract and one for the data pump, that contain the parameters required for your database environment. When configuring Extract, do the following:
    - Divide the source tables among different `TABLE` parameters.

- Link each `TABLE` statement to a different trail. This is done by placing the `TABLE` statements after the `EXTTRAIL` or `RMTTRAIL` parameter that specifies the trail you want those statements to be associated with.

To Create the Replicat Groups

1. On the target, create a Replicat checkpoint table. For instructions, see About Checkpoint Table. All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each trail that you created. Use the `EXTTRAIL` argument of ADD REPLICAT to link the Replicat group to the appropriate trail.
3. On the target, use the `EDIT PARAMS` command to create a Replicat parameter file for each Replicat group that contains the parameters required for your database environment. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to that group.
4. In the Manager parameter file on the target system, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trails.

Using Multiple Extract Groups with Multiple Replicat Groups

Multiple Extract groups write to their own trails. Each trail is read by a dedicated Replicat group.

For detailed instructions on configuring change synchronization, see Configuring Online Change Synchronization.

To Create the Extract Groups

> **Note:**
>
> This configuration includes data pumps.

1. On the source, use the `ADD EXTRACT` command to create the primary Extract groups.
2. On the source, use the `ADD EXTTRAIL` command to specify a local trail for each of the Extract groups that you created.
3. On the source create a data-pump Extract group to read each local trail that you created.
4. On the source, use the `ADD RMTTRAIL` command to specify a remote trail for each of the data-pumps that you created.
5. On the source, use the `EDIT PARAMS` command to create an Extract parameter file for each primary Extract group and each data-pump Extract group.

To Create the Replicat Groups

1. On the target, create a Replicat checkpoint table. For instructions, see Creating a Checkpoint Table. All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each trail. Use the `EXTTRAIL` argument of `ADD REPLICAT` to link the group to the trail.

3. On the target, use the `EDIT PARAMS` command to create a Replicat parameter file for each Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.
4. In the Manager parameter files on the source system and the target system, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trails.

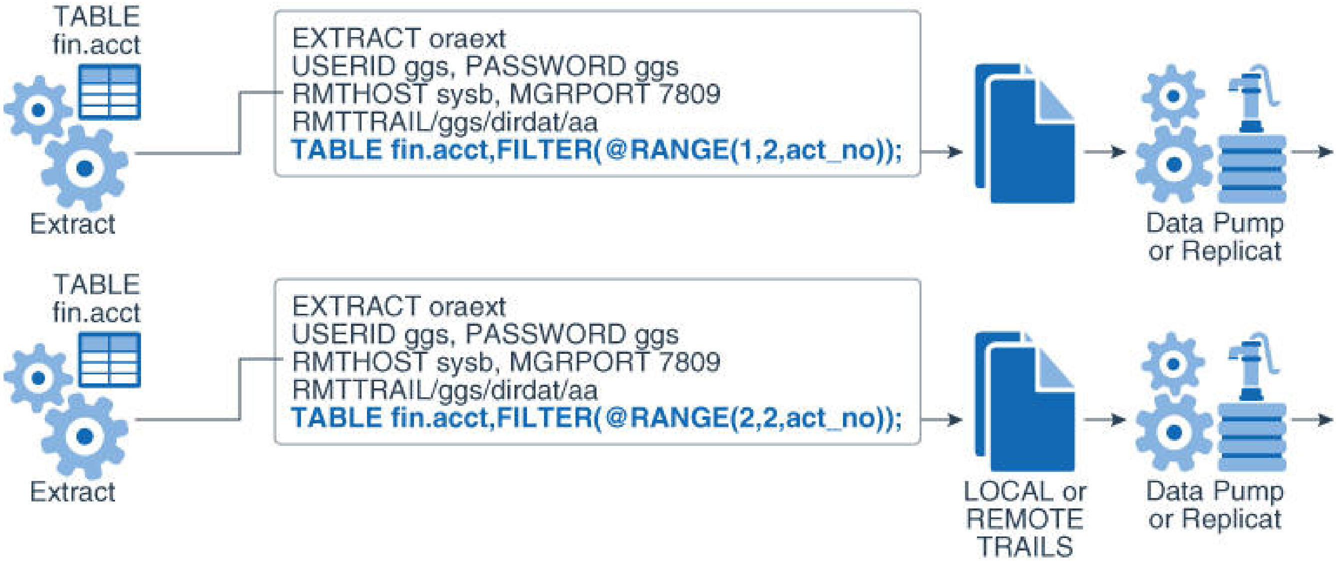
Splitting Large Tables Into Row Ranges Across Process Groups

You can use the `@RANGE` function to divide the rows of any table across two or more Oracle GoldenGate processes. It can be used to increase the throughput of large and heavily accessed tables and also can be used to divide data into sets for distribution to different destinations. Specify each range in a `FILTER` clause in a `TABLE` or `MAP` statement.

`@RANGE` is safe and scalable. It preserves data integrity by guaranteeing that the same row will always be processed by the same process group.

It might be more efficient to use the primary Extract or a data pump to calculate the ranges than to use Replicat. To calculate ranges, Replicat must filter through the entire trail to find data that meets the range specification. However, your business case should determine where this filtering is performed.

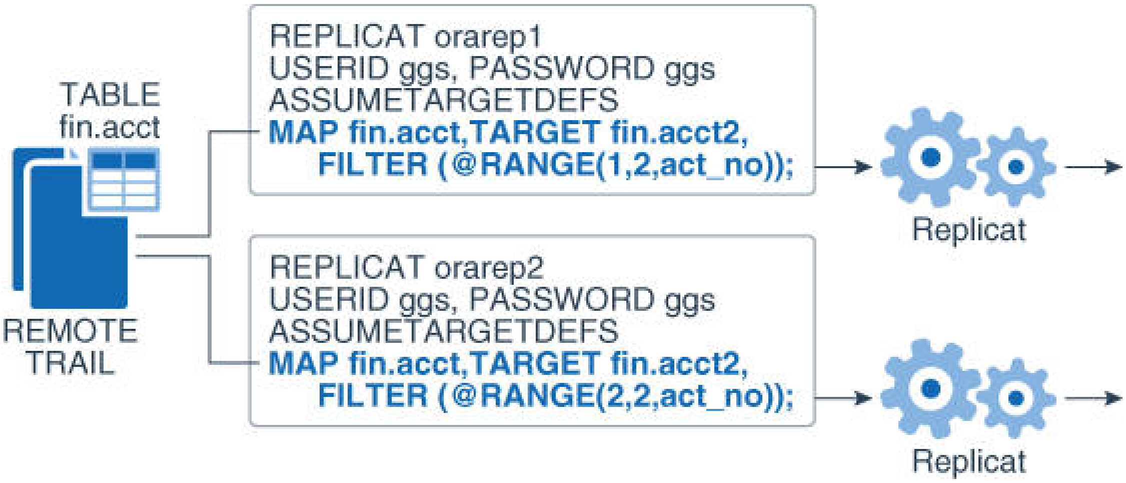
Figure 9-2 Dividing rows of a table between two Extract groups



```
EXTRACT oraext
USERID ggs, PASSWORD ggs
RMTHOST sysb, MGRPORT 7809
RMTTRAIL/ggs/dirdat/aa
TABLE fin.acct,FILTER(@RANGE(1,2,act_no));
```

```
EXTRACT oraext
USERID ggs, PASSWORD ggs
RMTHOST sysb, MGRPORT 7809
RMTTRAIL/ggs/dirdat/aa
TABLE fin.acct,FILTER(@RANGE(2,2,act_no));
```

Figure 9-3 Dividing rows of a table between two Replicat groups



```
REPLICAT orarep1
USERID ggs, PASSWORD ggs
ASSUMETARGETDEFS
MAP fin.acct,TARGET fin.acct2,
    FILTER (@RANGE(1,2,act_no));
```

```
REPLICAT orarep2
USERID ggs, PASSWORD ggs
ASSUMETARGETDEFS
MAP fin.acct,TARGET fin.acct2,
    FILTER (@RANGE(2,2,act_no));
```

Configuring Oracle GoldenGate to Use the Network Efficiently

Inefficiencies in the transfer of data across the network can cause lag in the Extract process and latency on the target. If not corrected, it can eventually cause process failures.

When you first start a new Oracle GoldenGate configuration:

1. Establish benchmarks for what you consider to be acceptable lag and throughput volume for Extract and for Replicat. Keep in mind that Extract will normally be faster than Replicat because of the kind of tasks that each one performs. Over time you will know whether the difference is normal or one that requires tuning or troubleshooting.
2. Set a regular schedule to monitor those processes for lag and volume, as compared to the benchmarks. Look for lag that remains constant or is growing, as opposed to occasional spikes. Continuous, excess lag indicates a bottleneck somewhere in the Oracle GoldenGate configuration. It is a critical first indicator that Oracle GoldenGate needs tuning or that there is an error condition.

To view volume statistics, use the `STATS EXTRACT` or `STATS REPLICAT` command. To view lag statistics, use the `LAG EXTRACT` or `LAG REPLICAT` command.

Detecting a Network Bottleneck that is Affecting Oracle GoldenGate

To detect a network bottleneck that is affecting the throughput of Oracle GoldenGate, follow these steps.

1. Issue the following command to view the ten most recent Extract checkpoints. If you are using a data-pump Extract on the source system, issue the command for the primary Extract and also for the data pump.

   ```
   INFO EXTRACT group, SHOWCH 10
   ```

2. Look for the `Write Checkpoint` statistic. This is the place where Extract is writing to the trail.

   ```
   Write Checkpoint #1

   GGS Log Trail
   Current Checkpoint (current write position):
   ```

```
Sequence #: 2
RBA: 2142224
Timestamp: 2011-01-09 14:16:50.567638
Extract Trail: ./dirdat/eh
```

3. For both the primary Extract and data pump:
    - Determine whether there are more than one or two checkpoints. There can be up to ten.
    - Find the `Write Checkpoint` *`n`* heading that has the highest increment number (for example, `Write Checkpoint #8`) and make a note of the `Sequence`, `RBA`, and `Timestamp` values. This is the most recent checkpoint.
4. Refer to the information that you noted, and make the following validation:
    - Is the primary Extract generating a series of checkpoints, or just the initial checkpoint?
    - If a data pump is in use, is it generating a series of checkpoints, or just one?
5. Issue `INFO EXTRACT` for the primary and data pump Extract processes again.
    - Has the most recent write checkpoint increased? Look at the most recent `Sequence`, `RBA`, and `Timestamp` values to see if their values were incremented forward since the previous `INFO EXTRACT` command.
6. Issue the following command to view the status of the Replicat process.

    ```
    SEND REPLICAT group, STATUS
    ```

    - The status indicates whether Replicat is delaying (waiting for data to process), processing data, or at the end of the trail (EOF).

There is a network bottleneck if the status of Replicat is either in delay mode or at the end of the trail file and either of the following is true:

- You are only using a primary Extract and its write checkpoint is not increasing or is increasing too slowly. Because this Extract process is responsible for sending data across the network, it will eventually run out of memory to contain the backlog of extracted data and abend.
- You are using a data pump, and its write checkpoint is not increasing, but the write checkpoint of the primary Extract is increasing. In this case, the primary Extract can write to its local trail, but the data pump cannot write to the remote trail. The data pump will abend when it runs out of memory to contain the backlog of extracted data. The primary Extract will run until it reaches the last file in the trail sequence and will abend because it cannot make a checkpoint.

> **Note:**
>
> Even when there is a network outage, Replicat will process in a normal manner until it applies all of the remaining data from the trail to the target. Eventually, it will report that it reached the end of the trail file.

Working Around Bandwidth Limitations by Using Data Pumps

Using parallel data pumps may enable you to work around bandwidth limitations that are imposed on a per-process basis in the network configuration. You can use parallel data pumps to send data to the same target system or to different target systems. Data pumps also remove TCP/IP responsibilities from the primary Extract, and their local trails provide fault tolerance.

Increasing the TCP/IP Packet Size

Use the `TCPBUFSIZE` option of the `RMTHOST` parameter to control the size of the TCP socket buffer that Extract maintains. By increasing the size of the buffer, you can send larger packets to the target system. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Use the following steps as a guideline to determine the optimum buffer size for your network.

1. Use the `ping` command from the command shell obtain the average round trip time (RTT), shown in the following example:

   ```
   C:\home\ggs>ping ggsoftware.com
   Pinging ggsoftware.com [192.168.116.171] with 32 bytes of data:
   Reply from 192.168.116.171: bytes=32 time=31ms TTL=56
   Reply from 192.168.116.171: bytes=32 time=61ms TTL=56
   Reply from 192.168.116.171: bytes=32 time=32ms TTL=56
   Reply from 192.168.116.171: bytes=32 time=34ms TTL=56
   Ping statistics for 192.168.116.171:
       Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
   Approximate round trip times in milli-seconds:
       Minimum = 31ms, Maximum = 61ms, Average = 39ms
   ```

2. Multiply that value by the network bandwidth. For example, if average RTT is .08 seconds, and the bandwidth is 100 megabits per second, then the optimum buffer size is:

   ```
   0.08 second * 100 megabits per second = 8 megabits
   ```

3. Divide the result by 8 to determine the number of bytes (8 bits to a byte). For example:

   ```
   8 megabits / 8 = 1 megabyte per second
   ```

   The required unit for `TCPBUFSIZE` is bytes, so you would set it to a value of 1000000.

The maximum socket buffer size for non-Windows systems is usually limited by default. Ask your system administrator to increase the default value on the source and target systems so that Oracle GoldenGate can increase the buffer size configured with `TCPBUFSIZE`.

Eliminating Disk I/O Bottlenecks

I/O activity can cause bottlenecks for both Extract and Replicat.

- A regular Extract generates disk writes to a trail and disk reads from a data source.
- A data pump and Replicat generate disk reads from a local trail.
- Each process writes a recovery checkpoint to its checkpoint file on a regular schedule.

Improving I/O performance Within the System Configuration

If there are I/O waits on the disk subsystems that contain the trail files, put the trails on the fastest disk controller possible.

Check the RAID configuration. Because Oracle GoldenGate writes data sequentially, RAID 0+1 (striping and mirroring) is a better choice than RAID 5, which uses checksums that slow down I/O and are not necessary for these types of files.

Improving I/O Performance Within the Oracle GoldenGate Configuration

You can improve I/O performance by making configurations changes within Oracle GoldenGate. Try increasing the values of the following parameters.

- Use the `CHECKPOINTSECS` parameter to control how often Extract and Replicat make their routine checkpoints.

  > **Note:**
  >
  > `CHECKPOINTSECS` is not valid for an integrated Replicat on an Oracle database system.

- Use the `GROUPTRANSOPS` parameter to control the number of SQL operations that are contained in a Replicat transaction when operating in its normal mode. Increasing the number of operations in a Replicat transaction improves the performance of Oracle GoldenGate by reducing the number of transactions executed by Replicat, and by reducing I/O activity to the checkpoint file and the checkpoint table, if used. Replicat issues a checkpoint whenever it applies a transaction to the target, in addition to its scheduled checkpoints.

  > **Note:**
  >
  > `GROUPTRANSOPS` is not valid for an integrated Replicat on an Oracle database system, unless the inbound server parameter `parallelism` is set to 1.

- Use the `EOFDELAY` or `EOFDELAYCSECS` parameter to control how often Extract, a data pump, or Replicat checks for new data after it has reached the end of the current data in its data source. You can reduce the system I/O overhead of these reads by increasing the value of this parameter.

> **Note:**
>
> Increasing the values of these parameters improves performance, but it also increases the amount of data that must be reprocessed if the process fails. This has an effect on overall latency between source and target. Some testing will help you determine the optimal balance between recovery and performance.

Managing Virtual Memory and Paging

Because Oracle GoldenGate replicates only committed transactions, it stores the operations of each transaction in a managed virtual-memory pool known as a *cache* until it receives either a commit or a rollback for that transaction. One global cache operates as a shared resource of an Extract or Replicat process. The Oracle GoldenGate cache manager takes advantage of the memory management functions of the operating system to ensure that Oracle GoldenGate processes work in a sustained and efficient manner. The `CACHEMGR` parameter controls the amount of virtual memory and temporary disk space that is available for caching uncommitted transaction data that is being processed by Oracle GoldenGate.

When a process starts, the cache manager checks the availability of resources for virtual memory, as shown in the following example:

```
CACHEMGR virtual memory values (may have been adjusted)CACHESIZE: 32GCACHEPAGEOUTSIZE
(normal): 8M PROCESS VM AVAIL FROM OS (min): 63.97GCACHESIZEMAX (strict force to disk):
48G
```

If the current resources are not sufficient, a message like the following may be returned:

```
2013-11-11 14:16:22 WARNING OGG-01842 CACHESIZE PER DYNAMIC DETERMINATION (32G) LESS
THAN RECOMMENDED: 64G (64bit system)vm found: 63.97GCheck swap space. Recommended swap/
extract: 128G (64bit system).
```

If the system exhibits excessive paging and the performance of critical processes is affected, you can reduce the `CACHESIZE` option of the `CACHEMGR`. parameter. You can also control the maximum amount of disk space that can be allocated to the swap directory with the `CACHEDIRECTORY` option. For more information about `CACHEMGR`, see *Parameters and Functions Reference for Oracle GoldenGate*.

Optimizing Data Filtering and Conversion

Heavy amounts of data filtering or data conversion add processing overhead. The following are suggestions for minimizing the impact of this overhead on the other processes on the system.

- Avoid using the primary Extract to filter and convert data. Keep it dedicated to data capture. It will perform better and is less vulnerable to any process failures that result from those activities. The objective is to make certain the primary Extract process is running and keeping pace with the transaction volume.
- Use Replicat or a data-pump to perform filtering and conversion. Consider any of the following configurations:
  - Use a data pump on the source if the system can tolerate the overhead. This configuration works well when there is a high volume of data to be filtered, because it uses less network bandwidth. Only filtered data gets sent to the target, which also can help with security considerations.
  - Use a data pump on an intermediate system. This configuration keeps the source and target systems free of the overhead, but uses more network bandwidth because unfiltered data is sent from the source to the intermediate system.
  - Use a data pump or Replicat on the target if the system can tolerate the overhead, and if there is adequate network bandwidth for sending large amounts of unfiltered data.
- If you have limited system resources, a least-best option is to divide the filtering and conversion work between Extract and Replicat.

Tuning Replicat Transactions

Replicat uses regular SQL, so its performance depends on the performance of the target database and the type of SQL that is being applied (inserts, versus updates or deletes). However, you can take certain steps to maximize Replicat efficiency.

Tuning Coordination Performance Against Barrier Transactions

In a coordinated Replicat configuration, barrier transactions such as updates to the primary key cause an increased number of commits to the database, and they interrupt the benefit of the `GROUPTRANSOPS` feature of Replicat. When there is a high number of barrier transactions in the overall workload of the coordinated Replicat, using a high number of threads can actually degrade Replicat performance.

To maintain high performance when large numbers of barrier transactions are expected, you can do the following:

- Reduce the number of active threads in the group. This reduces the overall number of commits that Replicat performs.
- Move the tables that account for the majority of the barrier transactions, and any tables with which they have dependencies, to a separate coordinated Replicat group that has a small number of threads. Keep the tables that have minimal barrier transactions in the original Replicat group with the higher number of threads, so that parallel performance is maintained without interruption by barrier transactions.
- (Oracle RAC) In a new Replicat configuration, you can increase the `PCTFREE` attribute of the Replicat checkpoint table. However, this must be done before Replicat is started for the first time. The recommended value of `PCTFREE` is 90.

Applying Similar SQL Statements in Arrays

Use the `BATCHSQL` parameter to increase the performance of Replicat. `BATCHSQL` causes Replicat to organize similar SQL statements into arrays and apply them at an accelerated rate. In its normal mode, Replicat applies one SQL statement at a time.

When Replicat is in `BATCHSQL` mode, smaller row changes will show a higher gain in performance than larger row changes. At 100 bytes of data per row change, `BATCHSQL` has been known to improve the performance of Replicat by up to 300 percent, but actual performance benefits will vary, depending on the mix of operations. At around 5,000 bytes of data per row change, the benefits of using `BATCHSQL` diminish.

The gathering of SQL statements into batches improves efficiency but also consumes memory. To maintain optimum performance, use the following `BATCHSQL` options:

```
BATCHESPERQUEUE
BYTESPERQUEUE
OPSPERBATCH
OPSPERQUEUE
```

As a benchmark for setting values, assume that a batch of 1,000 SQL statements at 500 bytes each would require less than 10 megabytes of memory.

You can use `BATCHSQL` with the `BATCHTRANSOPS` option to tune array sizing. `BATCHTRANSOPS` controls the maximum number of batch operations that can be grouped into a transaction before requiring a commit. The default for non-integrated Replicat is 1000. The default for integrated Replicat is 50. If there are many wait dependencies when using integrated Replicat, try reducing the value of `BATCHTRANSOPS`. To determine the number of wait dependencies, view the `TOTAL_WAIT_DEPS` column of the `V$GG_APPLY_COORDINATOR` database view in the Oracle database.

See *Parameters and Functions Reference for Oracle GoldenGate* for additional usage considerations and syntax.

Preventing Full Table Scans in the Absence of Keys

If a target table does not have a primary key, a unique key, or a unique index, Replicat uses all of the columns to build its `WHERE` clause. This is, essentially, a full table scan.

To make row selection more efficient, use a `KEYCOLS` clause in the `TABLE` and `MAP` statements to identify one or more columns as unique. Replicat will use the specified columns as a key. The following example shows a `KEYCOLS` clause in a `TABLE` statement:

```
TABLE hr.emp, KEYCOLS (FIRST_NAME, LAST_NAME, DOB, ID_NO);
```

For usage guidelines and syntax, see the `TABLE` and `MAP` parameters in *Parameters and Functions Reference for Oracle GoldenGate*.

Splitting Large Transactions

If the target database cannot handle large transactions from the source database, you can split them into a series of smaller ones by using the Replicat parameter `MAXTRANSOPS`. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

> **Note:**
>
> `MAXTRANSOPS` is not valid for an integrated Replicat on an Oracle database system.

Adjusting Open Cursors

The Replicat process maintains cursors for cached SQL statements and for `SQLEXEC` operations. Without enough cursors, Replicat must age more statements. By default, Replicat maintains as many cursors as allowed by the `MAXSQLSTATEMENTS` parameter. You might find that the value of this parameter needs to be increased. If so, you might also need to adjust the maximum number of open cursors that are permitted by the database. See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Improving Update Speed

Excessive block fragmentation causes Replicat to apply SQL statements at a slower than normal speed. Reorganize heavily fragmented tables, and then stop and start Replicat to register the new object ID.

Set a Replicat Transaction Timeout

Use the `TRANSACTIONTIMEOUT` parameter to prevent an uncommitted Replicat target transaction from holding locks on the target database and consuming its resources unnecessarily. You can change the value of this parameter so that Replicat can work within existing application timeouts and other database requirements on the target.

`TRANSACTIONTIMEOUT` limits the amount of time that Replicat can hold a target transaction open if it has not received the end-of-transaction record for the last source transaction in that transaction. By default, Replicat groups multiple source transactions into one target transaction to improve performance, but it will not commit a partial source transaction and will wait indefinitely for that last record. The Replicat parameter `GROUPTRANSOPS` controls the minimum size of a grouped target transaction.

The following events could last long enough to trigger `TRANSACTIONTIMEOUT`:

- Network problems prevent trail data from being delivered to the target system.
- Running out of disk space on any system, preventing trail data from being written.

- Collector abends (a rare event).
- Extract abends or is terminated in the middle of writing records for a transaction.
- An Extract data pump abends or is terminated.
- There is a source system failure, such as a power outage or system crash.

See *Parameters and Functions Reference for Oracle GoldenGate* for more information.

Using Healthcheck Scripts to Monitor and Troubleshoot

Oracle GoldenGate Healthcheck script provides database site information for Oracle Databases to allow monitoring and troubleshooting.

The Healtcheck script gathers all replication related configuration and performance information from a database in one single run. Within the scripts, there are many queries regarding the database instance and the database specific information from Extract and Replicat. You can run the script periodically to obtain the latest database side performance information regarding replication.

The output is one of the key information that is needed for support for a qualitative analysis of the replication environment.

Installing, Running, and Uninstalling Healthcheck Scripts

The Healtcheck directory contains three files to install, run and deinstall the Healthcheck script. After the PL/SQL package is installed with `ogghc_install.sql`, you can frequently run the `ohgghc_run.sql` script to generate an output file.

You can deinstall the Healthcheck script using the `ogghc_deinstall.sql` script.

The Healthcheck script is located in:

```
$OGG_HOME/lib/sql/healthcheck
```

To gather information, Oracle recommends you to install and run the Healthcheck as a `SYS` user. However, you can also install and run the script as an Oracle GoldenGate administrator user role. In this case, some system information is not available. The Healthcheck output displays the information that requires `SYS` privileges.

How to Deal with Healthcheck Information?

The output file of the Healthcheck script contains the instance name and a timestamp. By default, information about Integrated Extract and Replicat is gathered. However, you can retrieve information from the legacy Oracle GoldenGate schema or database profile. For this reason, you have to take out the argument of the `EXCLUDETAG` parameter.

Depending of the amount of information being queried, the run time of the script varies (in minutes).

You can eliminate a query that takes too long to process using the Healthcheck script and run another query in a parallel session to get the output.

Components of Healthcheck Information

The Healthcheck script generates an HTML file with JSON objects and HTML code. The output of the Healthcheck script contains the following sections:

- Overview
- Extract
- Replicat
- Table Statistics and Errors
- Tools
- Report Map (Legacy)

Each of these sections contain menus and sub-menus depending on the type of data available.

The following table describes the sections and the data available in those sections based on the query used to generate the Healthcheck output.

Menu	Description
Overview	The Overview section contains information about the following: • General Findings • Database, Extract and Replicat Summary • Capture Parameters • Apply Parameters
Database	Main Menu (The Main menu already contains the query information)
Database Objects	This sub-menu option displays information about the following: • Tables Not Supported By Oracle GoldenGate Integrated Capture • Instantiation SCNs for Apply Schema and Database (DDL)

Menu	Description
Database Details	This sub-menu has a detailed database related various connections and services along with key Oracle GoldenGate parameters and the manual or modified database parameters. The second part shows the basic information about the components, software and patch level of the database. The information is distributed amongst the following sections. • Connection Information • Key init.ora parameters • Database dictionary and fixed table statistics • Software edition • Registry information including History • Replication bundled patch information • NLS Database parameters • Registered log files for capture • Current Database incarnation • Standby redo longs • GoldenGate Administrator User
Replication SQL Analysis	This section provides a complete log of Oracle GoldenGate related information from the session at run time of the script and active session history. This contains complete details about Waits, Events, IO, Contention and SQL.
Objects Instantiation	This sub-menu provides details about the schemas and table-level supplemental logging for Oracle GoldenGate: • Schemas prepared for capture • Table-level supplemental log groups enabled for capture
Extract	Main Menu
Extract Details	The Extract details covered in this sub-menu are: • Capture Runtime Information • Capture Transaction Processing • Capture Processing Information • Logminer Session Statistics • Capture Rules & Rulesets

Menu	Description
Extract Performance	This sub-menu displays information about the Extract performance depending on the type of Extract being used. It displays the progress of the Extract which includes the following details: • Capture Name - Name of the Extract • Client Name - Name of the client where the Extract is running • Client Status - Status of the client where the Extract is running • Processed Low SCN - Processed SCN value of the Extract • Oldest SCN - Oldest SCN value of the Extract
Extract Logminer	This sub-menu contains information mainly used for debugging issues.
Replicat	Main Menu

Menu	Description
Replicat Performance	The Replicat performance provides the following information: - Apply Progress - GoldenGate Inbound Progress Table - Information about Apply Progress table - Apply Network Receiver (ANR) - Apply Reader - Apply Reader - Dequeue Information - Apply Coordinator - Apply Coordinator Watermarks - Open GoldenGate Apply Transactions - Open GoldenGate Apply Transactions -Details - Apply Server Transactions ordered by Server_id - Apply Server Statstics - Summary - Apply Server Statistics - Details - Apply Server Statistics - Auto Tuning - Apply Server Wait Events - Apply Server Session Events - Apply Reader Processes - Apply Coordinator Processes - Apply Server Processes
CDR	This sub-menu provides a detailed log of the Replicat error handlers.
Apply Handler	This sub-menu contains information about the Replicat name, DDL handler, and precommit handler.
Error Management	Main Menu
Error Management Details	This sub-menu has details about the Oracle GoldenGate table statistics sorted by table. It includes information such as server name, source table owner, source table name, destination table owner, destination table name, total operations, inserts, updates, deletes, insert/update/delete collisions, REPERROR discards, REPERROR ignores, WAIT dependencies, and CDR related updates.

Menu	Description
Tools	Main Menu
History	This sub-menu is dependent on the type of query you have run and displays subscriber history, Extract, and Replicat history.
Report Map	Main Menu (Legacy). It provides information on all the queries, which includes details such as: • Run time of the query • Number of Returned Rows • Information if the query has succeeded or failed • Information if this is internal information that is only visible if the script is run as SYS • Disabled Queries The hyperlinks directs you to the appropriate query.
Hints/Description	This sub-menu is a map of all the activities logged in the Healthcheck report.
Alerts	This sub-menu provides a log of the alerts from the general findings about the database, Extract, and Replicat along with general system information.

Menu	Description
Truncates	This sub-menu displays the Oracle GoldenGate related information from the `V$ACTIVE_SESSSION_HISTORY` and `V$SQLAREA` database views. **Note:** You can view only partial results for `V$ACTIVE_SESSSION_HISTORY` , as the original size of the query exceeds the maximum limit.

Menu	Description
Config	This sub-menu allows you to add rules to the following sections that are available on this page: • Column Rules: There can be any number of rules on a single table and they will be applied one after the other in the order they appear in the rules table. It can be an expression using the values of the columns in a row. • Menu Items Exclusions: This option is used to exclude any menu item from the Healthcheck output. Click **Add** to set the rule. • Group By: This section is used to group by keys and aggregates. You can go to any statistic and click G to add a group by sorting.
JS Errors	Main Menu. This page displays debugging information for this framework. **Note:** It is visible only in case of errors.

10

Oracle GoldenGate Business Solutions

Configuring Online Change Synchronization

This chapter describes how to configure online change synchronization.

Overview of Online Change Synchronization

Online change synchronization extracts and replicates data changes continuously to maintain a near real-time target database. The number of Extract and Replicat processes and trails that you will need depends on the replication topology that you want to deploy and the process mode that you will be using.

For detailed information about deploying specific replication topologies, see:

- Using Oracle GoldenGate for Live Reporting
- Using Oracle GoldenGate for Real-time Data Distribution
- Configuring Oracle GoldenGate to Maintain a Live Standby Database
- Configuring Oracle GoldenGate for Active-Active Configuration

You may need to configure multiple Replicat processes if you are replicating between Oracle multitenant container databases.

You may need to configure multiple process groups to achieve a certain performance level. For example, you may want to keep lag below a certain threshold. Lag is the difference between when changes are made within your source applications and when those changes are applied to the target database.

Oracle GoldenGate supports up to 5,000 concurrent Extract and Replicat groups per instance of Oracle GoldenGate Manager. At the supported level, all groups can be controlled and viewed in full with GGSCI commands such as the `INFO` and `STATUS` commands. Oracle GoldenGate recommends keeping the number of Extract and Replicat groups (combined) at the default level of 300 or below in order to manage your environment effectively.

See Tuning the Performance of Oracle GoldenGate for more information about configuring Oracle GoldenGate for best performance.

Initial Synchronization

After you configure your change-synchronization groups and trails following the directions in this chapter, see Instantiating Oracle GoldenGate Using Initial Load to prepare the target tables for synchronization.

An initial load takes a copy of entire source tables, transforms the data if necessary, and applies it to the target tables so that the movement of transaction data begins from a synchronized state. The first time that you start change synchronization should be during the initial synchronization process. Change synchronization keeps track of ongoing transactional changes while the load is being applied.

Choosing Names for Processes and Files

It is helpful to develop consistent naming conventions for the Oracle GoldenGate processes and files before you start configuration steps. Choosing meaningful names helps you differentiate among multiple processes and files in displays, error logs, and external monitoring programs. In addition, it accommodates the naming of additional processes and files later, as your environment changes or expands.

This section contains instructions for:

Naming Conventions for Processes

When specifying a process or group name, follow these rules.

- For the following types of processes, you can use up to eight characters, including non-alphanumeric characters such as the underscore (_):
  - Online Extract group
  - Initial-load Extract
  - Online Replicat group created in classic (non-coordinated) mode
  - Online Replicat group created in integrated mode (Oracle only)
- For coordinated and parallel Replicat process group, you can use up to five characters, including non-alphanumeric characters such as the underscore (_). Internally, a three-character thread ID is appended to the base name for each thread that is created based on the `MAXTHREADS` option of the `ADD REPLICAT` command. The resulting names cannot be duplicated for any other Replicat group. For example, if a coordinated Replicat group named `fin` is created with a `MAXTHREADS` of 50 threads, the resulting thread names could span from `fin000` through `fin050`, assuming those are the IDs specified in the `MAP` statements. Thus, no other Replicat group can be named `fin000` through `fin0050`. See the following rule for more information.
- You can include a number in a group name, but it is not recommended that a name end in any numerals. Understand that using a numeric value at the end of a group name (such as `fin1`) can cause duplicate report file names and errors, because the writing process appends a number to the end of the group name when generating a report. In addition, ending a group name with numeric values is not recommended when running Replicat in coordinated mode. Because numeric thread IDs are appended to a group name internally, if the base group name also ends in a number it can make the output of informational commands more complicated to analyze. Thread names could be confused with the names of other Replicat groups if the numeric appendages satisfy wildcards. Duplicate report file names also can occur. It may be more practical to put a numeric value at the beginning of a group name, such as `1_fin`, `1fin`, and so forth.
- Any character can be used in the name of a process, so long as the character set of the local operating system supports it, and the operating system allows that character to be in a file name. This is because a group is identified by its associated checkpoint file and parameter file.
- The following characters are not allowed in the name of a process:

  `\ / : * ? " < > |`
- On HP UX, Linux, and Solaris, it is possible to create a file name with a colon (:) or an asterisk (*), although it is not recommended.

- In general, process names and parameter file names are not case-sensitive within Oracle GoldenGate. For example, `finance`, `Finance`, and `FINANCE` are all considered to be the same. However, on Linux, the process name (and its parameter file name if explicitly defined in the `ADD` command) must be all uppercase or all lowercase. Mixed-case names specified for processes and parameter files will result in errors when starting the process.
- Use only one word for a name.
- The word `port` can be the full name for a process or parameter file. However, the string `port` can be part of a name.

Choosing File Names

Captured data must be processed into a series of files called a trail, where it is stored for processing by the next Oracle GoldenGate process downstream. The basic configuration is:

- A local trail on the source system
- A remote trail on the target system

The actual trail name contains only two characters, such as `./dirdat/tr`. Oracle GoldenGate appends this name with a nine-digit sequence number whenever a new file is created, such as `./dirdat/aa000000002`. It is recommended that you establish naming conventions for trails, because they are linked to Oracle GoldenGate processes and may need to be identified for the purposes of troubleshooting.

On Windows systems, if the name of any directory in the trail path name begins with a number, the path must be specified with forward slashes, not backward slashes, when listing the trail in a parameter file. For more information, see Specifying Filesystem Path Names in Parameter Files on Windows Systems.

See What is a Trail? for more information about Oracle GoldenGate trails.

Creating a Parameter File for Online Extraction

Follow these instructions to create a parameter file for an online Extract group. A parameter file is not required for an alias Extract group.

1. In GGSCI on the source system, issue the following command.

   ```
   EDIT PARAMS name
   ```

   Where:

   `name` is either the name of the Extract group that you created with the `ADD EXTRACT` command or the fully qualified name of the parameter file if you defined an alternate location when you created the group.

2. Enter the parameters in the order shown in the following table, starting a new line for each parameter statement. Some parameters apply only for certain configurations.

Parameter	Description
`EXTRACT` *`group`* • *`group`* is the name of the Extract group that you created with the `ADD EXTRACT` command.	Configures Extract as an online process with checkpoints.

Parameter	Description
[SOURCEDB *dsn* | *container* | *catalog*] [, USERIDALIAS *alias options* | , USERID *user*, *options*]	Specifies database connection information. SOURCEDB specifies the source data source name (DSN). See for more information. USERID and USERIDALIAS specify database credentials if required. The database connection can be omitted if the group is a data pump on an intermediary system that does not have a database. In this case, there can be no column mapping or conversion performed.
RMTHOSTOPTIONS *host*, MGRPORT *port*, [, ENCRYPT *algorithm* KEYNAME *key_name*]	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP. Only required when sending data over IP to a remote system (if ADD RMTTRAIL was used to create the trail). Not required if the trail is on the local system (if ADD EXTTRAIL was used). Not valid for a passive Extract group.
ENCRYPTTRAIL *algorithm*	Encrypts all trails that are specified after this entry.
DECRYPTTRAIL	(For a data pump) Decrypts the data in the input trail. Use only if the data pump must process the data before writing it to the output trail.
RMTTRAIL *pathname* | EXTTRAIL *pathname* • Use RMTTRAIL to specify the relative or fully qualified name of a remote trail created with the ADD RMTTRAIL command. • Use EXTTRAIL to specify the relative or fully qualified name of a local trail created with the ADD EXTTRAIL command (to be read by a data pump or VAM-sort Extract).	Specifies a trail. If specifying multiple trails, follow each designation with the appropriate TABLE statements. EXTTRAIL is not valid for a passive Extract group. If trails or files will be of different versions, use the FORMAT option of RMTTRAIL or EXTTRAIL. See EXTTRAILin *Parameters and Functions Reference for Oracle GoldenGate*
LOGALLSUPCOLS	Use when using integrated Replicat for an Oracle target, or when using Conflict Detection and Resolution (CDR) support. Writes the before images of scheduling columns to the trail. (Scheduling columns are primary key, unique index, and foreign key columns.) See LOGALLSUPCOLS in *Parameters and Functions Reference for Oracle GoldenGate*.
SOURCECATALOG	Specifies a default container in an Oracle multitenant container database or SEQUENCE statements. Enables the use of two-part names (*schema.object*) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of TABLE or SEQUENCE parameters.
SEQUENCE [*container*.]*owner*.*sequence*;	Specifies the fully qualified name of an Oracle sequence to capture. Include the container name if the database is a multitenant container database (CDB).
TABLE [*container*. | *catalog*.]owner.*object*;	Specifies the fully qualified name of an object or a fully qualified wildcarded specification for multiple objects. If the database is an Oracle multitenant container database, the object name must include the name of the container or catalog unless SOURCECATALOG is used. See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.

Parameter	Description

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`TABLEEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific objects from a wildcard specification in the associated `TABLE` statement.

3. Enter any appropriate optional Extract parameters listed in the Oracle GoldenGate Parameters in *Parameters and Functions Reference for Oracle GoldenGate*.
4. Save and close the parameter file.

Parameter	Description

`VAM` *`library`*`,`
`PARAMS ('`*`param`*`'`
`[, '`*`param`*`'] [, ...])`
	Supplies the name of the library and parameters that must be passed to the Oracle GoldenGate API, such as the name of the TAM initialization file and the program that interacts with the library as the callback library.
	Example:
	`VAM vam.dll, PARAMS ('inifile', 'vamerge1.ini', 'callbacklib', 'extract.exe')`

	NA

Creating an Online Replicat Group

Before creating a Replicat group, you should evaluate which of the Replicat modes is appropriate for your environment: *classic mode* (also known as *nonintegrated mode* in Oracle environments), *coordinated mode, and integrated mode*.

About the Global Watermark

A clean shutdown of a Replicat ensures that all threads stop at the same transaction boundary in the trail, known as the *global watermark*. This is defined as the synchronized point where all records before this position were either committed or ignored by all of their respective threads. If a clean shutdown is not possible, you can use the `SYNCHRONIZE REPLICAT` command to return all of the threads to the position of the thread that made the most recent checkpoint. This command is valid for coordinated, integrated, and parallel Replicats. See Synchronizing Threads After an Unclean Stop for more information about recovering a coordinated Replicat group.

> **Note:**
>
> Coordinated Replicat is an online process only. Do not use it to perform initial loads.

Creating the Replicat Group

To create an online Replicat group, run GGSCI on the target system and issue the `ADD REPLICAT` command. Separate all command arguments with a comma.

```
ADD REPLICAT group, EXTTRAIL path
[, {INTEGRATED | COORDINATED [MAXTHREADS number]}]
[, BEGIN start_point | , EXTSEQNO seqno, EXTRBA rba]
[, CHECKPOINTTABLE owner.table]
[, NODBCHECKPOINT]
[, PARAMS path]
[, REPORT path]
```

Where:

- `group` is the name of the Replicat group. A group name is required. See Naming Conventions for Processes for Oracle GoldenGate naming conventions.
- `EXTTRAIL path` is the relative or fully qualified name of the trail that you defined with the `ADD RMTTRAIL` command.
- `INTEGRATED` specified that this Replicat group will operate in integrated mode. This mode is available for Oracle databases..
- `COORDINATED` specifies that this Replicat group will operate in coordinated mode. `MAXTHREADS` specifies the maximum number of threads allowed for this group. Valid values are from 1 through 500. `MAXTHREADS` is optional. The default number of threads without `MAXTHREADS` is 25.

  > **Note:**
  >
  > Each Replicat thread is considered a Replicat group in the context of the `MAXGROUPS` parameter. `MAXGROUPS` controls the maximum number of process groups allowed in the Oracle GoldenGate instance. `MAXTHREADS` plus the number of other process groups in the Oracle GoldenGate instance must not exceed the value set with `MAXGROUPS` (default is 1000).

- `BEGIN start_point` defines an online Replicat group by establishing an initial checkpoint and start point for processing. Use one of the following:
  - `NOW` to begin replicating changes timestamped at the point when the `ADD REPLICAT` command is executed to create the group.
  - `YYYY-MM-DD HH:MM[:SS[.CCCCCC]]` as the format for specifying an exact timestamp as the begin point.
- `EXTSEQNO seqno, EXTRBA rba` specifies the sequence number of the file in a trail in which to begin reading data and the relative byte address within that file. By default, processing begins at the beginning of a trail unless this option is used. For the sequence number, specify the number, but not any zeroes used for padding. For example, if the trail file is `c:\ggs\dirdat\aa000000026`, specify `EXTSEQNO 26`. Contact Oracle Support before using this option.
- `CHECKPOINTTABLE owner.table` specifies the owner and name of a checkpoint table other than the default specified in the `GLOBALS` file. To use this argument, you must add the checkpoint table to the database with the `ADD CHECKPOINTTABLE` command (see About Checkpoint Table).
- `NODBCHECKPOINT` specifies that this Replicat group will not use a checkpoint table.
- `PARAMS path` is required if the parameter file for this group will be stored in a location other than the `dirprm` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.

- `REPORT` *`path`* is required if the process report for this group will be stored in a location other than the `dirrpt` sub-directory of the Oracle GoldenGate directory. Specify the fully qualified name. The default location is recommended.

Example 10-1   Creating an Online Replicat Group

This example creates an online Replicat group named `finance` and specifies a trail of `c:\ggs\dirdat\rt`. The parameter file is stored in the alternate location of `\ggs\params`, and the report file is stored in its default location.

```
ADD REPLICAT finance, EXTTRAIL c:\ggs\dirdat\rt, PARAMS \ggs\params
```

Creating a Parameter File for Online Replication

Follow these instructions to create a parameter file for an online Replicat group.

1. In GGSCI on the target system, issue the following command.

   ```
   EDIT PARAMS name
   ```

   Where:

   `name` is either the name of the Replicat group that you created with the `ADD REPLICAT` command or the fully qualified name of the parameter file if you defined an alternate location when you created the group.

2. Enter the parameters listed in the table below in the order shown, starting a new line for each parameter statement.

Table 10-1   Online Change-Replication Parameters

Parameter	Description
`REPLICAT` *`group`* • *`group`* is the name of the Replicat group that you created with the `ADD REPLICAT` command.	Configures Replicat as an online process with checkpoints.
`{SOURCEDEFS` *`path`*`} | ASSUMETARGETDEFS` • Use `SOURCEDEFS` if the source and target tables have different definitions. Specify the source data-definitions file generated by `DEFGEN`. • Use `ASSUMETARGETDEFS` if the source and target tables have the same definitions.	Specifies how to interpret data definitions. For Oracle databases that use multi-byte character sets, you must use `SOURCEDEFS` (with a DEFGEN-generated definitions file) if the source semantics setting is in bytes and the target is in characters. This is required even when the source and target data definitions are identical.

`[DEFERAPPLYINTERVAL` *`n unit`*`]`
• *`n`* is a numeric value for the amount of time to delay before applying transactions. Minimum is set by the `EOFDELAY` parameter. Maximum is seven days.
• *`unit`* can be:
`S | SEC | SECS | SECOND | SECONDS | MIN | MINS | MINUTE | MINUTES | HOUR | HOURS | DAY | DAYS`
	Optional. Specifies an amount of time for Replicat to wait before applying its transactions to the target system.

Table 10-1 (Cont.) Online Change-Replication Parameters

Parameter	Description

`[TARGETDB` *`dsn`* `|` *`container`* `|` *`catalog`*`]`
`[, USERIDALIAS` *`alias options`* `|`
`, USERID` *`user`*`,` *`options`*`]`
	Specifies database connection information.
	`TARGETDB` specifies the target datasource name (DSN). See `TARGETDB` in *Parameters and Functions Reference for Oracle GoldenGate*for more information .
	`USERID` and `USERIDALIAS` specify database credentials if required.

`HANDLECOLLISIONS`	Specifies collision handling. Use only if you are performing an initial load concurrently with starting online processing and the source database will remain active during the load. `HANDLECOLLISIONS` resolves the results of the copy with the ongoing replicated transactional changes. It resolves insert operations for which the row already exists and update and delete operations for which the row does not exist. It can be used globally for all `MAP` statements in a parameter file or within a `MAP` statement, or both.
`SOURCECATALOG`	Specifies a default container in a source Oracle multitenant container database. Enables the use of two-part names (*`schema.object`*) where three-part names otherwise would be required for those databases. You can use multiple instances of this parameter to specify different default containers or catalogs for different sets of `MAP` parameters.

`MAP [`*`container`*`. |` *`catalog`*`.]`*`owner.object`*`,`
`TARGET` *`owner.object`*`[, DEF` *`template`*`]`
`[THREAD (`*`thread_ID`*`)]`
`[THREADRANGE (thread_range[,` *`column_list`*`])]`
`[COORDINATED]`
`;`
	Specifies a relationship between a source object or objects and a target object or objects. `MAP` specifies the source object, and `TARGET` specifies the target object.
	For the source object, specify the fully qualified name of the object or a fully qualified wildcarded specification for multiple objects. For an Oracle multitenant container database the source object name must include the name of the container or catalog unless `SOURCECATALOG` is used.
	For the target object, specify only the *`owner.object`* components of the name, regardless of the type of database. Replicat can only connect to one Oracle container. Use a separate Replicat process for each container or catalog to which you want to apply data.
	See Specifying Object Names in Oracle GoldenGate Input for guidelines for specifying object names in parameter files.
	The `THREAD`, `THREADRANGE`, and `COORDINATED` options are valid for Replicat when in coordinated mode. They enable you to partition the workload to one or more specific Replicat threads. See in *Parameters and Functions Reference for Oracle GoldenGate*for syntax and usage.
	The `DEF` option specifies a definitions template.

`CATALOGEXCLUDE`
`SCHEMAEXCLUDE`
`MAPEXCLUDE`
`EXCLUDEWILDCARDOBJECTSONLY`
	Parameters that can be used in conjunction with one another to exclude specific source objects from a wildcard specification in the associated `MAP` statement.

1. Enter any appropriate optional Replicat parameters listed in the *Parameters and Functions Reference for Oracle GoldenGate*.

2. Save and close the file.

> **Note:**
>
> If using integrated Replicat for Oracle, see Understanding Replicat Processing in Relation to Parameter Changes for important information about making configuration changes to Replicat once processing is started.

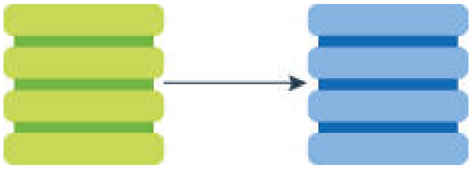
Using Oracle GoldenGate for Live Reporting

This chapter describes the usage of Oracle GoldenGate for live reporting.

Overview of the Reporting Configuration

The most basic Oracle GoldenGate configuration is a one-to-one configuration that replicates in one direction: from a source database to a target database that is used only for data retrieval purposes such as reporting and analysis. Oracle GoldenGate supports transfer of data to Oracle or Non-Oracle databases, with capabilities for filtering and conversion on either system in the configuration (support varies by database platform).

Oracle GoldenGate is ideal for creating a reporting environment because the target can be optimized for reporting, while allowing the source to be optimized for OLTP workloads. This includes adding additional indexes or materialized views on the target database to allow faster execution of queries. Oracle GoldenGate can also pull metadata from the source database to help track how the records changed, when the record changed, who changed it, and even track the history of how column values changed.



Oracle GoldenGate supports different reporting topologies that enable you to custom-configure the processes based on your requirements for scalability, availability, and performance. This section contains things to take into consideration when choosing a reporting configuration.

Filtering and Conversion

Data filtering and data conversion both add overhead, and these activities are sometimes prone to configuration errors. If Oracle GoldenGate must perform a large amount of filtering and conversion, consider using one or more data pumps to handle this work. You can use Replicat for this purpose, but you would be sending more data across the network that way, as it will be unfiltered. You can split filtering and conversion between the two systems by dividing it between the data pump and Replicat.

To filter data, you can use:

- A `FILTER` or `WHERE` clause in a `TABLE` statement (Extract) or in a `MAP` statement (Replicat)
- A SQLEXEC can perform a query or execute a stored procedure on the database. The values returned can then be used to evaluate a `FILTER` clause.

- User exits

To transform data, you can use:

- The Oracle GoldenGate conversion functions. See Column Conversion Functions in the *Parameters and Functions Reference for Oracle GoldenGate*.
- Metadata from the source database or trail. See GETENV function.
- A user exit from the Extract or Replicat process that applies rules from an external transformation solution, then returns the manipulated data to Oracle GoldenGate.
- Replicat to deliver data directly to an ETL solution or other transformation engine.

For more information about Oracle GoldenGate's filtering and conversion support, see:

- Mapping and Manipulating Data
- Customizing Oracle GoldenGate Processing

Read-only vs. High Availability

The Oracle GoldenGate live reporting configuration supports a read-only target. See Configuring Bi-Directional Replication if the target in this configuration will also be used for transactional activity in support of high availability.

Additional Information

The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

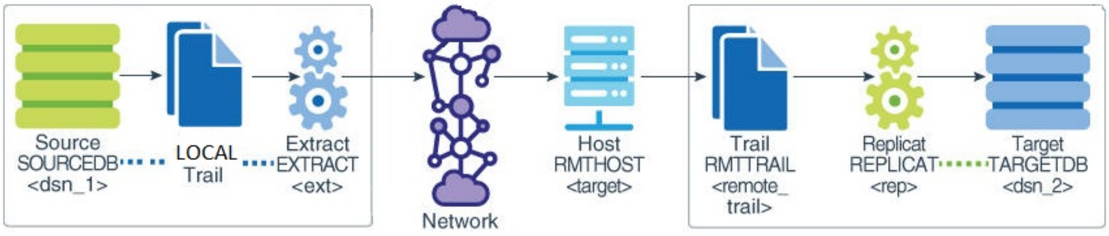
- For additional system requirements, process configuration, and database setup requirements, see Install and Prepare.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see Configuring Online Change Synchronization.
- For additional tuning options for Oracle GoldenGate, see Tuning the Performance of Oracle GoldenGate.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Standard Reporting Configuration

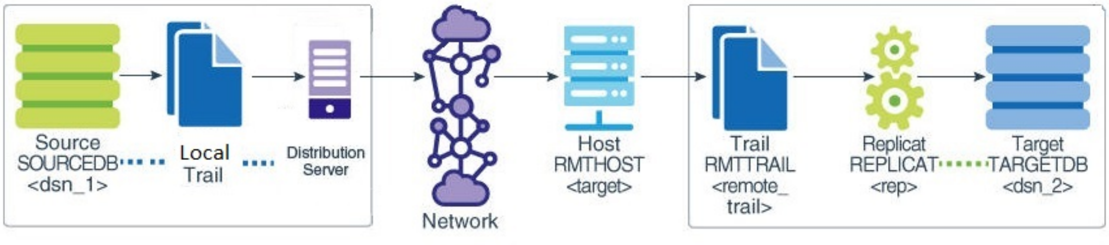
In the standard Oracle GoldenGate configuration, one Extract group sends captured data over TCP/IP to a trail on the target system, where it is stored until processed by one Replicat group.

Refer to the following figure for a representation of the objects you will be creating in Classic Architecture.

Figure 10-1 Configuration Elements for Creating a Standard Reporting Configuration



Refer to the following figure for a representation of the objects you will be creating in Microservices Architecture.



Source System

Configure the Manager process and Extract group on the source system.

To Configure the Manager Process

On the source, configure the Manager process.

To Configure the Extract Group

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext*.

   ```
   ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail to be created on the target system.

   ```
    ADD EXTTRAIL local_trail, EXTRACT ext
   ```

   Use the `EXTRACT` argument to link this trail to the Extract group.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the Extract group. Include the following parameters plus any others that apply to your database environment.

   For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Log  all of the supplementally logged columns if using integrated
   Replicat
   LOGALLSUPCOLS
   ENCRYPTTRAIL algorithm
   EXTTRAIL local_trail
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

To send the trail files to the target

1. For Classic Architecture, use the following command option:

   ```
   ADD EXTRACT ext_pmp, EXTTRAILSOURCE remote_trail, BEGIN time [option[,...]]
   ```

   For Microservices Architecture, use the following command:

   ```
   ADD DISTPATH path-name
             SOURCE source-uri
             TARGET target-uri|
            [TARGETTYPE ( MANAGER | COLLECTOR | RECVSRVR ) ]|
   ```

2. Run the following command to add the remote trail:

   ```
   ADD RMTTRAIL remote_trail, EXTRACT ext_pmp
   ```

3. Use the `EDIT PARAMS` command to create a parameter file for the target.

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process

1. On the target, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Replicat Group

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions. All Replicat groups can use the same checkpoint table.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep*.

   ```
   ADD REPLICAT rep
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail
   , BEGIN time
   ```

   Use the `EXTTRAIL` argument to link the Replicat group to the remote trail.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Replicat group:
   REPLICAT rep
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_2][, USERIDALIAS alias]
   ```

```
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Note:

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Creating a Reporting Configuration with a Data Pump on the Source System

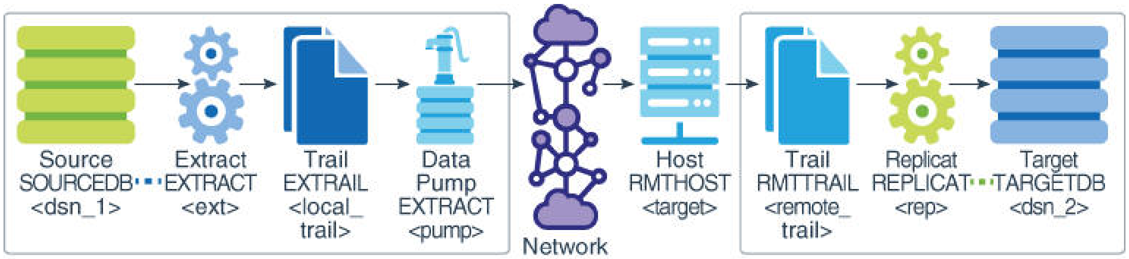
You can add a data pump on the source system to isolate the primary Extract from TCP/IP functions, to add storage flexibility, and to offload the overhead of filtering and conversion processing from the primary Extract.

In this configuration, the primary Extract writes to a local trail on the source system. A local data pump reads that trail and moves the data to a remote trail on the target system, which is read by Replicat.

You can, but are not required to, use a data pump to improve the performance and fault tolerance of Oracle GoldenGate.

Here's a representation of the objects, you will be creating.

Figure 10-2 Configuration Elements for Replicating to One Target with a Data Pump



Source System

Configure the Manager process and Extract group on the source system.

To Configure the Manager Process

1. On the source, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Primary Extract Group

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext*.

   ```
   ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail. The primary Extract writes to this trail, and the data-pump Extract reads it.

   ```
   ADD EXTTRAIL local_trail, EXTRACT ext
   ```

   Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump group reads it.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][,USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   -- Specify the local trail that this Extract writes to and
   -- encryption algorithm:
   ENCRYPTTRAIL algorithm
   EXTTRAIL local_trail
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

To Configure the Data Pump Extract Group

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump*.

   ```
   ADD EXTRACT pump, EXTTRAILSOURCE local_trail, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the target system.

   ```
   ADD RMTTRAIL remote_trail, EXTRACT pump
   ```

   Use the `EXTRACT` argument to link the remote trail to the data pump group. The linked data pump writes to this trail.

See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the target system:
ENCRYPTTRAIL alogrithm
RMTTRAIL remote_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process

1. On the target, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Replicat Group

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep*.

```
ADD REPLICAT rep
[, INTEGRATED | PARALLEL | COORDINATED [MAXTHREADS number] |  ]
, EXTTRAIL remote_trail
, BEGIN time
( SPECIALRUN |
             ( EXTFILE       file-name  |
               EXTTRAIL        trail-name )
              [ BEGIN           ( NOW | begin-datetime ) |
               EXTSEQNO        trail-sequence-number [ EXTRBA trail-offset-
number ] ]
              [ CHECKPOINTTABLE table-name | NODBCHECKPOINT ] )
              [ DESC            description ]
              [ CRITICAL        [ YES | NO ]   ]
              [ PROFILE         profile-name
              [ AUTOSTART       [ YES | NO ]
```

```
  [ DELAY            delay-number ] ]
[ AUTORESTART      [ YES | NO ]
  [ RETRIES          retries-number ]
  [ WAITSECONDS      wait-number    ]
  [ RESETSECONDS     reset-number   ]
  [ DISABLEONFAILURE [ YES | NO ]      ] ] ]
```

Use the `EXTTRAIL` argument to link the Replicat group to the remote trail.

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

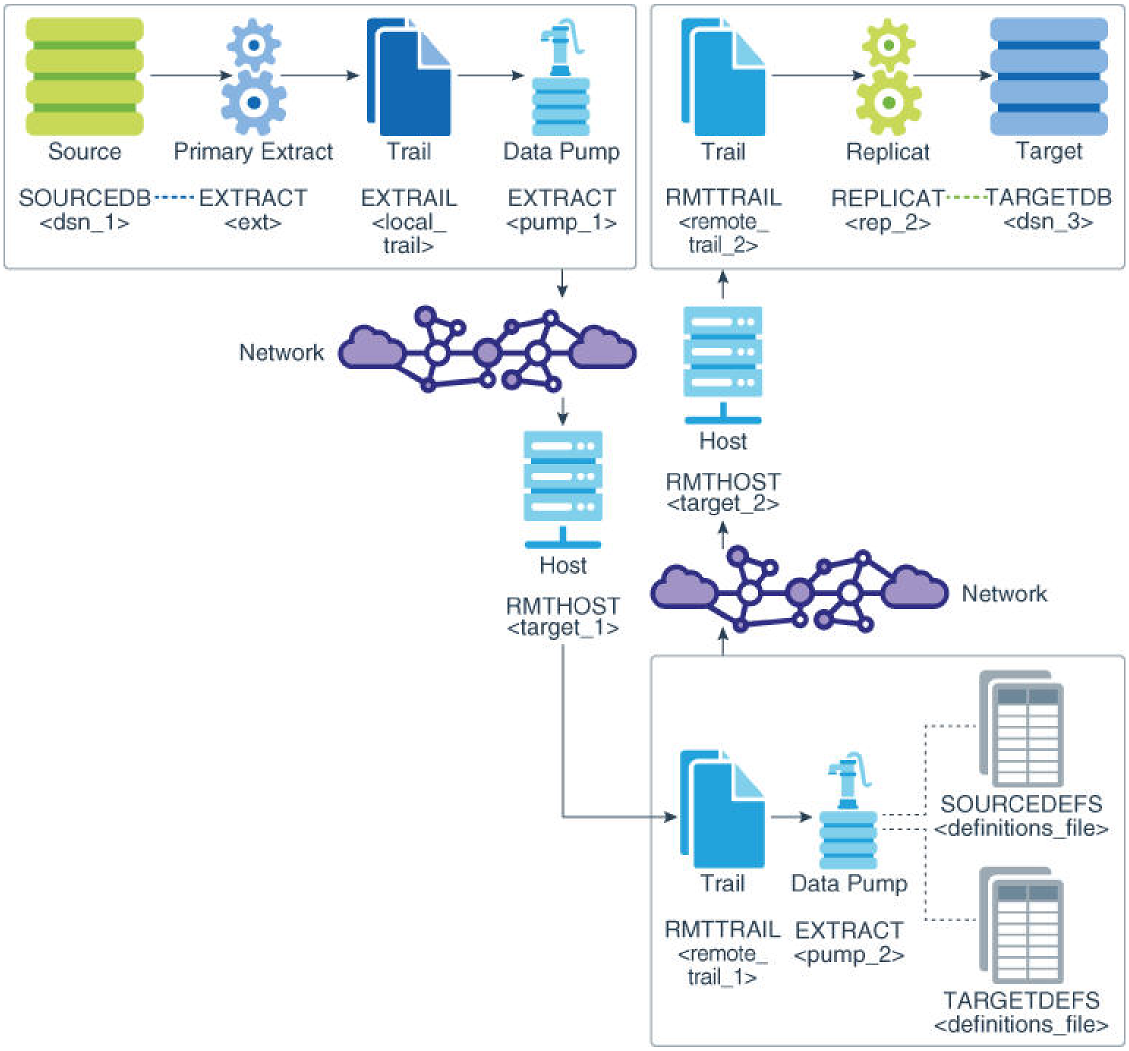
Note:

For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Creating a Reporting Configuration with a Data Pump on an Intermediary System

You can use an intermediary system as a transfer point between the source and target systems. In this configuration, a data pump on the source system sends captured data to a remote trail on the intermediary system. A data pump on the intermediary system reads the trail and sends the data to a remote trail on the target. A Replicat on the target reads the remote trail and applies the data to the target database.

Figure 10-3 Configuration Elements for Replication through an Intermediary System



When considering this topology, take note of the following:

- This configuration is practical if the source and target systems are in different networks and there is no direct connection between them. You can transfer the data through an intermediary system that can connect to both systems.
- This configuration can be used to add storage flexibility to compensate for deficiences on the source or target.
- This configuration can be used to perform data filtering and conversion if the character sets on all systems are identical. If character sets differ, the data pump cannot perform conversion between character sets, and you must configure Replicat to perform the conversion and transformation on the target.
- This configuration is a form of cascaded replication. However, in this configuration, data is not applied to a database on the intermediary system. See Creating a Cascading Reporting Configuration to include a database on the intermediary system in the Oracle GoldenGate configuration.

Source System

Here are the objects you will be creating.

To Configure the Manager Process

1. On the source, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Primary Extract Group on the Source

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext*.

   ```
   ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail. The primary Extract writes to this trail, and the data-pump Extract reads it.

   ```
   ADD EXTTRAIL local_trail, EXTRACT ext
   ```

   Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump group reads it.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   -- Specify the local trail that this Extract writes to and
   -- encryption algorithm:
   ENCRYPTTRAIL algorithm
   EXTTRAIL local_trail
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

To Configure the Data Pump on the Source

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_1*.

   ```
   ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail, BEGIN time
   ```

Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail. For a local Extract, you must use `EXTTRAIL` not `RMTTRAIL`.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the intermediary system.

   ```
   ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
   ```

   Use the `EXTRACT` argument to link the remote trail to the *pump_1* data pump group. The linked data pump writes to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the *pump_1* data pump. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pump_1
   -- Specify database login information:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the intermediary system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify remote trail and encryption algorithm on intermediary system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL remote_trail_1
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

Intermediary System

Configure the Manager process and data pump on the intermediary system.

To Configure the Manager Process on the Intermediary System

1. On the intermediary system, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Data Pump on the Intermediary System

1. On the intermediary system, use the `ADD EXTRACT` command to create a data-pump group. For documentation purposes, this group is called *pump_2*.

   ```
   ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_1, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and specify the name of the trail that you created on this system

2. On the intermediary system, use the `ADD RMTTRAIL` command to specify a remote trail on the target system.

   ```
   ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
   ```

   Use the `EXTRACT` argument to link the remote trail to the *pump_2* data pump. The linked data pump writes to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the intermediary system, use the `EDIT PARAMS` command to create a parameter file for the *pump_2* data pump. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pump_2
   -- Note that no database login parameters are required in this case.
   -- Specify the target definitions file if SOURCEDEFS was used:
   TARGETDEFS full_pathname
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the target system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify the remote trail and encryption algorithm on the target system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL remote_trail_2
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

Target System

Configure the Manager process and Replicat group on the target system.

To Configure the Manager Process on the Target

1. On the target system, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group on the Target

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.
2. On the target, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep*.

   ```
   ADD REPLICAT rep
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_2,
   , BEGIN time
   ```

Use the `EXTTRAIL` argument to link the Replicat group to the trail on this system.

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep
-- Specify database login information as needed for the database:
[TARGETDB dsn_2][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

> **Note:**
>
> For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

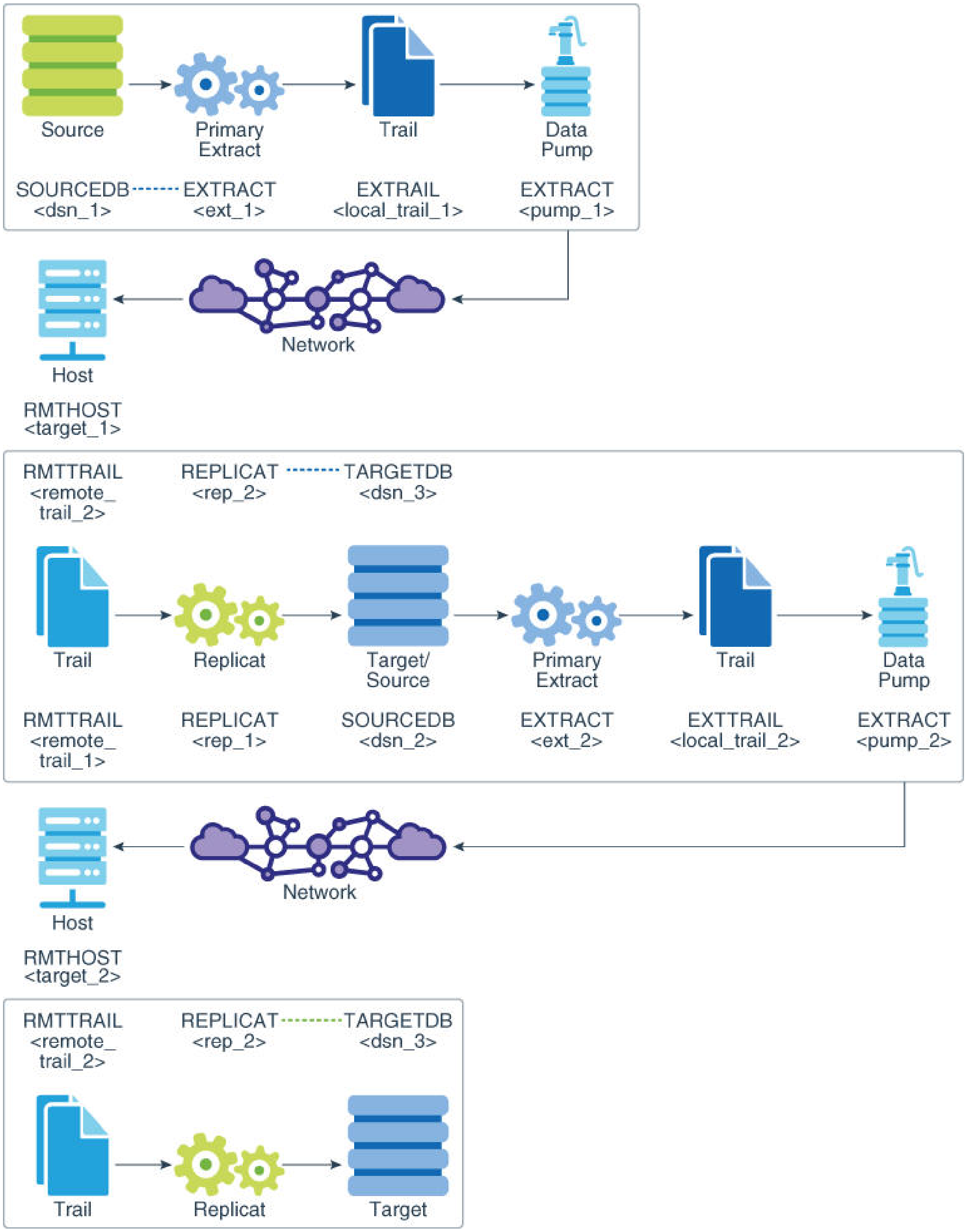
Creating a Cascading Reporting Configuration

Oracle GoldenGate supports cascading synchronization, where Oracle GoldenGate propagates data changes from the source database to a second database, and then on to a third database. In this configuration:

- A primary Extract on the source writes captured data to a local trail, and a data pump sends the data to a remote trail on the second system in the cascade.
- On the second system, Replicat applies the data to the local database.
- Another primary Extract on that same system captures the data from the local database and writes it to a local trail.
- A data pump sends the data to a remote trail on the third system in the cascade, where it is applied to the local database by another Replicat.

> **Note:**
>
> See Creating a Reporting Configuration with a Data Pump on an Intermediary System if you do not need to apply the replicated changes to a database on the secondary system.

Figure 10-4   Cascading Configuration



Use this configuration if:

- One or more of the target systems does not have a direct connection to the source, but the second system can connect in both directions.
- You want to limit network activity from the source system.

- You are sending data to two or more servers that are very far apart geographically, such as from Chicago to Los Angeles and then from Los Angeles to servers throughout China.

When considering this topology, take note of the following:

- This configuration can be used to perform data filtering and conversion if the character sets on all systems are identical. If character sets differ, a data pump cannot perform conversion between character sets, and you must configure Replicat to perform the conversion and transformation on the target.
- On the second system, you must configure the Extract group to capture Replicat activity and to ignore local business application activity. The Extract parameters that control this behavior are `IGNOREAPPLOPS` and `GETREPLICATES`.

Source System

Refer to #unique_1014/unique_1014_Connect_42_BABBJAHA for a visual representation of the objects you will be creating.

To Configure the Manager Process on the Source

1. On the source, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Primary Extract Group on the Source

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext_1*.

   ```
   ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail.

   ```
   ADD EXTTRAIL local_trail_1, EXTRACT ext_1
   ```

   Use the `EXTRACT` argument to link this trail to the *ext_1* Extract group.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the *ext_1* Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext_1
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   -- Specify the local trail that this Extract writes to
   -- and encryption algorithm:
   ENCRYPTTRAIL algorithm
   ```

```
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pump on the Source

1. On the source, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_1*.

   ```
   ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.
2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the second system in the cascade.

   ```
   ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
   ```

   Use the `EXTRACT` argument to link the remote trail to the *pump_1* data pump group. The linked data pump writes to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.
3. On the source, use the `EDIT PARAMS` command to create a parameter file for the *pump_1* data pump. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pump_1
   -- Specify database login information if using NOPASSTHROUGH:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of second system in cascade
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify the remote trail and encryption algorithm on the second system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL remote_trail_1
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

Second System in the Cascade

Configure the Manager process, Replicat group, and data pump on the second system in the cascade.

To Configure the Manager Process on the Second System

1. On the second system, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group on the Second System

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.
2. On the second system, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep_1*.

   ```
   ADD REPLICAT rep_1
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_1,
   , BEGIN time
   ```

   Use the `EXTTRAIL` option to link the *rep_1* group to the remote trail *remote_trail_1* that is on the local system.

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.
3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Replicat group:
   REPLICAT rep_1
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_2][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

   > **Note:**
   >
   > For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

To Configure an Extract Group on the Second System

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext_2*.

   ```
   ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   [option[, ...]]
   ```

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the second system, use the `ADD EXTTRAIL` command to specify a local trail that will be created on the third system.

   ```
   ADD EXTTRAIL local_trail_2, EXTRACT ext_2
   ```

   Use the `EXTRACT` argument to link this local trail to the *ext_2* Extract group.

3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the *ext_2* Extract group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext_2
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_2][, USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   -- Specify the local trail that this Extract writes to
   -- and encryption algorithm:
   ENCRYPTTRAIL algorithm
   EXTTRAIL local_trail_2
   -- Ignore local DML, capture Replicat DML:
   IGNOREAPPLOPS
   GETREPLICATES
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

> **Note:**
>
> If replicating DDL operations, `IGNOREAPPLOPS, GETREPLICATES` functionality is controlled by the `DDLOPTIONS` parameter.

To Configure the Data Pump on the Second System

1. On the second system, use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_2*.

   ```
   ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and specify the name of the local trail.

2. On the second system, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the third system in the cascade.

   ```
   ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
   ```

Use the `EXTRACT` argument to link the remote trail to the *pump_2* data pump group. The linked data pump writes to this trail.

See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the second system, use the `EDIT PARAMS` command to create a parameter file for the *pump_2* data pump. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_2
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of third system in cascade
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the third system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Third System in the Cascade

Configure the Manager process and Replicat group on the third system in the cascade.

To Configure the Manager Process

1. On the third system, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Group

1. On the third system, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.
2. On the third system, use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep_2*.

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2,
, BEGIN time
```

Use the `EXTTRAIL` option to link the *rep_2* group to the *remote_trail_2* trail.

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the third system, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Include the following parameters plus any others that apply to your

database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Replicat group:
REPLICAT rep_2
-- Specify database login information as needed for the database:
[TARGETDB dsn_3][, USERIDALIAS alias]
-- Specify error handling rules:
REPERROR (error, response)
-- Specify tables for delivery and threads if using coordinated Replicat:
MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
[, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
;
```

Note:

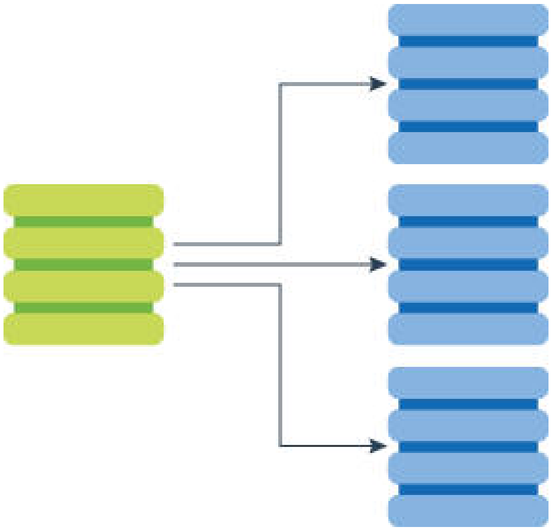
For DB2 for i, you may need to use the `ADD TRANDATA` command on the target tables if they are not already journaled. Alternatively, you could use the `STRJRNPF` command to assign the tables to the appropriate journal. If the target tables are not required to be replicated by Oracle GoldenGate, the `IMAGES(*AFTER)` option can be used with `STRJRNPF`. Since Oracle GoldenGate operates using transactions, all tables must be journaled to support transactions and this is not the default with DB2 for i.

Using Oracle GoldenGate for Real-time Data Distribution

This chapter describes the usage of Oracle GoldenGate for real-time data distribution.

Overview of the Data-distribution Configuration

A data distribution configuration is a one-to-many configuration. Oracle GoldenGate supports synchronization of a source database to any number of target systems. Oracle GoldenGate supports like-to-like or Non-Oracle transfer of data, with capabilities for filtering and conversion on any system in the configuration (support varies by database platform).



Considerations for a Data-distribution Configuration

These sections describe considerations for a data-distribution configuration.

Fault Tolerance

For a data distribution configuration, the use of data pumps on the source system ensures that if network connectivity to any of the targets fails, the captured data still can be sent to the other targets. Use a primary Extract group and one data-pump Extract group for each target.

Filtering and Conversion

You can use any process to perform filtering and conversion. However, using the data pumps to perform filtering operations removes that processing overhead from the primary Extract group, and it reduces the amount of data that is sent across the network. See Mapping and Manipulating Data for filtering and conversion options.

Read-only vs. High Availability

The data distribution configuration supports read-only targets. See Configuring Oracle GoldenGate for Active-Active Configuration if any target in this configuration will also be used for transactional activity in support of high availability.

Additional Information

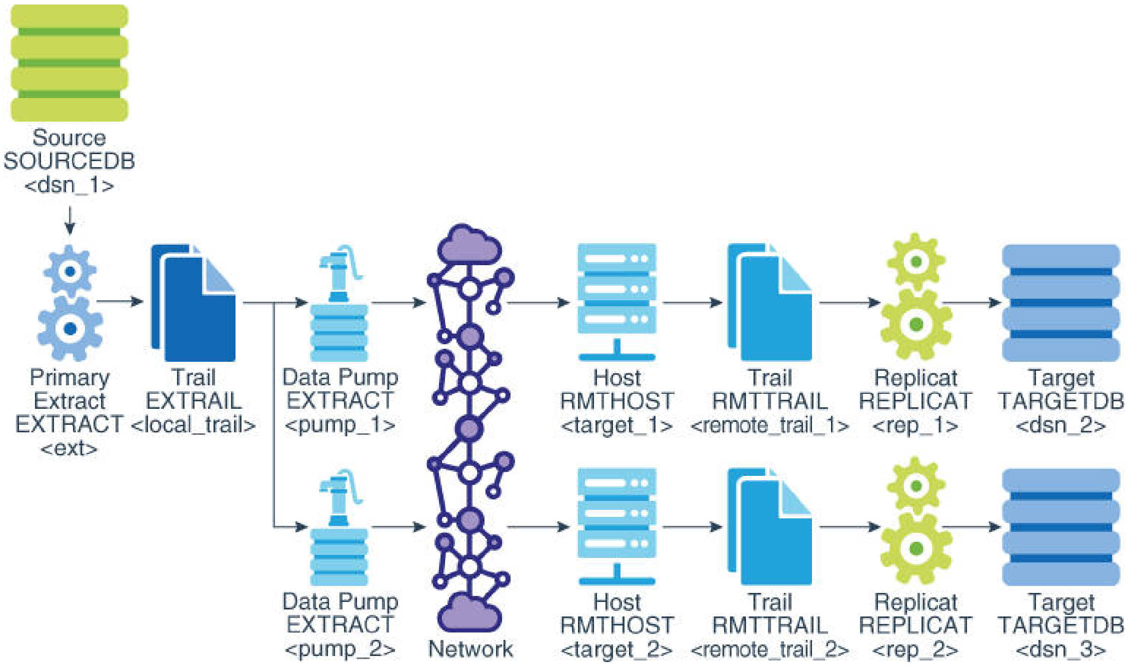
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see Install and Prepare.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see Configuring Online Change Synchronization.
- For additional tuning options for Oracle GoldenGate, see Tuning the Performance of Oracle GoldenGate.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Data Distribution Configuration

Refer to #unique_1024/unique_1024_Connect_42_BABCEHBA for a visual representation of the objects you will be creating.

Figure 10-5 Oracle GoldenGate Configuration Elements for Data Distribution



Source System

Configure the Manager process and primary Extract on the source system.

To Configure the Manager Process

1. On the source, configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the local trail.

To Configure the Primary Extract

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext*.

   ```
   ADD EXTRACT ext, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On the source, use the `ADD EXTTRAIL` command to create a local trail.

   ```
   ADD EXTTRAIL local_trail, EXTRACT ext
   ```

   Use the `EXTRACT` argument to link this trail to the primary Extract group. The primary Extract group writes to this trail, and the data pump groups read it

3. On the source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract group. Include the following parameters plus any others that apply to your

database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Use `EXTTRAIL` to specify the local trail.

To Configure the Data Pump Extract Groups

1. On the source, use the `ADD EXTRACT` command to create a data pump for each target system. For documentation purposes, these groups are called *pump_1* and *pump_2*.

   ```
   ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail, BEGIN time
   ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and supply the name of the local trail.

2. On the source, use the `ADD RMTTRAIL` command to specify a remote trail that will be created on each of the target systems.

   ```
   ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
   ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
   ```

   Use the `EXTRACT` argument to link each remote trail to a different data pump group. The linked data pump writes to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On the source, use the `EDIT PARAMS` command to create a parameter file for each of the data pumps. Include the following parameters plus any others that apply to your database environment.

   Parameter file for *pump_1*:

   ```
   -- Identify the data pump group:
   EXTRACT pump_1
   -- Specify database login information:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the first target system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target_1, MGRPORT port_number, ENCRYPT encryption_options
   ```

```
-- Specify remote trail and encryption algorithm on first target system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Parameter file for *pump_2*:

```
-- Identify the data pump group:
EXTRACT pump_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the second target system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS target_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on second target system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Target Systems

Configure the Manager process and Replicat groups on the target systems.

To Configure the Manager Process

1. On each target, configure the Manager process.
2. In each Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Groups

1. On each target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.
2. On each target, use the `ADD REPLICAT` command to create a Replicat group for the remote trail on that system. For documentation purposes, these groups are called *rep_1* and *rep_2*.

   Command on *target_1*:

   ```
   ADD REPLICAT rep_1
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_1, BEGIN time
   ```

Command on *target_2*:

```
ADD REPLICAT rep_2
[, INTEGRATED | COORDINATED [MAXTHREADS number]]
, EXTTRAIL remote_trail_2, BEGIN time
```

Use the `EXTTRAIL` argument to link the Replicat group to the correct trail.

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On each target, use the `EDIT PARAMS` command to create a parameter file for the Replicat group. Use the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   Parameter file for *rep_1*:

   ```
   -- Identify the Replicat group:
   REPLICAT rep_1
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_2][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

   Parameter file for *rep_2*:

   ```
   -- Identify the Replicat group:
   REPLICAT rep_2
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_3][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

   You can use any number of `MAP` statements for any given Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

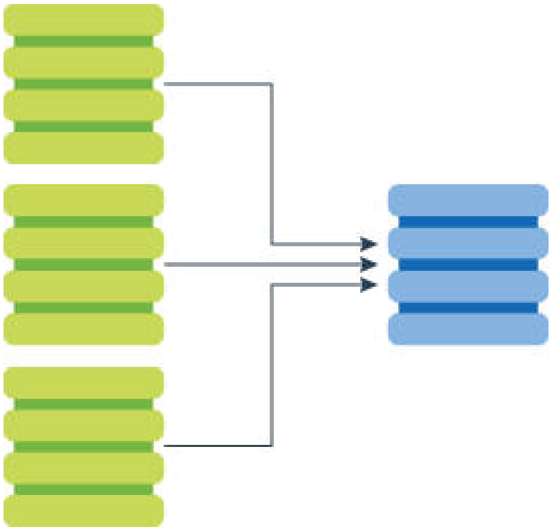
Configuring Oracle GoldenGate for Real-time Data Warehousing

This chapter describes how to configure Oracle GoldenGate for real-time data warehousing.

Overview of the Data Warehousing Configuration

A data warehousing configuration is a many-to-one configuration. Multiple source databases send data to one target warehouse database. Oracle GoldenGate supports like-to-like or Non-

Oracle transfer of data, with capabilities for filtering and conversion on any system in the configuration (support varies by database platform).



Considerations for a Data Warehousing Configuration

This section describes considerations for a data warehousing configuration.

Isolation of Data Records

This configuration assumes that each source database contributes different records to the target system. If the same record exists in the same table on two or more source systems and can be changed on any of those systems, conflict resolution routines are needed to resolve conflicts when changes to that record are made on both sources at the same time and replicated to the target table. See Managing Conflicts for more information about resolving conflicts.

Data Storage

You can divide the data storage between the source systems and the target system to reduce the need for massive amounts of disk space on the target system. This is accomplished by using a data pump on each source, rather than sending data directly from each Extract across the network to the target.

- A primary Extract writes to a local trail on each source.
- A data-pump Extract on each source reads the local trail and sends it across TCP/IP to a dedicated Replicat group.

Filtering and Conversion

If not all of the data from a source system will be sent to the data warehouse, you can use the data pump to perform the filtering. This removes that processing overhead from the primary Extract group, and it reduces the amount of data that is sent across the network. See Mapping and Manipulating Data for filtering and conversion options.

Additional Information

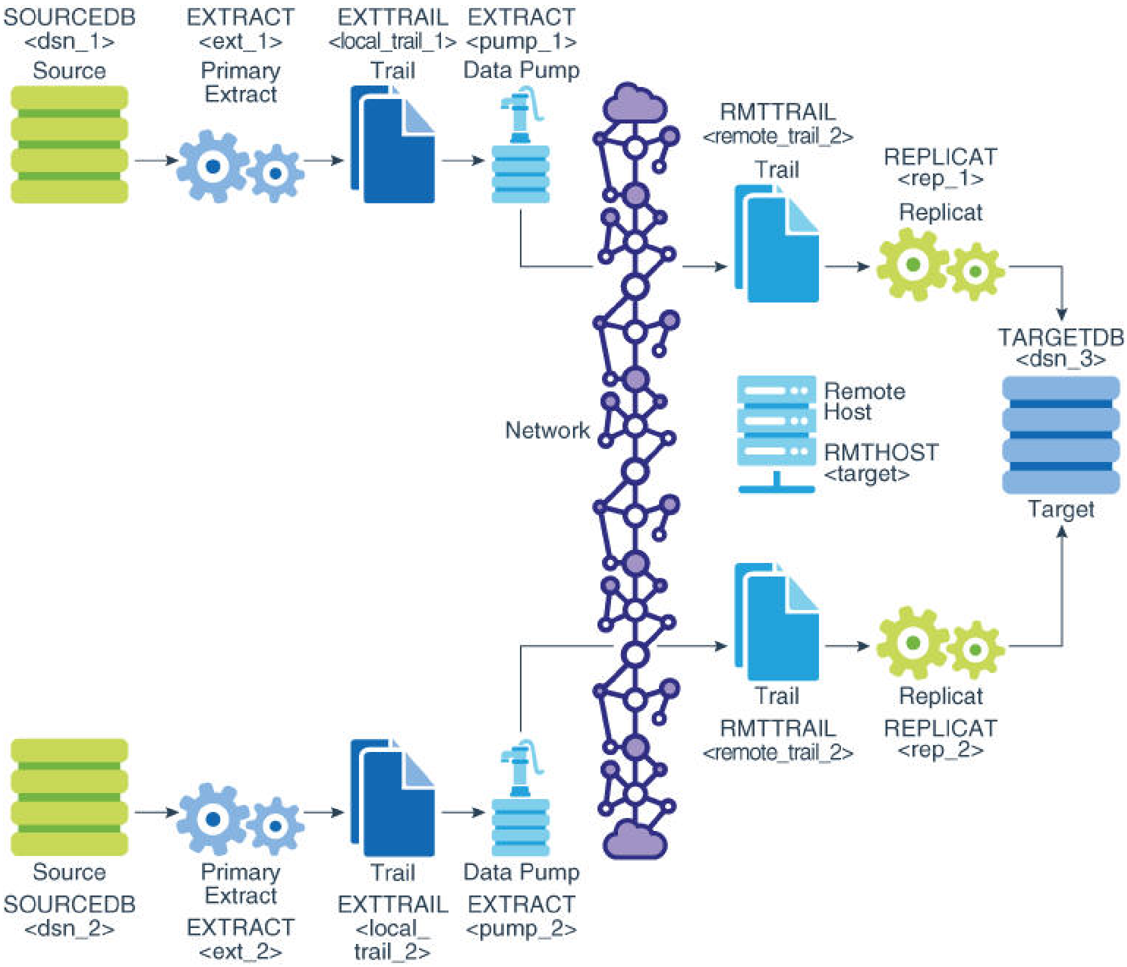
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see Install and Prepare.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see Configuring Online Change Synchronization.
- For additional tuning options for Oracle GoldenGate, see Tuning the Performance of Oracle GoldenGate.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Data Warehousing Configuration

Refer below for a visual representation of the objects you will be creating.

Figure 10-6 Configuration for Data Warehousing



Source Systems

Configure the Manager process and primary Extract groups for the source systems.

To Configure the Manager Process

1. On each source, configure the Manager process.
2. In each Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail on the local system.

To Configure the primary Extract Groups

1. On each source, use the `ADD EXTRACT` command to create a primary Extract group. For documentation purposes, these groups are called *ext_1* and *ext_2*.

   Command on *source_1*:

   ```
   ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   [option[, ...]]
   ```

   Command on *source_2*:

   ```
   ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. On each source, use the `ADD EXTTRAIL` command to create a local trail.

   Command on *source_1*:

   ```
   ADD EXTTRAIL local_trail_1, EXTRACT ext_1
   ```

   Command on *source_2*:

   ```
   ADD EXTTRAIL local_trail_2, EXTRACT ext_2
   ```

   Use the `EXTRACT` argument to link each Extract group to the local trail on the same system. The primary Extract writes to this trail, and the data-pump reads it.

3. On each source, use the `EDIT PARAMS` command to create a parameter file for the primary Extract. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   Parameter file for *ext_1*:

   ```
   -- Identify the Extract group:
   EXTRACT ext_1
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   ```

```
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_1
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

Parameter file for *ext_2*:

```
-- Identify the Extract group:
EXTRACT ext_2
-- Specify database login information as needed for the database:
[SOURCEDB dsn_2][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat or CDR
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_2
-- Specify tables and sequences to be captured:
SEQUENCE [container.|catalog.]owner.sequence;
TABLE [container.|catalog.]owner.table;
```

To Configure the Data Pumps

1. On each source, use the `ADD EXTRACT` command to create a data pump Extract group. For documentation purposes, these pumps are called *pump_1* and *pump_2*.

   Command on *source_1*:

   ```
   ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
   ```

   Command on *source_2*:

   ```
   ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
   ```

   Use `EXTTRAILSOURCE` as the data source option, and specify the name of the trail on the local system

2. On each source, use the `ADD RMTTRAIL` command to create a remote trail on the target.

   Command on *source_1*:

   ```
   ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
   ```

   Command on *source_2*:

   ```
   ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
   ```

   Use the `EXTRACT` argument to link each remote trail to a different data pump. The data pump writes to this trail over TCP/IP, and a Replicat reads from it.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. On each source, use the `EDIT PARAMS` command to create a parameter file for the data pump group. Include the following parameters plus any others that apply to your database environment.

   Parameter file for *pump_1*:

   ```
   -- Identify the data pump group:
   EXTRACT pump_1
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_1][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the target system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify the remote trail and encryption algorithm on the target system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL remote_trail_1
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

   Parameter file for *pump_2*:

   ```
   -- Identify the data pump group:
   EXTRACT pump_1
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_2][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the target system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS target, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify the remote trail and encryption algorithm on the target system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL remote_trail_2
   -- Specify tables and sequences to be captured:
   SEQUENCE [container.|catalog.]owner.sequence;
   TABLE [container.|catalog.]owner.table;
   ```

Target System

Configure the Manager process and primary Replicat groups for the target system.

To Configure the Manager Process

1. Configure the Manager process.
2. In the Manager parameter file, use the `PURGEOLDEXTRACTS` parameter to control the purging of files from the trail.

To Configure the Replicat Groups

1. On the target, create a Replicat checkpoint table (unless using Oracle integrated Replicat). See About Checkpoint Table for instructions.

2. On the target, use the `ADD REPLICAT` command to create a Replicat group for each remote trail that you created. For documentation purposes, these groups are called *rep_1* and *rep_2*.

   Command to add *rep_1*:

   ```
   ADD REPLICAT rep_1
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_1, BEGIN time
   ```

   Command to add *rep_2*:

   ```
   ADD REPLICAT rep_2
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_2, BEGIN time
   ```

   Use the `EXTTRAIL` argument to link the Replicat group to the trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

3. On the target, use the `EDIT PARAMS` command to create a parameter file for each Replicat group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   Parameter file for *rep_1*:

   ```
   -- Identify the Replicat group:
   REPLICAT rep_1
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_3][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

   Parameter file for *rep_1*:

   ```
   -- Identify the Replicat group:
   REPLICAT rep_2
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_3][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

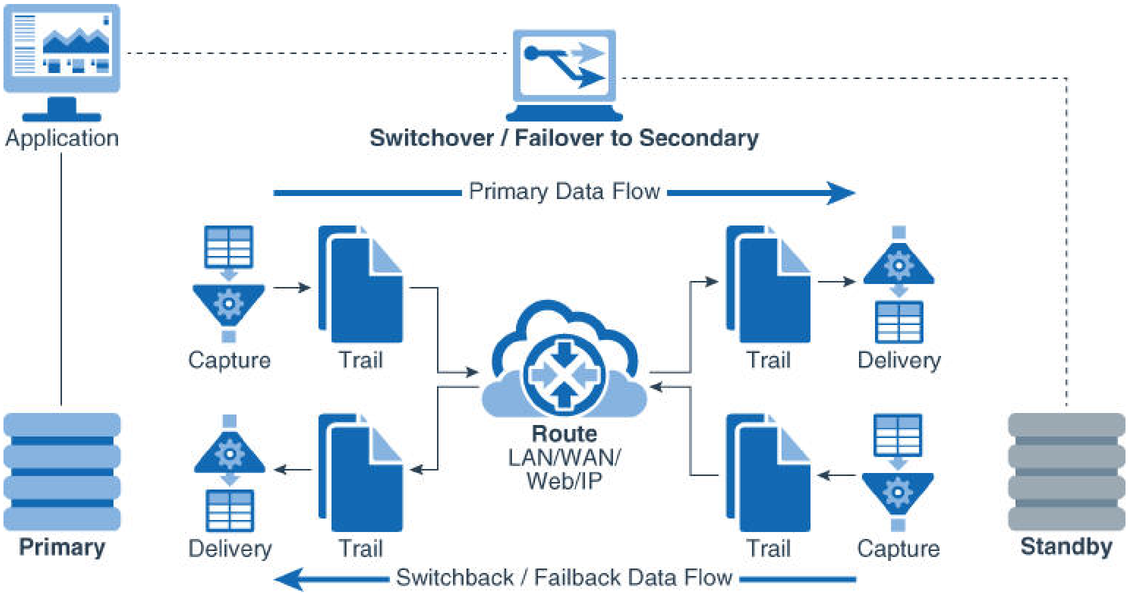
   You can use any number of `MAP` statements for any given Replicat group. All `MAP` statements for a given Replicat group must specify the same objects that are contained in the trail that is linked to the group.

Configuring Oracle GoldenGate to Maintain a Live Standby Database

This chapter describes how to configure Oracle GoldenGate to maintain a live standby database.

Overview of a Live Standby Configuration

Oracle GoldenGate supports an active-passive bi-directional configuration, where Oracle GoldenGate replicates data from an active primary database to a full replica database on a live standby system that is ready for failover during planned and unplanned outages.



In this configuration, there is an inactive Oracle GoldenGate Extract group and an inactive data pump on the live standby system. Both of those groups remain stopped until just before user applications are switched to the live standby system in a switchover or failover. When user activity moves to the standby, those groups begin capturing transactions to a local trail, where the data is stored on disk until the primary database can be used again.

In the case of a failure of the primary system, the Oracle GoldenGate Manager and Replicat processes work in conjunction with a database instantiation taken from the standby to restore parity between the two systems after the primary system is recovered. At the appropriate time, users are moved back to the primary system, and Oracle GoldenGate is configured in ready mode again, in preparation for future failovers.

Considerations for a Live Standby Configuration

These sections describe considerations for a live standby configuration.

Trusted Source

The primary database is the *trusted source*. This is the database that is the *active source* during normal operating mode, and it is the one from which the other database is derived in the initial synchronization phase and in any subsequent resynchronizations. Maintain frequent backups of the trusted source data.

Duplicate Standby

In most implementations of a live standby, the source and target databases are identical in content and structure. Data mapping, conversion, and filtering typically are not appropriate practices in this kind of configuration, but Oracle GoldenGate does support such functionality if required by your business model. To support these functions, use the options of the `TABLE` and `MAP` parameters.

DML on the Standby System

If your applications permit, you can use the live standby system for reporting and queries, but not DML. If there will be active transactional applications on the live standby system that affect objects in the Oracle GoldenGate configuration, you should configure this as an active-active configuration. See Configuring Oracle GoldenGate for Active-Active Configuration for more information.

Oracle GoldenGate Processes

During normal operating mode, leave the primary Extract and the data pump on the live standby system stopped, and leave the Replicat on the active source stopped. This prevents any DML that occurs accidentally on the standby system from being propagated to the active source. Only the Extract, data pump, and Replicat that move data from the active source to the standby system can be active.

Backup Files

Make regular backups of the Oracle GoldenGate working directories on the primary and standby systems. This backup must include all of the files that are installed at the root level of the Oracle GoldenGate installation directory and all of the sub-directories within that directory. Having a backup of the Oracle GoldenGate environment means that you will not have to recreate your process groups and parameter files.

Failover Preparedness

Make certain that the primary and live standby systems are ready for immediate user access in the event of a planned switchover or an unplanned source failure. The following components of a high-availability plan should be made easily available for use on each system:

- Scripts that grant insert, update, and delete privileges.
- (Optional) Scripts that enable triggers and cascaded delete constraints on the live standby system. (These may have been disabled during the setup procedures that were outlined in the Oracle GoldenGate installation and configuration document for your database type.)

> **Note:**
>
> Scripts to enable triggers and cascaded delete constraints on the live standby system are not required with Oracle. It's controlled by the `DEFERREFCONST` and `SUPRESSTRIGGERS` parameter settings.

- Scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.
- A failover procedure for moving users to the live standby if the source system fails.

Sequential Values that are Generated by the Database

If database-generated values, such as Oracle sequences, are used as part of a key, the range of values must be different on each system, with no chance of overlap. If the application permits, you can add a location identifier to the value to enforce uniqueness.

For Oracle databases, Oracle GoldenGate can be configured to replicate sequences in a manner that ensures uniqueness on each database. To replicate sequences, use the `SEQUENCE` and `MAP` parameters. .

Additional Information

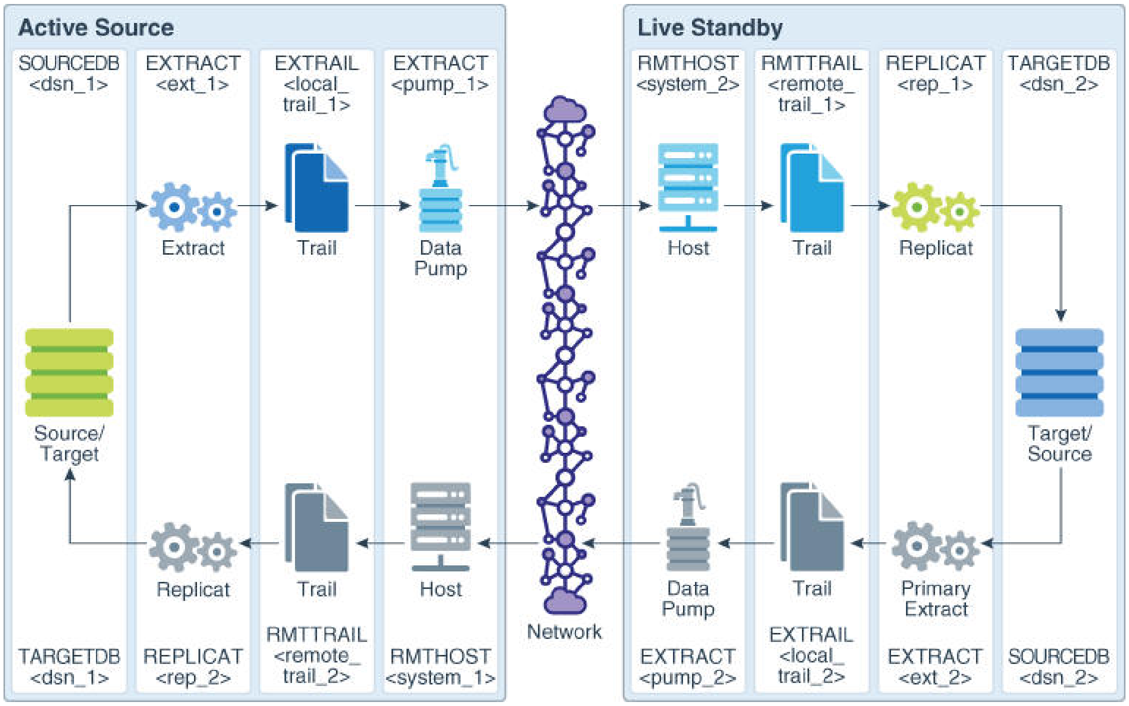
The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional system requirements, process configuration, and database setup requirements, see Install and Prepare.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see Configuring Online Change Synchronization.
- For additional tuning options for Oracle GoldenGate, see Tuning the Performance of Oracle GoldenGate.
- For complete syntax and descriptions of the Oracle GoldenGate commands and parameters, see Reference for Oracle GoldenGate for Windows and UNIX.

Creating a Live Standby Configuration

Refer to #unique_1047/unique_1047_Connect_42_I1041175 for a visual representation of the objects you will be creating.

Figure 10-7 Oracle GoldenGate configuration elements for live standby



Prerequisites on Both Systems

Perform the following prerequisites on both systems.

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). For instructions, see About Checkpoint Table.
2. Configure the Manager process.

Configuration from Active Source to Standby

These steps configure Oracle GoldenGate to capture data from the primary database and replicate it to the standby database.

To Configure the Primary Extract Group

Perform these steps on the active source.

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext_1*.

   ```
   ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.
2. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called *local_trail_1*.

   ```
   ADD EXTTRAIL local_trail_1, EXTRACT ext_1
   ```

   For `EXTRACT`, specify the *ext_1* group to write to this trail.

3. Use the `EDIT PARAMS` command to create a parameter file for the *ext_1* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT ext_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Log all scheduling columns if using integrated Replicat
LOGALLSUPCOLS
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL local_trail_1
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Data Pump

Perform these steps on the active source.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_1*.

   ```
   ADD EXTRACT pump_1, EXTTRAILSOURCE local_trail_1, BEGIN time
   ```

   For `EXTTRAILSOURCE`, specify *local_trail_1* as the data source.

2. Use the `ADD RMTTRAIL` command to specify a remote trail that will be created on the standby system.

   ```
   ADD RMTTRAIL remote_trail_1, EXTRACT pump_1
   ```

   For `EXTRACT`, specify the *pump_1* data pump to write to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

3. Use the `EDIT PARAMS` command to create a parameter file for the *pump_1* group. Include the following parameters plus any others that apply to your database environment.

```
-- Identify the data pump group:
EXTRACT pump_1
-- Specify database login information as needed for the database:
[SOURCEDB dsn_1][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the standby system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS system_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify the remote trail and encryption algorithm on the standby system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Replicat Group

Perform these steps on the live standby system.

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). See Creating a Checkpoint Table for instructions.
2. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep_1*.

   ```
   ADD REPLICAT rep_1
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_1, BEGIN time
   ```

   For `EXTTRAIL`, specify *remote_trail_1* as the trail that this Replicat reads.

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.
3. Use the `EDIT PARAMS` command to create a parameter file for the *rep_1* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Replicat group:
   REPLICAT rep_1
   -- State that source and target definitions are identical:
   ASSUMETARGETDEFS
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_2][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

Configuration from Standby to Active Source

These steps configure Oracle GoldenGate in passive mode. In this mode, the Oracle GoldenGate processes are ready, but not started, to capture data from the secondary database and replicate it to the primary database after a switchover of transaction activity to the secondary system.

> **Note:**
>
> This is a reverse image of the configuration that you just created.

To Configure the Primary Extract Group

Perform these steps on the live standby system.

1. On the source, use the `ADD EXTRACT` command to create an Extract group. For documentation purposes, this group is called *ext_2*.

   ```
   ADD EXTRACT ext_2, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time [option[, ...]]
   ```

See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other `ADD EXTRACT` options that may be required for your installation.

2. Start the `TRANLOG` Extract `ext_2`. Also see Preventing Data Looping.
3. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called *local_trail_2*.

   ```
   ADD EXTTRAIL local_trail_2, EXTRACT ext_2
   ```

   For `EXTRACT`, specify the *ext_2* group to write to this trail.
4. Use the `EDIT PARAMS` command to create a parameter file for the *ext_2* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT ext_2
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_2][, USERIDALIAS alias]
   -- Log all scheduling columns if using integrated Replicat
   LOGALLSUPCOLS
   -- Specify the local trail this Extract writes to and the encryption algorithm:
   ENCRYPTTRAIL algorithm
   EXTTRAIL local_trail_2
   -- Specify sequences to be captured:
   SEQUENCE [container.]owner.sequence;
   -- Specify tables to be captured:
   TABLE [container.]owner.*;
   ```

To Configure the Data Pump

Perform these steps on the live standby system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *pump_2*.

   ```
   ADD EXTRACT pump_2, EXTTRAILSOURCE local_trail_2, BEGIN time
   ```

   For `EXTTRAILSOURCE`, specify *local_trail_2* as the data source.
2. Use the `ADD RMTTRAIL` command to add a remote trail *remote_trail_2* that will be created on the active source system.

   ```
   ADD RMTTRAIL remote_trail_2, EXTRACT pump_2
   ```

   For `EXTRACT`, specify the *pump_2* data pump to write to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.
3. Use the `EDIT PARAMS` command to create a parameter file for the *pump_2* group. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pump_2
   -- Specify database login information as needed for the database:
   [SOURCEDB dsn_2][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the active source system
   -- and optional encryption of data over TCP/IP:
   ```

```
RMTHOSTOPTIONS system_1, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on active source system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_2
-- Specify sequences to be captured:
SEQUENCE [container.]owner.sequence;
-- Specify tables to be captured:
TABLE [container.]owner.*;
```

To Configure the Replicat Group

Perform these steps on the active source.

1. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *rep_2*.

   ```
   ADD REPLICAT rep_2
   [, INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL remote_trail_1, BEGIN time
   ```

   For `EXTTRAIL`, specify *remote_trail_2* as the trail that this Replicat reads.

   See *Parameters and Functions Reference for Oracle GoldenGate* for detailed information about these and other options that may be required for your installation.

2. Use the `EDIT PARAMS` command to create a parameter file for the *rep_2* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Replicat group:
   REPLICAT rep_2
   -- State that source and target definitions are identical:
   ASSUMETARGETDEFS
   -- Specify database login information as needed for the database:
   [TARGETDB dsn_1][, USERIDALIAS alias]
   -- Handle collisions between failback data copy and replication:
   HANDLECOLLISIONS
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery and threads if using coordinated Replicat:
   MAP [container.|catalog.]owner.table, TARGET owner.table[, DEF template]
   [, THREAD (thread_ID)][, THREADRANGE (thread_range[, column_list])]
   ;
   ```

Moving User Activity in a Planned Switchover

This procedure moves user application activity from a primary database to a live standby system in a planned, graceful manner so that system maintenance and other procedures that do not affect the databases can be performed on the primary system.

Moving User Activity to the Live Standby

To move user activity to the live standby:

1. (Optional) If you need to perform system maintenance on the secondary system, you can do so now or at the specified time later in these procedures, after moving users from the secondary system back to the primary system. In either case, be aware of the following risks if you must shut down the secondary system for any length of time:

  - The local trail on the primary system could run out of disk space as data accumulates while the standby is offline. This will cause the primary Extract to abend.
  - If the primary system fails while the standby is offline, the data changes will not be available to be applied to the live standby when it is functional again, thereby breaking the synchronized state and requiring a full re-instantiation of the live standby.

2. On the **primary** system, stop the user applications, but leave the primary Extract and the data pump on that system running so that they capture any backlogged transaction data.
3. On the **primary** system, issue the following command for the primary Extract until it returns "`At EOF, no more records to process.`" This indicates that all transactions are now captured.

   ```
   LAG EXTRACT ext_1
   ```

   > **Note:**
   >
   > Since capture continues to read `REDO`, the non-production workload continues to work. In this case, there is possibility that `At EOF` is never returned even though the production workload has already stopped8.5.1..

4. On the **primary** system, stop the primary Extract process

   ```
   STOP EXTRACT ext_1
   ```

5. On the **primary** system, issue the following command for the data pump until it returns "`At EOF, no more records to process.`" This indicates that the pump sent all of the captured data to the live standby.

   ```
   LAG EXTRACT pump_1
   ```

6. On the **live standby** system, issue the `STATUS REPLICAT` command until it returns "`At EOF (end of file).`" This confirms that Replicat applied all of the data from the trail to the database.

   ```
   STATUS REPLICAT rep_1
   ```

7. On the **live standby** system, stop Replicat.

   ```
   STOP REPLICAT rep_1
   ```

8. On the **live standby** system, do the following:
   - Run the script that grants insert, update, and delete permissions to the users of the business applications.
   - Run the script that enables triggers and cascade delete constraints.
   - Run the scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.
9. Switch user activity to the **live standby** system.
10. On the **primary system**, perform the system maintenance.

Moving User Activity Back to the Primary System

To move user activity back to the primary system:

1. On the **live standby** system, stop the user applications, but leave the primary Extract running so that it captures any backlogged transaction data.
2. On the **primary** system, start Replicat in preparation to receive changes from the live standby system.

   ```
   START REPLICAT rep_2
   ```

3. On the **live standby** system, start the data pump to begin moving the data that is stored in the local trail across TCP/IP to the primary system.

   ```
   START EXTRACT pump_2
   ```

4. On the **live standby** system, issue the following command for the primary Extract until it returns "`At EOF, no more records to process.`" This indicates that all transactions are now captured.

   ```
   LAG EXTRACT ext_2
   ```

5. On the **live standby** system, stop the primary Extract.

   ```
   STOP EXTRACT ext_2
   ```

6. On the **live standby** system, issue the following command for the data pump until it returns "`At EOF, no more records to process.`" This indicates that the pump sent all of the captured data to the primary system.

   ```
   LAG EXTRACT pump_2
   ```

7. On the **live standby** system, stop the data pump.

   ```
   STOP EXTRACT pump_2
   ```

8. On the **primary** system, issue the `STATUS REPLICAT` command until it returns "`At EOF (end of file).`" This confirms that Replicat applied all of the data from the trail to the database.

   ```
   STATUS REPLICAT rep_2
   ```

9. On the **primary** system, stop Replicat.

   ```
   STOP REPLICAT rep_2
   ```

10. On the **primary** system, do the following:
    - Run the script that grants insert, update, and delete permissions to the users of the business applications.
    - Run the script that enables triggers and cascade delete constraints.
    - Run the scripts that switch over the application server, start applications, and copy essential files that are not part of the replication environment.
11. On the **primary** system, alter the primary Extract to begin capturing data based on the current timestamp. Otherwise, Extract will spend unnecessary time looking for operations that were already captured and replicated while users were working on the standby system.

    ```
    ALTER EXTRACT ext_1, BEGIN NOW
    ```

12. On the **primary** system, start the primary Extract so that it is ready to capture transactional changes.

    ```
    START EXTRACT ext_1
    ```

13. Switch user activity to the **primary** system.

14. (Optional) If system maintenance must be done on the **live standby** system, you can do it now, before starting the data pump on the primary system. Note that captured data will be accumulating on the primary system while the standby is offline.
15. On the **primary** system, start the data pump.

    ```
    START EXTRACT pump_1
    ```

16. On the **live standby** system, start Replicat.

    ```
    START REPLICAT rep_1
    ```

Moving User Activity in an Unplanned Failover

These sections describe how to move user activity in an unplanned failover.

Moving User Activity to the Live Standby

This procedure does the following:

- Prepares the live standby for user activity.
- Ensures that all transactions from the primary system are applied to the live standby.
- Activates Oracle GoldenGate to capture transactional changes on the live standby.
- Moves users to the live standby system.

Perform these steps on the live standby system

To move users to the live standby

1. Issue the `STATUS REPLICAT` command until it returns "`At EOF (end of file)`" to confirm that Replicat applied all of the data from the trail to the database.

    ```
    STATUS REPLICAT rep_1
    ```

2. Stop the Replicat process.

    ```
    STOP REPLICAT rep_1
    ```

3. Run the script that grants insert, update, and delete permissions to the users of the business applications.
4. Run the script that enables triggers and cascade delete constraints.
5. Run the scripts that fail over the application server, start applications, and copy essential files that are not part of the replication environment.
6. Start the primary Extract process on the live standby.

    ```
    START EXTRACT ext_2
    ```

7. Move the users to the standby system and let them start working.

> **Note:**
>
> Do not start the data pump group on the standby. The user transactions must accumulate there until just before user activity is moved back to the primary system.

Moving User Activity Back to the Primary System

This procedure does the following:

- Recovers the Oracle GoldenGate environment.
- Makes a copy of the live standby data to the restored primary system.
- Propagates user transactions that occurred while the copy was being made.
- Reconciles the results of the copy with the propagated changes.
- Moves users from the standby system to the restored primary system.
- Prepares replication to maintain the live standby again.

Perform these steps after the recovery of the primary system is complete.

To Recover the Source Oracle GoldenGate Environment

1. On the **primary** system, recover the Oracle GoldenGate directory from your backups.
2. On the **primary** system, run GGSCI.
3. On the **primary** system, delete the primary Extract group.

   ```
   DELETE EXTRACT ext_1
   ```

4. On the **primary** system, delete the local trail.

   ```
   DELETE EXTTRAIL local_trail_1
   ```

5. On the **primary** system, add the primary Extract group again, using the same name so that it matches the parameter file that you restored from backup. For documentation purposes, this group is called *ext_1*. This step initializes the Extract checkpoint from its state before the failure to a clean state.

   ```
   ADD EXTRACT ext_1, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   [, THREADS n]
   ```

   - For `TRANLOG` and `INTEGRATED TRANLOG`, see *Parameters and Functions Reference for Oracle GoldenGate*. `INTEGRATED TRANLOG` enables integrated capture for an Oracle database.

6. On the **primary** system, add the local trail again, using the same name as before. For documentation purposes, this trail is called *local_trail_1*.

   ```
   ADD EXTTRAIL local_trail_1, EXTRACT ext_1
   ```

   - For `EXTRACT`, specify the *ext_1* group to write to this trail.

7. On the **primary** system, start the Manager process.

   ```
   START MANAGER
   ```

To Copy the Database from Standby to Primary System

1. On the **primary** system, run scripts to disable triggers and cascade delete constraints.
2. On the **standby** system, start making a hot copy of the database.
3. On the **standby** system, record the time at which the copy finishes.
4. On the **standby system**, stop user access to the applications. Allow all open transactions to be completed.

To Propagate Data Changes Made During the Copy

1. On the **primary** system, start Replicat.

   ```
   START REPLICAT rep_2
   ```

2. On the **live standby** system, start the data pump. This begins transmission of the accumulated user transactions from the standby to the trail on the primary system.

   ```
   START EXTRACT pump_2
   ```

3. On the **primary** system, issue the `INFO REPLICAT` command until you see that it posted all of the data changes that users generated on the standby system during the initial load. Refer to the time that you recorded previously. For example, if the copy stopped at 12:05, make sure that change replication has posted data up to that point.

   ```
   INFO REPLICAT rep_2
   ```

4. On the **primary** system, issue the following command to turn off the `HANDLECOLLISIONS` parameter and disable the initial-load error handling.

   ```
   SEND REPLICAT rep_2, NOHANDLECOLLISIONS
   ```

5. On the **primary** system, issue the `STATUS REPLICAT` command until it returns "`At EOF (end of file)`" to confirm that Replicat applied all of the data from the trail to the database.

   ```
   STATUS REPLICAT rep_2
   ```

6. On the **live standby** system, stop the data pump. This stops transmission of any user transactions from the standby to the trail on the primary system.

   ```
   STOP EXTRACT pump_2
   ```

7. On the **primary** system, stop the Replicat process.

   ```
   STOP REPLICAT rep_2
   ```

At this point in time, the primary and standby databases should be in a state of synchronization again.

(Optional) To Verify Synchronization

1. Use a compare tool, such as Oracle GoldenGate Veridata, to compare the source and standby databases for parity.
2. Use a repair tool, such as Oracle GoldenGate Veridata, to repair any out-of-sync conditions.

To Switch Users to the Primary System

1. On the **primary** system, run the script that grants insert, update, and delete permissions to the users of the business applications.
2. On the **primary** system, run the script that enables triggers and cascade delete constraints.
3. On the **primary** system, run the scripts that fail over the application server, start applications, and copy essential files that are not part of the replication environment.
4. On the **primary** system, start the primary Extract process.

   ```
   START EXTRACT ext_1
   ```

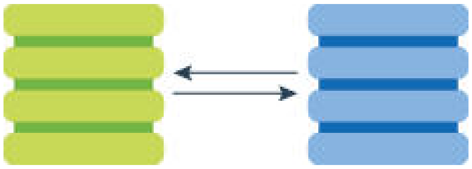
5. On the **primary** system, allow users to access the applications.

Configuring Oracle GoldenGate for Active-Active Configuration

This chapter describes how to configure Oracle GoldenGate for active-active configuration.

Overview of an Active-Active Configuration

Oracle GoldenGate supports an active-active, bidirectional configuration, where there are two systems with identical sets of data that can be changed by application users on either system. Oracle GoldenGate replicates transactional data changes from each database to the other to keep both sets of data current.



In a bidirectional configuration, there is a complete set of active Oracle GoldenGate processes on each system. Data captured by an Extract process on one system is propagated to the other system, where it is applied by a local Replicat process.

This configuration supports load sharing. It can be used for disaster tolerance if the business applications are identical on any two peers.

Oracle GoldenGate supports active-active configurations for:

- DB2 on z/OS, LUW, and IBM i
- MySQL
- Oracle
- PostgreSQL
- SQL Server

Considerations for an Active-Active Configuration

The following considerations apply in an active-active configuration. In addition, review the Oracle GoldenGate installation and configuration document for your type of database to see if there are any other limitations or requirements to support a bi-directional configuration.

Application Design

When using Active-Active replication, the time zones must be the same on both systems so that timestamp-based conflict resolution and detection can operate.

Active-active replication is not recommended for use with commercially available packaged business applications, unless the application is designed to support it. Among the obstacles that these applications present are:

- Packaged applications might contain objects and data types that are not supported by Oracle GoldenGate.

- They might perform automatic DML operations that you cannot control, but which will be replicated by Oracle GoldenGate and cause conflicts when applied by Replicat.
- You probably cannot control the data structures to make modifications that are required for active-active replication.

Keys

For accurate detection of conflicts, all records must have a unique, not-null identifier. If possible, create a primary key. If that is not possible, use a unique key or create a substitute key with a `KEYCOLS` option of the `MAP` and `TABLE` parameters. In the absence of a unique identifier, Oracle GoldenGate uses all of the columns that are valid in a `WHERE` clause, but this will degrade performance if the table contains numerous columns.

To maintain data integrity and prevent errors, the following must be true of the key that you use for any given table:

- contain the same columns in all of the databases where that table resides.
- contain the same values in each set of corresponding rows across the databases.

Database-Generated Values

Do not replicate database-generated sequential values, such as Oracle sequences, in a bi-directional configuration. The range of values must be different on each system, with no chance of overlap. For example, in a two-database environment, you can have one server generate even values, and the other odd. For an *n*-server environment, start each key at a different value and increment the values by the number of servers in the environment. This method may not be available to all types of applications or databases. If the application permits, you can add a location identifier to the value to enforce uniqueness.

Database Configuration

One of the databases must be designated as the *trusted source*. This is the primary database and its host system from which the other database is derived in the initial synchronization phase and in any subsequent resynchronizations that become necessary. Maintain frequent backups of the trusted source data.

Preventing Data Looping

In a bidirectional configuration, SQL changes that are replicated from one system to another must be prevented from being replicated back to the first system. Otherwise, it moves back and forth in an endless loop, as in this example:

1. A user application updates a row on system A.
2. Extract extracts the row on system A and sends it to system B.
3. Replicat updates the row on system B.
4. Extract extracts the row on system B and sends it back to system A.
5. The row is applied on system A (for the second time).
6. This loop continues endlessly.

To prevent data loopback, you may need to provide instructions that:

- prevent the capture of SQL operations that are generated by Replicat, but enable the capture of SQL operations that are generated by business applications if they contain objects that are specified in the Extract parameter file.
- identify local Replicat transactions, in order for the Extract process to ignore them.

Identifying Replicat Transactions

To configure Extract to identify Replicat transactions, follow the instructions for the database from which Extract will capture data.

DB2 z/OS, DB2 LUW, and DB2 for i

Identify the Replicat user name by using the following parameter statement in the Extract parameter file.

```
TRANLOGOPTIONS EXCLUDEUSER user
```

This parameter statement marks all DDL and DML transactions that are generated by this user as Replicat transactions. The user name is included in the transaction record that is read by Extract.

MySQL

Identify the name of the Replicat checkpoint table by using the following parameter statement in the Extract parameter file.

```
TRANLOGOPTIONS FILTERTABLE table_name
```

Replicat writes a checkpoint to the checkpoint table at the end of each of its transactions as part of its checkpoint procedure. (This is the table that is created with the `ADD CHECKPOINTTABLE` command.) Because every Replicat transaction includes a write to this table, it can be used to identify Replicat transactions in a bidirectional configuration. `FILTERTABLE` identifies the name of the checkpoint table, so that Extract ignores transactions that contain any operations on it.

PostgreSQL and SQL Server

Identify the name of the Replicat checkpoint table by using the following parameter statement in the Extract parameter file and ensure that the Replicat checkpoint table has been enabled for supplemental logging with the `ADD TRANDATA` command.

```
TRANLOGOPTIONS FILTERTABLE table_name
```

Replicat writes a checkpoint to the checkpoint table at the end of each of its transactions as part of its checkpoint procedure. (This is the table that is created with the `ADD CHECKPOINTTABLE` command). Because every Replicat transaction includes a write to this table, it can be used to identify Replicat transactions in a bi-directional configuration. `FILTERTABLE` identifies the name of the checkpoint table, so that Extract ignores transactions that contain any operations on it.

Oracle

There are multiple ways to identify Replicat transaction in an Oracle environment. When Replicat is in classic or integrated mode, you use the following parameters:

- Replicats set a tag of 00 by default. Use `DBOPTIONS` with the `SETTAG` option in the Replicat parameter file to change the tag that Replicat sets. Replicat tags the transactions being applied with the specified value, which identifies those transactions in the redo stream. Valid values are a single TAG value consisting of hexadecimal digits.
- Use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG` option in the Extract parameter file. The logmining server associated with that Extract excludes redo that is tagged with the `SETTAG` value.

  The following shows how `SETTAG` can be set in the Replicat parameter file:

  ```
  DBOPTIONS SETTAG 0935
  ```

  The following shows how `EXCLUDETAG` can be set in the Extract parameter file:

  ```
  TRANLOGOPTIONS EXCLUDETAG 0935
  ```

  If you are excluding multiple tags, each must have a separate `TRANLOGOPTIONS EXCLUDETAG` statement specified.

You can also use the transaction name or userid of the Replicat user to identify Replicat transactions. You can choose which of these to ignore when you configure Extract. See Preventing the Capture of Replicat Transactions (Oracle).

For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Preventing the Capture of Replicat Operations

Depending on which database you are using, you may or may not need to provide explicit instructions to prevent the capture of Replicat operations.

Preventing the Capture of Replicat Transactions (Oracle)

When Extract is integrated capture mode, use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG` *`tag`* option. This parameter directs the Extract process to ignore transactions that are tagged with the specified redo tag. See Identifying Replicat Transactions to set the tag value. This is the recommended approach for Oracle.

Preventing Capture of Replicat Transactions (Other Databases)

To prevent the capture of SQL that is applied by Replicat to other database types (including Oracle, if Extract operates in classic capture mode), use the following parameters:

- `GETAPPLOPS | IGNOREAPPLOPS`: Controls whether or not data operations (DML) produced by business applications *except Replicat* are included in the content that Extract writes to a specific trail or file.
- `GETREPLICATES | IGNOREREPLICATES`: Controls whether or not DML operations produced by *Replicat* are included in the content that Extract writes to a specific trail or file.

Replicating DDL in a Bidirectional Configuration

There are additional consideration when replicating DDL bidirectionally, supported in Oracle and MySQL databases. For information on DDL support for Oracle, see Configuring DDL Support . For information on DDL support in MySQL, see Using DDL Replication.

Managing Conflicts

Uniform conflict-resolution procedures must be in place on all systems in an active-active configuration. Conflicts should be identified immediately and handled with as much automation as possible; however, different business applications will present their own unique set of requirements in this area.

Because Oracle GoldenGate is an asynchronous solution, conflicts can occur when modifications are made to identical sets of data on separate systems at (or almost at) the same time. Conflicts occur when the timing of simultaneous changes results in one of these out-of-sync conditions:

- A **uniqueness conflict** occurs when Replicat applies an insert or update operation that violates a uniqueness integrity constraint, such as a `PRIMARY KEY` or `UNIQUE` constraint. An example of this conflict type is when two transactions originate from two different databases, and each one inserts a row into a table with the same primary key value.
- An **update conflict** occurs when Replicat applies an update that conflicts with another update to the same row. Update conflicts happen when two transactions that originate from different databases update the same row at nearly the same time. Replicat detects an update conflict when there is a difference between the old values (the before values) that are stored in the trail record and the current values of the same row in the target database.
- A **delete conflict** occurs when two transactions originate at different databases, and one deletes a row while the other updates or deletes the same row. In this case, the row does not exist to be either updated or deleted. Replicat cannot find the row because the primary key does not exist.

For example, UserA on DatabaseA updates a row, and UserB on DatabaseB updates the same row. If UserB's transaction occurs before UserA's transaction is synchronized to DatabaseB, there will be a conflict on the replicated transaction.

A more complicated example involves three databases and illustrates a more complex ordering conflict. Assume three databases A, B, and C. Suppose a user inserts a row at database A, which is then replicated to database B. Another user then modifies the row at database B, and the row modification is replicated to database C. If the row modification from B arrives at database C before the row insert from database A, C will detect a conflict.

Where possible, try to minimize or eliminate any chance of conflict. Some ways to do so are:

- Configure the applications to restrict which columns can be modified in each database. For example, you could limit access based on geographical area, such as by allowing different sales regions to modify only the records of their own customers. As another example, you could allow a customer service application on one database to modify only the `NAME` and `ADDRESS` columns of a customer table, while allowing a financial application on another database to modify only the `BALANCE` column. In each of those cases, there cannot be a conflict caused by concurrent updates to the same record.
- Keep synchronization latency low. If UserA on DatabaseA and UserB on DatabaseB both update the same rows at about the same time, and UserA's transaction gets replicated to the target row before UserB's transaction is completed, conflict is avoided. See Tuning the Performance of Oracle GoldenGate for suggestions on improving the performance of the Oracle GoldenGate processes.

To avoid conflicts, replication latency must be kept as low as possible. When conflicts are unavoidable, they must be identified immediately and resolved with as much automation as possible, either through the Oracle GoldenGate Conflict Detection and Resolution (CDR) feature, or through methods developed on your own. Custom methods can be integrated into

Oracle GoldenGate processing through the `SQLEXEC` and user exit functionality. See Manual Conflict Detection and Resolution for more information about using Oracle GoldenGate to handle conflicts.

For Oracle database, the automatic Conflict Detection Resolution (CDR) feature exists. To know more, see Automatic Conflict Detection and Resolution.

Additional Information

The following documentation provides additional information of relevance to configuring Oracle GoldenGate.

- For additional database configuration requirements, see the *Configure Oracle GoldenGate for Oracle Database* and *Configure Oracle GoldenGate for Non-Oracle Databases* guides.
- For a bidirectional replication, see Configure Bi-Directional Replication.
- For detailed instructions on configuring Oracle GoldenGate change capture and delivery groups, see Configuring Online Change Synchronization.
- For additional tuning options for Oracle GoldenGate, see Tuning the Performance of Oracle GoldenGate.

Creating an Active-Active Configuration

See the Quickstart Bidirectional Replication for steps to configure an active-active bidirectional replication.

Prerequisites on Both Systems

Perform the following prerequisites on both systems.

1. Create a Replicat checkpoint table (unless using Oracle integrated Replicat). For instructions, see About Checkpoint Table.
2. Configure the Manager process.

Configuration from Primary System to Secondary System

These steps add the processes necessary to send data from the primary system to the secondary database.

To Configure the Primary Extract Group

Perform these steps on the primary system.

1. Use the `ADD EXTRACT` command to create a primary Extract group. For documentation purposes, this group is called *`ext_1`*.

   ```
   ADD EXTRACT exte, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   ```

2. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called *`trail_east`*.

   ```
   ADD EXTTRAIL trail_east, EXTRACT exte
   ```

   For `EXTRACT`, specify the *`ext_1`* group to write to this trail

3. Use the `EDIT PARAMS` command to create a parameter file for the *`ext_1`* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

```
-- Identify the Extract group:
EXTRACT exte
-- Specify database login information as needed for the database:
[SOURCEDB dsne][, USERIDALIAS alias]
-- Specify the local trail that this Extract writes to
-- and the encryption algorithm:
ENCRYPTTRAIL algorithm
EXTTRAIL trail_east
-- Exclude Replicat transactions. Uncomment ONE of the following:
-- DB2 z/OS, DB2 LUW, DB2 IBM i, and Oracle:
-- TRANLOGOPTIONS EXCLUDEUSER Replicat_user
-- Oracle integrated capture:
-- EXCLUDETAG tag
-- SQL Server:
-- TRANLOGOPTIONS FILTERTABLE schema.checkpointtable"
-- -- Teradata:
-- SQLEXEC 'SET SESSION OVERRIDE REPLICATION ON;'
-- SQLEXEC 'COMMIT;'
-- Log all scheduling columns for CDR and if using integrated Replicat
LOGALLSUPCOLS
-- Specify tables to be captured and (optional) columns to fetch:
TABLE [container.|catalog.]owner.* [, FETCHCOLS cols | FETCHCOLSEXCEPT
cols];
```

To Configure the Data Pump

Perform these steps on the primary system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *`pumpe`*.

   ```
   ADD EXTRACT pumpe, EXTTRAILSOURCE ea, BEGIN time
   ```

   For `EXTTRAILSOURCE`, specify *`ea`* as the data source.

2. Use the `ADD RMTTRAIL` command to add a remote trail that will be created on the secondary system. For documentation purposes, this trail is called *`er`*.

   ```
   ADD RMTTRAIL er, EXTRACT pumpr
   ```

   For `EXTRACT`, specify the *`pump_1`* data pump to write to this trail.

   See *Parameters and Functions Reference for Oracle GoldenGate* for additional `ADD RMTTRAIL` options.

   Use the `EDIT PARAMS` command to create a parameter file for the *`pumpr`*group. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pumpe
   ```

```
-- Specify database login information as needed for the database:
[SOURCEDB dsne][, USERIDALIAS alias]
-- Decrypt the data only if the data pump must process it.
-- DECRYPTTRAIL
-- Specify the name or IP address of the secondary system
-- and optional encryption of data over TCP/IP:
RMTHOSTOPTIONS system_2, MGRPORT port_number, ENCRYPT encryption_options
-- Specify remote trail and encryption algorithm on secondary system:
ENCRYPTTRAIL algorithm
RMTTRAIL remote_trail_1
-- Specify tables to be captured:
TABLE [container.|catalog.]owner.*;
```

To Configure the Replicat Group

Perform these steps on the secondary system.

1. Create the Replicat checkpoint table after using the `DBLOGIN` command to connect to the database. See `ADD CHECKPOINTTABLE` in *Command Line Interface Reference for Oracle GoldenGate*.
2. Run the command:

   ```
   ADD CHECKPOINTTABLE schema.checkpointtable
   ```

3. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called `repe`.

   ```
   ADD REPLICAT repe
   [, PARALLEL | INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL er, CHECKPOINTTABLE schema.checkpointtable
   ```

   For `EXTTRAIL`, specify `remote_trail_1` as the trail that this Replicat reads.

4. Use the `EDIT PARAMS` command to create a parameter file for the `rep_1` group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Replicat group:
   REPLICAT repe
   -- Specify database login information as needed for the database:
   [TARGETDB dsnw][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Set redo tag for Oracle only replicat via settag
   -- Default is 00.
   SETTAG tag_value
   -- Valid for Oracle only. Specify tables for delivery, threads if
   coordinated Replicat
   -- and conflict-resolution:
   MAP [container.|catalog.]owner.*, TARGET owner.*, COMPARECOLS (ON
   operation {ALL | KEY | KEYINCLUDING (col_list) | ALLEXCLUDING
   (col_list)}), RESOLVECONFLICT (conflict type (resolution_name,
   resolution_type COLS (col[,...]))
   [, THREAD (thread_ID)]
   [, THREADRANGE (thread_range[, column_list])]
   ```

```
;
-- Specify mapping of exceptions to exceptions table:
MAP [container.|catalog.]owner.*, TARGET owner.exceptions, EXCEPTIONSONLY;
```

Configuration from Secondary System to Primary System

These steps add the processes necessary to send data from the secondary system to the primary database.

To Configure the Primary Extract Group

Perform these steps on the secondary system.

> **Note:**
>
> This is a reverse image of the configuration that you just created.

1. Use the `ADD EXTRACT` command to create a primary Extract group. For documentation purposes, this group is called *`extn`*.

   ```
   ADD EXTRACT extn, {TRANLOG | INTEGRATED TRANLOG}, BEGIN time
   ```

2. Use the `ADD EXTTRAIL` command to add a local trail. For documentation purposes, this trail is called *`en`*.

   ```
   ADD EXTTRAIL en, EXTRACT extn
   ```

   For Extract, specify the *`extn`* group to write to this trail.

3. Use the `EDIT PARAMS` command to create a parameter file for the *`extn`* group. Include the following parameters plus any others that apply to your database environment. For possible additional required parameters, see the Oracle GoldenGate installation and setup guide for your database.

   ```
   -- Identify the Extract group:
   EXTRACT extn
   -- Specify database login information as needed for the database:
   [SOURCEDB dsnn][, USERIDALIAS alias]
   -- Specify the local trail that this Extract writes to
   -- and the encryption algorithm:
   ENCRYPTTRAIL algorithm
   EXTTRAIL en
   -- Exclude Replicat transactions. Uncomment ONE of the following:
   -- Db2 z/OS, Db2 LUW, Db2 IBM i, and Oracle:
   -- TRANLOGOPTIONS EXCLUDEUSER Replicat_user
   -- Oracle integrated capture:
   -- EXCLUDETAG tag
   -- SQL Server:
   -- TRANLOGOPTIONS EXCLUDETRANS FILTERTABLE schema.checkpointtable
   -- Oracle:
   -- TRACETABLE trace_table_name
   -- Log all scheduling columns for CDR and if using integrated Replicat
   ```

```
LOGALLSUPCOLS
-- Specify tables to be captured and (optional) columns to fetch:
TABLE [container.|catalog.]owner.* [, FETCHCOLS cols | FETCHCOLSEXCEPT
cols];
```

> **Note:**
>
> To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node. For more information on identifying these tables and configuring DBFS for propagation by Oracle GoldenGate, see Applying the Required Patch in *Using Oracle GoldenGate for Oracle Database*.

To Configure the Data Pump

Perform these steps on the secondary system.

1. Use the `ADD EXTRACT` command to create a data pump group. For documentation purposes, this group is called *`pumpn`*.

   ```
   ADD EXTRACT pumpn, EXTTRAILSOURCE en, BEGIN time
   ```

   For `EXTTRAILSOURCE`, specify *`en`* as the data source.
2. Use the `ADD RMTTRAIL` command to add a remote trail that will be created on the primary system. For documentation purposes, this trail is called *`rt`*.

   ```
   ADD RMTTRAIL rt, EXTRACT pumpn
   ```

   For `EXTRACT`, specify the *`pumpn`* data pump to write to this trail.
3. Use the `EDIT PARAMS` command to create a parameter file for the *`pump_2`* group. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the data pump group:
   EXTRACT pumpn
   -- Specify database login information as needed for the database:
   [SOURCEDB dsnn][, USERIDALIAS alias]
   -- Decrypt the data only if the data pump must process it.
   -- DECRYPTTRAIL
   -- Specify the name or IP address of the primary system
   -- and optional encryption of data over TCP/IP:
   RMTHOSTOPTIONS system_1, MGRPORT port_number, ENCRYPT encryption_options
   -- Specify the remote trail and encryption algorithm on the primary system:
   ENCRYPTTRAIL algorithm
   RMTTRAIL rt
   -- Specify tables to be captured:
   TABLE [container.|catalog.]owner.*;
   ```

> **Note:**
>
> To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node. For more information on identifying these tables and configuring DBFS for propagation by Oracle GoldenGate, see Configuring the DBFS File System in *Using Oracle GoldenGate for Oracle Database*.

To Configure the Replicat Group

Perform these steps on the primary system.

1. Create the Replicat checkpoint table after using the `DBLOGIN` command to connect to the database. See `ADD CHECKPOINTTABLE` in *Command Line Interface Reference for Oracle GoldenGate*.
2. Run the command:

   ```
   ADD CHECKPOINTTABLE schema.checkpointtable
   ```

3. Use the `ADD REPLICAT` command to create a Replicat group. For documentation purposes, this group is called *`reps`*.

   ```
   ADD REPLICAT reps
   [, PARALLEL | INTEGRATED | COORDINATED [MAXTHREADS number]]
   , EXTTRAIL rt, CHECKPOINTTABLE schema.checkpointtable
   ```

   For `EXTTRAIL`, specify *`remote_trail_1`* as the trail that this Replicat reads.

4. Use the `EDIT PARAMS` command to create a parameter file for the *`rep_2`* group. Include the following parameters plus any others that apply to your database environment.

   ```
   -- Identify the Replicat group:
   REPLICAT reps
   -- Specify database login information as needed for the database:
   [TARGETDB dsns][, USERIDALIAS alias]
   -- Specify error handling rules:
   REPERROR (error, response)
   -- Specify tables for delivery, threads if coordinated Replicat
   -- and conflict-resolution:
   MAP [container.|catalog.]owner.*, TARGET owner.*, COMPARECOLS (ON
   operation {ALL | KEY | KEYINCLUDING (col_list) | ALLEXCLUDING
   (col_list)}), RESOLVECONFLICT (conflict type (resolution_name,
   resolution_type COLS (col[,...]))
   [, THREAD (thread_ID)]
   [, THREADRANGE (thread_range[, column_list])]
   ;
   -- Specify mapping of exceptions to exceptions table:
   MAP [container.|catalog.]owner.*, TARGET owner.exceptions, EXCEPTIONSONLY;
   ```

> **Note:**
>
> To replicate Oracle DBFS data, specify the internally generated local read-write DBFS tables in the `TABLE` statement on each node.

Manual Conflict Detection and Resolution

This chapter contains instructions for manually configuring Conflict Detection and Resolution (CDR) using specific parameters. Conflict detection and resolution is required in active-active configurations, where Oracle GoldenGate must maintain data synchronization among multiple databases that contain the same data sets.

Overview of the Oracle GoldenGate CDR Feature

Oracle GoldenGate Conflict Detection and Resolution (CDR) provides basic conflict resolution routines that:

- Resolve a uniqueness conflict for an `INSERT`.
- Resolve a "no data found" conflict for an `UPDATE` when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a "no data found" conflict for an `UPDATE` when the row does not exist.
- Resolve a "no data found" conflict for a `DELETE` when the row exists, but the before image of one or more columns is different from the current value in the database.
- Resolve a "no data found" conflict for a `DELETE` when the row does not exist.

To use conflict detection and resolution (CDR), the target database must reside on a Windows, Linux, or UNIX system. It is not supported for databases on the NonStop platform.

CDR supports scalar data types such as:

- `NUMERIC`
- `DATE`
- `TIMESTAMP`
- `CHAR/NCHAR`
- `VARCHAR/ NVARCHAR`

This means that these column types can be used with the `COMPARECOLS` parameter and as the resolution column in the `USEMIN` and `USEMAX` options of the `RESOLVECONFLICT` parameter. Only `NUMERIC` columns can be used for the `USEDELTA` option of `RESOLVECONFLICT`. Do not use CDR for columns that contain LOBs, abstract data types (ADT), or user-defined types (UDT).

Conflict resolution is not performed when Replicat operates in `BATCHSQL` mode. If a conflict occurs in `BATCHSQL` mode, Replicat reverts to `GROUPTRANSOPS` mode, and then to single-transaction mode. Conflict detection occurs in all three modes. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Configuring the Oracle GoldenGate Parameter Files for Error Handling

CDR should be used in conjunction with error handling to capture errors that were resolved and errors that CDR could not resolve.

1. Conflict resolution is performed before these other error-handling parameters: `HANDLECOLLSIONS`, `INSERTMISSINGUPDATES`, and `REPERROR`. Use the `REPERROR` parameter to assign rules for handling errors that cannot be resolved by CDR, or for errors that you do not want to handle through CDR. It might be appropriate to have `REPERROR` handle some errors, and CDR handle others; however, if `REPERROR` and CDR are configured to handle the same conflict, CDR takes precedence. The `INSERTMISSINGUPDATES` and `HANDLECOLLISIONS` parameters also can be used to handle some errors not handled by CDR. See the Reference for Oracle GoldenGate for details about these parameters.
2. (Optional) Create an exceptions table. When an exceptions table is used with an exceptions `MAP` statement, Replicat sends every operation that generates a conflict (resolved or not) to the exceptions `MAP` statement to be mapped to the exceptions table. Omit a primary key on this table if Replicat is to process `UPDATE` and `DELETE` conflicts; otherwise there can be integrity constraint errors.

   At minimum, an exceptions table should contain the same columns as the target table. These rows will contain each row image that Replicat applied to the target (or tried to apply).

   In addition, you can define additional columns to capture other information that helps put the data in transactional context. Oracle GoldenGate provides tools to capture this information through the exceptions `MAP` statement. Such columns can be, but are not limited to, the following:

   - The before image of the trail record. This is a duplicate set of the target columns with names such as *`col1`*_before, *`col2`*_before, and so forth.
   - The current values of the target columns. This also is a duplicate set of the target columns with names such as *`col1`*_current, *`col2`*_current, and so forth.
   - The name of the target table
   - The timestamp of the conflict
   - The operation type
   - The database error number
   - (Optional) The database error message
   - Whether the conflict was resolved or not
3. Create an exceptions `MAP` statement to map the exceptions data to the exceptions table. An exceptions `MAP` statement contains:
   - (Required) The `INSERTALLRECORDS` option. This parameter converts all mapped operations to `INSERT`s so that all column values are mapped to the exceptions table.
   - (Required) The `EXCEPTIONSONLY` option. This parameter causes Replicat to map operations that generate an error, but not those that were successful.
   - (Optional) A `COLMAP` clause. If the names and definitions of the columns in the exceptions table are identical to those of the source table, and the exceptions table only contains those columns, no `COLMAP` is needed. However, if any names or definitions differ, or if there are extra columns in the exceptions table that you want to populate with additional data, use a `COLMAP` clause to map all columns.

Tools for Mapping Extra Data to the Exceptions Table

The following are some tools that you can use in the `COLMAP` clause to populate extra columns:

- If the names and definitions of the source columns are identical to those of the target columns in the exceptions table, you can use the `USEDEFAULTS` keyword instead of explicitly mapping names. Otherwise, you must map those columns in the `COLMAP` clause, for example:

```
COLMAP (exceptions_col1 = col1, [...])
```

- To map the before image of the source row to columns in the exceptions table, use the `@BEFORE` conversion function, which captures the before image of a column from the trail record. This example shows the `@BEFORE` usage.

```
COLMAP (USEDEFAULTS, exceptions_col1 = @BEFORE (source_col1), &
exceptions_col2 = @BEFORE (source_col2), [...])
```

- To map the current image of the target row to columns in the exceptions table, use a `SQLEXEC` query to capture the image, and then map the results of the query to the columns in the exceptions table by using the '*`queryID.column`*' syntax in the `COLMAP` clause, as in the following example:

```
COLMAP (USEDEFAULTS, name_current = queryID.name, phone_current = queryID.phone,
[...])
```

- To map timestamps, database errors, and other environmental information, use the appropriate Oracle GoldenGate column-conversion functions. For example, the following maps the current timestamp at time of execution.

```
res_date = @DATENOW ()
```

See Sample Exceptions Mapping with Additional Columns in the Exceptions Table , for how to combine these features in a `COLMAP` clause in the exceptions `MAP` statement to populate a detailed exceptions table.

See Reference for Oracle GoldenGate for Windows and UNIX for the usage and syntax of the parameters and column-conversion functions shown in these examples.

Sample Exceptions Mapping with Source and Target Columns Only

The following is a sample parameter file that shows error handling and simple exceptions mapping for the source and target tables that are used in the CDR examples that begin. This example maps source and target columns, but no extra columns. For the following reasons, a `COLMAP` clause is not needed in the exceptions `MAP` statement in this example:

- The source and target exceptions columns are identical in name and definition.
- There are no other columns in the exceptions table.

> **Note:**
>
> This example intentionally leaves out other parameters that are required in a Replicat parameter file, such as process name and login credentials, as well as any optional parameters that may be required for a given database type. When using line breaks to split a parameter statement into multiple lines, use an ampersand (&) at the end of each line.

```
    -- REPERROR error handling: DEFAULT represents all error types. DISCARD
    -- writes operations that could not be processed to a discard file.
REPERROR (DEFAULT, DISCARD)
    -- Specifies a discard file.
DISCARDFILE /users/ogg/discards/discards.dsc, PURGE
    -- The regular MAP statement with the CDR parameters
MAP fin.src, TARGET fin.tgt, &
```

```
COMPARECOLS (ON UPDATE ALL, ON DELETE ALL), &
RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)), &
);
    -- Starts the exceptions MAP statement by mapping the source table to the
    -- exceptions table.
MAP fin.src, TARGET fin.exception, &
    -- directs Replicat only to map operations that caused the error specified
    -- in REPERROR.
EXCEPTIONSONLY, &
    -- directs Replicat to convert all the exceptions to inserts into the
    -- exceptions table. This is why there cannot be a primary key constraint
    -- on the exceptions table.
INSERTALLRECORDS
;
```

Sample Exceptions Mapping with Additional Columns in the Exceptions Table

The following is a sample parameter file that shows error handling and complex exceptions mapping for the source and target tables that are used in the CDR examples that begin. In this example, the exceptions table has the same rows as the source table, but it also has additional columns to capture context data.

Note:

This example intentionally leaves out other parameters that are required in a Replicat parameter file, such as process name and login credentials, as well as any optional parameters that may be required for a given database type. When using line breaks to split a parameter statement into multiple lines, use an ampersand (&) at the end of each line.

```
    -- REPERROR error handling: DEFAULT represents all error types. DISCARD
    -- writes operations that could not be processed to a discard file.
REPERROR (DEFAULT, DISCARD)
    -- Specifies the discard file.
DISCARDFILE /users/ogg/discards/discards.dsc, PURGE
    -- The regular MAP statement with the CDR parameters
MAP fin.src, TARGET fin.tgt, &
COMPARECOLS (ON UPDATE ALL, ON DELETE ALL), &
RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)), &
RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)), &
RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD))
);
    -- Starts the exceptions MAP statement by mapping the source table to the --
exceptions table.
MAP fin.src, TARGET fin.exception, &
    -- directs Replicat only to map operations that caused the error specified
    -- in REPERROR.
EXCEPTIONSONLY, &
    -- directs Replicat to convert all the exceptions to inserts into the
    -- exceptions table. This is why there cannot be a primary key constraint
    -- on the exceptions table.
INSERTALLRECORDS &
```

```
    -- SQLEXEC query to select the values from the target record before the
    -- Replicat statement is applied. These are mapped to the *_target
    -- columns later.
SQLEXEC (id qry, query 'select name, phone, address, salary, balance, & comment,
last_mod_time from fin.tgt where name = :p1', PARAMS(p1 = name )), &
    -- Start of the column mapping, specifies use default column definitions.
COLMAP ( &
    -- USEDEFAULTS maps the source columns to the target exceptions columns
    -- that receive the after image that Replicat applied or tried to apply.
    -- In this case, USEDEFAULTS can be used because the names and definitions
    -- of the source and target exceptions columns are identical; otherwise
    -- the columns must be mapped explicitly in the COLMAP clause.
USEDEFAULTS, &
    -- captures the timestamp when the resolution was performed.
res_date = @DATENOW (), &
    -- captures and maps the DML operation type.
optype = @GETENV ('LASTERR', 'OPTYPE'), &
    -- captures and maps the database error number that was returned.
dberrnum = @GETENV ('LASTERR', 'DBERRNUM'), &
    -- captures and maps the database error that was returned.
dberrmsge = @GETENV ('LASTERR', 'DBERRMSG'), &
    -- captures and maps the name of the target table
tabname = @GETENV ('GGHEADER', 'TABLENAME'), &
    -- If the names and definitions of the source columns and the target
    -- exceptions columns were not identical, the columns would need to
    -- be mapped in the COLMAP clause instead of using USEDEFAULTS, as
    -- follows:
       -- name_after = name, &
       -- phone_after = phone, &
       -- address_after = address, &
       -- salary_after = salary, &
       -- balance_after = balance, &
       -- comment_after = comment, &
       -- last_mod_time_after = last_mod_time &
    -- maps the before image of each column from the trail to a column in the
    -- exceptions table.
name_before = @BEFORE (name), &
phone_before = @BEFORE (phone), &
address_before = @BEFORE (address), &
salary_before = @BEFORE (salary), &
balance_before = @BEFORE (balance), &
comment_before = @BEFORE (comment), &
last_mod_time_before = @BEFORE (last_mod_time), &
    -- maps the results of the SQLEXEC query to rows in the exceptions table
    -- to show the current image of the row in the target.
name_current = qry.name, &
phone_current = qry.phone, &
address_current = qry.address, &
salary_current = qry.salary, &
balance_current = qry.balance, &
comment_current = qry.comment, &
last_mod_time_current = qry.last_mod_time)
;
```

For more information about creating an exceptions table and using exceptions mapping, see Handling Replicat Errors during DML Operations.

Once you are confident that your routines work as expected in all situations, you can reduce the amount of data that is logged to the exceptions table to reduce the overhead of the resolution routines.

Configuring the Oracle GoldenGate Parameter Files for Conflict Resolution

The following parameters are required to support conflict detection and resolution.

1. Use the `COMPARECOLS` option of the `MAP` parameter in the Replicat parameter file to specify columns that are to be used with before values in the Replicat `WHERE` clause. The before values are compared with the current values in the target database to detect update and delete conflicts. (By default, Replicat only uses the primary key in the `WHERE` clause; this may not be enough for conflict detection).
2. Use the `RESOLVECONFLICT` option of the `MAP` parameter to specify conflict resolution routines for different operations and conflict types. You can use `RESOLVECONFLICT` multiple times in a `MAP` statement to specify different resolutions for different conflict types. However, you cannot use `RESOLVECONFLICT` multiple times for the same type of conflict. Use identical conflict-resolution procedures on all databases, so that the same conflict produces the same end result. One conflict-resolution method might not work for every conflict that could occur. You might need to create several routines that can be called in a logical order of priority so that the risk of failure is minimized.

> **Note:**
>
> Additional consideration should be given when a table has a primary key and additional unique indexes or unique keys. The automated routines provided with the `COMPARECOLS` and `RESOLVECONFLICT` parameters require a consistent way to uniquely identify each row. Failure to consistently identify a row will result in an error during conflict resolution. In these situations the additional unique keys should be disabled or you can use the `SQLEXEC` feature to handle the error thrown and resolve the conflict.

For detailed information about these parameters, see *Parameters and Functions Reference for Oracle GoldenGate*. See the examples starting on CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD, for more information on these parameters.

Making the Required Column Values Available to Extract

To use CDR, the following column values must be logged so that Extract can write them to the trail.

- The full before image of each record. Some databases do not provide a before image in the log record, and must be configured to do so with supplemental logging. For most supported databases, you can use the `ADD TRANDATA` command for this purpose.
- Use the `LOGALLSUPCOLS` parameter to ensure that the full before and after images of the scheduling columns are written to the trail. Scheduling columns are primary key, unique index, and foreign key columns. `LOGALLSUPCOLS` causes Extract to include in the trail record the before image for `UPDATE` operations and the before image of all supplementally logged columns for both `UPDATE` and `DELETE` operations.

For detailed information about these parameters and commands, see the *Parameters and Functions Reference for Oracle GoldenGate*. See the examples starting on CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD for more information on how these parameters work with CDR.

Configuring Oracle GoldenGate CDR

Here are the steps to configure the source database, target database, and Oracle GoldenGate for conflict detection and resolution.

Viewing CDR Statistics

The CDR feature provides the following methods for viewing the results of conflict resolution.

Report File

Replicat writes CDR statistics to the report file:

```
Total CDR conflicts                    7
    CDR resolutions succeeded          6
    CDR resolutions failed             1
    CDR INSERTROWEXISTS conflicts      1
    CDR UPDATEROWEXISTS conflicts      4
    CDR UPDATEROWMISSING conflicts
    CDR DELETEROWEXISTS conflicts         1
    CDR DELETEROWMISSING conflicts        1
```

GGSCI

You can view CDR statistics from GGSCI by using the `STATS REPLICAT` command with the `REPORTCDR` option:

```
STATS REPLICAT group, REPORTCDR
```

Column-conversion Functions

The following CDR statistics can be retrieved and mapped to an exceptions table or used in other Oracle GoldenGate parameters that accept input from column-conversion functions, as appropriate.

- Number of conflicts that Replicat detected
- Number of resolutions that Replicat resolved
- Number of resolutions that Replicat could not resolve

To retrieve these statistics, use the `@GETENV` column-conversion function with the `'STATS'` or `'DELTASTATS'` information type. The results are based on the current Replicat session. If Replicat stops and restarts, it resets the statistics.

You can return these statistics for a specific table or set of wildcarded tables:

```
@GETENV ('STATS','TABLE','SCHEMA.TABLNAME','CDR_CONFLICTS')
@GETENV ('STATS','TABLE','SCHEMA.TABLNAME','CDR_RESOLUTIONS_SUCCEEDED')
@GETENV ('STATS','TABLE','SCHEMA.TABLNAME','CDR_RESOLUTIONS_FAILED')
```

You can return these statistics for all of the tables in all of the `MAP` statements in the Replicat parameter file:

```
@GETENV ('STATS','CDR_CONFLICTS')
@GETENV ('STATS','CDR_RESOLUTIONS_SUCCEEDED')
@GETENV ('STATS','CDR_RESOLUTIONS_FAILED')
```

The `'STATS'` information type in the preceding examples can be replaced by `'DELTASTATS'` to return the requested counts since the last execution of `'DELTASTATS'`.

For more information about `@GETENV`, see *Parameters and Functions Reference for Oracle GoldenGate*.

CDR Example 1: All Conflict Types with USEMAX, OVERWRITE, DISCARD

This example resolves all conflict types by using the `USEMAX`, `OVERWRITE`, and `DISCARD` resolutions.

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
    name varchar2(30) primary key,
    phone varchar2(10),
    address varchar2(100),
    salary number,
    balance number,
    comment varchar2(100),
    last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement with Conflict Resolution Specifications

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS (ON UPDATE ALL, ON DELETE ALL),
    RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
    RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (last_mod_time)),
    RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)),
    RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)),
    RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)),
    );
```

Description of MAP Statement

The following describes the `MAP` statement:

- Per `COMPARECOLS`, use the before image of all columns in the trail record in the Replicat `WHERE` clause for updates and deletes.
- Per `DEFAULT`, use all columns as the column group for all conflict types; thus the resolution applies to all columns.
- For an `INSERTROWEXISTS` conflict, use the `USEMAX` resolution: If the row exists during an insert, use the `last_mod_time` column as the resolution column for deciding which is the greater value: the value in the trail or the one in the database. If the value in the trail is greater, apply the record but change the insert to an update. If the database value is higher, ignore the record.
- For an `UPDATEROWEXISTS` conflict, use the `USEMAX` resolution: If the row exists during an update, use the `last_mod_time` column as the resolution column: If the value in the trail is greater, apply the update.
- If you use `USEMIN` or `USEMAX`, and the values are exactly the same, then `RESOLVECONFLICT` isn't triggered and the incoming row is ignored. If you use `USEMINEQ` or `USEMAXEQ`, and the values are exactly the same, then the resolution is triggered.

- For a `DELETEROWEXISTS` conflict, use the `OVERWRITE` resolution: If the row exists during a delete operation, apply the delete.
- For an `UPDATEROWMISSING` conflict, use the `OVERWRITE` resolution: If the row does not exist during an update, change the update to an insert and apply it.
- For a `DELETROWMISSING` conflict use the `DISCARD` resolution: If the row does not exist during a delete operation, discard the trail record.

> **Note:**
>
> As an alternative to `USEMAX`, you can use the `USEMAXEQ` resolution to apply a `>=` condition. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see Configuring the Oracle GoldenGate Parameter Files for Error Handling.

INSERTROWEXISTS with the USEMAX Resolution

For this example, the `USEMAX` resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve an insert where the row exists in the source and target, but some or all row values are different.

Table 10-2 INSERTROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments
Before image in trail	`None (row was inserted on the source).`	N/A

After image in trail	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=100`
	`comment=NULL`
	`last_mod_time='9/1/10 3:00'`
		`last_mod_time='9/1/10 3:00` is the after image of the resolution column. Since there is an after image, this will be used to determine the resolution.

Target database image	`name='Mary'`
	`phone='111111'`
	`address='Ralston'`
	`salary=200`
	`balance=500`
	`comment='aaa'`
	`last_mod_time='9/1/10 1:00'`
		`last_mod_time='9/1/10 1:00` is the current image of the resolution column in the target against which the resolution column value in the trail is compared.

Table 10-2 (Cont.) INSERTROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments

Initial `INSERT` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'Mary'`
	`2)'1234567890'`
	`3)'Oracle Pkwy'`
	`4)100`
	`5)100`
	`6)NULL`
	`7)'9/1/10 3:00'`
		This SQL returns a uniqueness conflict on 'Mary'.

`UPDATE` applied by Replicat to resolve the conflict	SQL bind variables:
	`1)'1234567890'`
	`2)'Oracle Pkwy'`
	`3)100`
	`4)100`
	`5)NULL`
	`6)'9/1/10 3:00'`
	`7)'Mary'`
	`8)'9/1/10 3:00'`
		Because `USEMAX` is specified for `INSERTROWEXISTS`, Replicat converts the insert to an update, and it compares the value of `last_mod_time` in the trail record with the value in the database. The value in the record is greater, so the after images for columns in the trail file are applied to the target.

UPDATEROWEXISTS with the USEMAX Resolution

For this example, the `USEMAX` resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve an update where the row exists in the source and target, but some or all row values are different.

Table 10-3 UPDATEROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments

Before image in trail	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=100`
	`comment=NULL`
	`last_mod_time='9/1/10 3:00'`
		`last_mod_time='9/1/10 3:00` is the before image of the resolution column.

After image in trail	`phone='222222'`
	`address='Holly'`
	`last_mod_time='9/1/10 5:00'`
		`last_mod_time='9/1/10 5:00` is the after image of the resolution column. Since there is an after image, this will be used to determine the resolution.

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 6:00'`
		`last_mod_time='9/1/10 6:00` is the current image of the resolution column in the target against which the resolution column value in the trail is compared.

Table 10-3 (Cont.) UPDATEROWEXISTS Conflict with USEMAX Resolution

Image	SQL	Comments

Initial `UPDATE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'222222'`
	`2)'Holly'`
	`3)'9/1/10 5:00'`
	`4)'Mary'`
	`5)'1234567890'`
	`6)'Oracle Pkwy'`
	`7)100`
	`8)100`
	`9)NULL`
	`10)'9/1/10 3:00'`
		This SQL returns a no-data-found error because the values for the `balance`, `comment`, and `last_mod_time` are different in the target.
		All columns are used in the `WHERE` clause because the `COMPARECOLS` statement is set to `ALL`.

`UPDATE` applied by Replicat to resolve the conflict	SQL bind variables:
	`1)'Mary'`
	`2)'222222'`
	`3)'Holly'`
	`4)100`
	`5)100`
	`6)NULL`
	`7)'9/1/10 5:00'`
	`8)'Mary'`
	`9)'9/1/10 5:00'`
		Because the after value of `last_mod_time` in the trail record is less than the current value in the database, the database value is retained. Replicat applies the operation with a `WHERE` clause that contains the primary key plus a `last_mod_time` value set to less than `9/1/10 5:00`. No rows match this criteria, so the statement fails with a "data not found" error, but Replicat ignores the error because a `USEMAX` resolution is expected to fail if the condition is not satisfied.

UPDATEROWMISSING with OVERWRITE Resolution

For this example, the `OVERWRITE` resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the case where the target row is missing. The logical resolution, and the one used, is to overwrite the row into the target so that both databases are in sync again.

Table 10-4 UPDATEROWMISSING Conflict with OVERWRITE Resolution

Image	SQL	Comments

Before image in trail	`name='Jane'`
	`phone='333'`
	`address='Oracle Pkwy'`
	`salary=200`
	`balance=200`
	`comment=NULL`
	`last_mod_time='9/1/10 7:00'`
		N/A

After image in trail	`phone='4444'`
	`address='Holly'`
	`last_mod_time='9/1/10 8:00'`	

Target database image	`None (row for Jane is missing)`	

Table 10-4 (Cont.) UPDATEROWMISSING Conflict with OVERWRITE Resolution

Image	SQL	Comments

Initial `UPDATE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'4444'`
	`2)'Holly'`
	`3)'9/1/10 8:00'`
	`4)'Jane'`
	`5)'333'`
	`6)'Oracle Pkwy'`
	`7)200`
	`8)200`
	`9)NULL`
	`10)'9/1/10 7:00'`
		This SQL returns a no-data-found error. All columns are used in the `WHERE` clause because the `COMPARECOLS` statement is set to `ALL`.

`INSERT` applied by Replicat to resolve the conflict	SQL bind variables:
	`1)'Jane'`
	`2)'4444'`
	`3)'Holly'`
	`4)200`
	`5)200`
	`6)NULL`
	`7)'9/1/10 8:00'`
		The update is converted to an insert because `OVERWRITE` is the resolution. The after image of a column is used if available; otherwise the before image is used.

DELETEROWMISSING with DISCARD Resolution

For this example, the `DISCARD` resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the case where the target row is missing. In the case of a delete on the source, it is acceptable for the target row not to exist (it would need to be deleted anyway), so the resolution is to discard the `DELETE` operation that is in the trail.

Table 10-5 DELETEROWMSING Conflict with DISCARD Resolution

Image	SQL	Comments

Before image in trail	`name='Jane'`
	`phone='4444'`
	`address='Holly'`
	`salary=200`
	`balance=200`
	`comment=NULL`
	`last_mod_time='9/1/10 8:00'`
		N/A

After image in trail	`None`	N/A
Target database image	`None (row missing)`	N/A

Table 10-5 (Cont.) DELETEROWMSING Conflict with DISCARD Resolution

Image	SQL	Comments

Initial `DELETE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'Jane'`
	`2)'4444'`
	`3)'Holly'`
	`4)200`
	`5)200`
	`6)NULL`
	`7)'9/1/10 8:00'`
		This SQL returns a no-data-found error. All columns are used in the `WHERE` clause because the `COMPARECOLS` statement is set to `ALL`.

SQL applied by Replicat to resolve the conflict	`None`	Because `DISCARD` is specified as the resolution for `DELETEROWMISSING`, so the delete from the trail goes to the discard file.

DELETEROWEXISTS with OVERWRITE Resolution

For this example, the `OVERWRITE` resolution is illustrated with the applicable before and after images for the record in the trail and in the database. It shows how to resolve the case where the source row was deleted but the target row exists. In this case, the `OVERWRITE` resolution applies the delete to the target.

Table 10-6 DELETEROWEXISTS Conflict with OVERWRITE Resolution

Image	SQL	Comments

Before image in trail	`name='Mary'`
	`phone='222222'`
	`address='Holly'`
	`salary=100`
	`balance=100`
	`comment=NULL`
	`last_mod_time='9/1/10 5:00'`
		N/A

After image in trail	`None`	N/A

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=600`
	`comment=com`
	`last_mod_time='9/1/10 7:00'`
		The row exists on the target, but the `phone`, `address`, `balance`, `comment`, and `last_mod_time` columns are different from the before image in the trail.

Initial `DELETE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'Mary'`
	`2)'222222'`
	`3)'Holly'`
	`4)100`
	`5)100d`
	`6)NULL`
	`7)'9/1/10 5:00'`
		All columns are used in the `WHERE` clause because the `COMPARECOLS` statement is set to `ALL`.
		A no-data-found error occurs because of the difference between the before and current values.

Table 10-6 (Cont.) DELETEROWEXISTS Conflict with OVERWRITE Resolution

Image	SQL	Comments
`DELETE` applied by Replicat to resolve the conflict	SQL bind variables: `1)'Mary'`	Because `OVERWRITE` is the resolution. the `DELETE` is applied using only the primary key (to avoid an integrity error).

CDR Example 2: UPDATEROWEXISTS with USEDELTA and USEMAX

This example resolves the condition where a target row exists on `UPDATE` but non-key columns are different, and it uses two different resolution types to handle this condition based on the affected column.

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
    name varchar2(30) primary key,
    phone varchar2(10),
    address varchar2(100),
    salary number,
    balance number,
    comment varchar2(100),
    last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS
    (ON UPDATE KEYINCLUDING (address, phone, salary, last_mod_time),
    ON DELETE KEYINCLUDING (address, phone, salary, last_mod_time)),
    RESOLVECONFLICT (
    UPDATEROWEXISTS,
    (delta_res_method, USEDELTA, COLS (salary)),
    (DEFAULT, USEMAX (last_mod_time)));
```

Description of MAP Statement

For an `UPDATEROWEXISTS` conflict, where a target row exists on `UPDATE` but non-key columns are different, use two different resolutions depending on the column:

- Per the `delta_res_method` resolution, use the `USEDELTA` resolution logic for the `salary` column so that the change in value will be added to the current value of the column.
- Per `DEFAULT`, use the `USEMAX` resolution logic for all other columns in the table (the default column group), using the `last_mod_time` column as the resolution column. This column is updated with the current time whenever the row is modified; the value of this column in the trail is compared to the value in the target. If the value of `last_mod_time` in the trail record is greater than the current value of `last_mod_time` in the target database, the changes to `name`, `phone`, `address`, `balance`, `comment` and `last_mod_time` are applied to the target.

Per `COMPARECOLS`, use the primary key (`name` column) plus the `address`, `phone`, `salary`, and `last_mod_time` columns as the comparison columns for conflict detection for `UPDATE` and `DELETE` operations. (The `balance` and `comment` columns are not compared.)

> **Note:**
>
> As an alternative to `USEMAX`, you can use the `USEMAXEQ` resolution to apply a `>=` condition. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see Configuring the Oracle GoldenGate Parameter Files for Error Handling.

Table 10-7 UPDATEROWEXISTS with USEDELTA and USEMAX

Image	SQL	Comments

Before image in trail	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=100`
	`comment=NULL`
	`last_mod_time='9/1/10 3:00'`
		`last_mod_time='9/1/10 3:00` is the before image of the resolution column for the `USEMAX` resolution.
		`salary=100` is the before image for the `USEDELTA` resolution.

After image in trail	`phone='222222'`
	`address='Holly'`
	`salary=200`
	`comment='new'`
	`last_mod_time='9/1/10 5:00'`
		`last_mod_time='9/1/10 5:00` is the after image of the resolution column for `USEMAX`. Since there is an after image, this will be used to determine the resolution.

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=600`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 4:00'`
		`last_mod_time='9/1/10 4:00` is the current image of the resolution column in the target against which the resolution column value in the trail is compared.
		`salary=600` is the current image of the target column for the `USEDELTA` resolution.

Initial `UPDATE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'222222'`
	`2)'Holly'`
	`3)200`
	`4)'new'`
	`5)'9/1/10 5:00'`
	`6)'Mary'`
	`7)'1234567890'`
	`8)'Oracle Pkwy'`
	`9)100`
	`10)'9/1/10 3:00'`
		This SQL returns a no-data-found error because the values for the `salary` and `last_mod_time` are different. (The values for `comment` and `balance` are also different, but these columns are not compared.)

Table 10-7 (Cont.) UPDATEROWEXISTS with USEDELTA and USEMAX

Image	SQL	Comments

`UPDATE` applied by Replicat to resolve the conflict for `salary`, using `USEDELTA`.	SQL bind variables:
	`1)200`
	`2)100`
	`3)'Mary'`
		Per `USEDELTA`, the difference between the after image of `salary` (200) in the trail and the before image of `salary` (100) in the trail is added to the current value of `salary` in the target (600). The result is 700.
		`600 + (200 - 100) = 700`

`UPDATE` applied by Replicat to resolve the conflict for the default columns, using `USEMAX`.	SQL bind variables:
	`1)'222222'`
	`2)'Holly'`
	`3)'new'`
	`4)'9/1/10 5:00'`
	`5)'Mary'`
	`6)'9/1/10 5:00'`
		Per `USEMAX`, because the after value of `last_mod_time` in the trail record is greater than the current value in the database, the row is updated with the after values from the trail record.
		Note that the `salary` column is not set here, because it is resolved with the `UPDATE` from the `USEDELTA` resolution.

CDR Example 3: UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

This example resolves the conflict where a target row exists on `UPDATE` but non-key columns are different, and it uses three different resolution types to handle this condition based on the affected column.

Table Used in this Example

The examples assume identical Oracle databases.

```
CREATE TABLE tgt(
    name varchar2(30) primary key,
    phone varchar2(10),
    address varchar2(100),
    salary number,
    balance number,
    comment varchar2(100),
    last_mod_time timestamp);
```

At the source database, all columns are supplementally logged:

```
ADD TRANDATA scott.src, COLS (name, phone, address, salary, balance, comment,
last_mod_time);
```

MAP Statement

```
MAP fin.src, TARGET fin.tgt,
    COMPARECOLS
    (ON UPDATE ALLEXCLUDING (comment)),
    RESOLVECONFLICT (
    UPDATEROWEXISTS,
    (delta_res_method, USEDELTA, COLS (salary, balance)),
    (max_res_method, USEMAX (last_mod_time), COLS (address, last_mod_time)),
    (DEFAULT, IGNORE));
```

Description of MAP Statement

- For an `UPDATEROWEXISTS` conflict, where a target row exists on `UPDATE` but non-key columns are different, use two different resolutions depending on the column:
  - Per the `delta_res_method` resolution, use the `USEDELTA` resolution logic for the `salary` and `balance` columns so that the change in each value will be added to the current value of each column.
  - Per the `max_res_method` resolution, use the `USEMAX` resolution logic for the `address` and `last_mod_time` columns. The `last_mod_time` column is the resolution column. This column is updated with the current time whenever the row is modified; the value of this column in the trail is compared to the value in the target. If the value of `last_mod_time` in the trail record is greater than the current value of `last_mod_time` in the target database, the changes to `address` and `last_mod_time` are applied to the target; otherwise, they are ignored in favor of the target values.
  - Per `DEFAULT`, use the `IGNORE` resolution logic for the remaining columns (`phone` and `comment`) in the table (the default column group). Changes to these columns will always be ignored by Replicat.
- Per `COMPARECOLS`, use all columns except the `comment` column as the comparison columns for conflict detection for `UPDATE` operations. Comment will not be used in the `WHERE` clause for updates, but all other columns that have a before image in the trail record will be used.

> **Note:**
>
> As an alternative to `USEMAX`, you can use the `USEMAXEQ` resolution to apply a `>=` condition. For more information, see *Parameters and Functions Reference for Oracle GoldenGate*.

Error Handling

For an example of error handling to an exceptions table, see Configuring the Oracle GoldenGate Parameter Files for Error Handling.

Table 10-8 UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments

Before image in trail	`name='Mary'`
	`phone='1234567890'`
	`address='Oracle Pkwy'`
	`salary=100`
	`balance=100`
	`comment=NULL`
	`last_mod_time='9/1/10 3:00`
		`last_mod_time='9/1/10 3:00` is the before image of the resolution column for the `USEMAX` resolution.
		`salary=100` and `balance=100` are the before images for the `USEDELTA` resolution.

Table 10-8 (Cont.) UPDATEROWEXISTS with USEDELTA, USEMAX, and IGNORE

Image	SQL	Comments

After image in trail	`phone='222222'`
	`address='Holly'`
	`salary=200`
	`comment='new'`
	`last_mod_time='9/1/10 5:00'`
		`last_mod_time='9/1/10 5:00'` is the after image of the resolution column for `USEMAX`. Since there is an after image, this will be used to determine the resolution.
		`salary=200` is the only after image available for the `USEDELTA` resolution. For `balance`, the before image will be used in the calculation.

Target database image	`name='Mary'`
	`phone='1234567890'`
	`address='Ralston'`
	`salary=600`
	`balance=600`
	`comment='com'`
	`last_mod_time='9/1/10 4:00'`
		`last_mod_time='9/1/10 4:00'` is the current image of the resolution column in the target against which the resolution column value in the trail is compared for `USEMAX`.
		`salary=600` and `balance=600` are the current images of the target columns for `USEDELTA`.

Initial `UPDATE` applied by Replicat that detects the conflict	SQL bind variables:
	`1)'222222'`
	`2)'Holly'`
	`3)200`
	`4)'new'`
	`5)'9/1/10 5:00'`
	`6)'Mary'`
	`7)'1234567890'`
	`8)'Oracle Pkwy'`
	`9)100`
	`10)100`
	`11)'9/1/10 3:00'`
		This SQL returns a no-data-found error because the values for the `address`, `salary`, `balance` and `last_mod_time` columns are different.

`UPDATE` applied by Replicat to resolve the conflict for `salary`, using `USEDELTA`.	SQL bind variables:
	`1)200`
	`2)100`
	`3)'Mary'`
		For `salary`, there is a difference of 100, but there was no change in value for `balance`, so it is not needed in the update SQL. Per `USEDELTA`, the difference (delta) between the after (200) image and the before image (100) of `salary` in the trail is added to the current value of `salary` in the target (600). The result is 700.

`UPDATE` applied by Replicat to resolve the conflict for `USEMAX`.	SQL bind variables:
	`1)'Holly'`
	`2)'9/1/10 5:00'`
	`3)'Mary'`
	`4)'9/1/10 5:00'`
		Because the after value of `last_mod_time` in the trail record is greater than the current value in the database, that column plus the `address` column are updated with the after values from the trail record.
		Note that the `salary` column is not set here, because it is resolved with the `UPDATE` from the `USEDELTA` resolution.

`UPDATE` applied by Replicat for `IGNORE`.	SQL bind variables:
	`1)'222222'`
	`2)'new'`
	`3)'Mary'`
		`IGNORE` is specified for the `DEFAULT` column group (`phone` and `comment`), so no resolution SQL is applied.

11

Autonomous Database

This section provides details about configuring Oracle GoldenGate with Oracle Autonomous Database, and using Extract and Replicat processes with Autonomous Database instances.

Using Oracle GoldenGate with Autonomous Database

You can replicate data to Oracle Autonomous Database using Oracle GoldenGate.

About Capturing and Replicating Data Using Autonomous Databases

Oracle GoldenGate can be used to replicate data into Oracle Autonomous Database from any certified source platform and replicate data from Oracle Autonomous Database to any certified target platform.

Use Case: When Using Oracle GoldenGate with Autonomous Databases

Oracle GoldenGate can be configured to support the following scenarios in the Oracle Autonomous Database:

- **Scalable Active-Active architecture**: Synchronize changes made across two or more databases to scale out workloads, provide increase resilience and near instantaneous failover across multiple data centers or regions.
- **Real-Time Data Warehouse**: Provide continuous, real-time capture and delivery of changed data between Oracle Autonomous Database systems.
- **Big Data Integration**: With Oracle GoldenGate for Big Data you can replicate data from the Oracle Autonomous Database to provide real-time streaming integration to all platforms supported by Big Data targets.
- **Real-Time Streaming Analytics**: Oracle GoldenGate integrates seamlessly with Oracle Stream Analytics to enable users to identify events of interest by executing queries against event streams in real time. It allows creating custom operational dashboards that provide real-time monitoring, transform streaming data, or raise alerts based on stream analysis.
- **Hybrid Replication**: Oracle GoldenGate replicates data from the Oracle Autonomous Database instance back to on-premise or to another cloud database or platform.

> **Note:**
>
> Oracle GoldenGate cannot be used to extract from the Always Free Autonomous Database due to lack of a supplemental logging feature in the database.

Details of Support When Using Oracle GoldenGate with Autonomous Databases

Review the supported data types and limitations before replicating data to or from an Oracle Autonomous Database.

Oracle GoldenGate Replicat Limitations for Autonomous Databases

These are the limitations of Oracle GoldenGate when replicating to or from the Oracle Autonomous Database.

Supported Replicats
To replicate data into an Oracle Autonomous Database you must use Parallel Replicat (integrated or non-integrated mode) or Integrated Replicat.

Data Type Limitations for DDL and DML Replication

See the section .

Also see Non-Supported Oracle Data Types in the *Autonomous Database on Dedicated Exadata Infrastructure Documentation* and Data Types in the *Using Oracle Autonomous Database Serverless* guide.

DDL replication is supported depending on the restrictions in the Autonomous Databases.

Details of Support for Archived Log Retention

The two types of Autonomous Databases, Oracle Autonomous Database Serverless and Oracle Autonomous Database on Dedicated Exadata Infrastructure have different log retention behavior.

- Oracle Autonomous Database Serverless: Archived log files are kept in Fast Recovery Area (FRA) for up to 48 hours. After that, it is purged and the archived log files are moved to NFS mount storage, which is accessible by logminer. Three copies are created. The logminer should be able to access any of the copies. This is transparent to Oracle GoldenGate Extract. After it reaches 7 days, the NFS mounted copy is permanently removed. The Extract abends with the `archived log unavailable` error if the required archived log file is older than 7 days.
- Oracle Autonomous Database on Dedicated Exadata Infrastructure: When Oracle Autonomous Data Guard or Oracle GoldenGate is enabled, archived log files are kept in Fast Recovery Area (FRA) for up to 7 days. After that, the files are purged. There is no NFS mount location available for logminer to access archived log files that are older than 7 days. The Extract abends with the `archived log unavailable` error if the required archived log file is older than 7 days.

> **Note:**
>
> If the database instance is closed for more than 15 minutes, then the retention time is set back to 3 days. This implies that retention of archived log files is confirmed only for 3 days, regardless of whether the database instance is closed. The files are retained for 7 days only if the database instance is not closed.

> **Note:**
>
> If Extract abends with the archived log unavailable error, you will not be able to restart it, and the data that was not captured in those archive logs cannot be read. You will need to drop and recreate the extract, and resynchronize the data on any target systems.

Configuring Extract to Capture from an Autonomous Database

Oracle Autonomous Database has a tight integration with Oracle GoldenGate. There are a number of differences when setting up Extract for an Autonomous database instance compared to a traditional Oracle Database.

Oracle Autonomous Database security has been enhanced to ensure that Extract is only able to capture changes from the specific tenant it is connected to. Therefore, Downstream Integrated Extract is not supported.

Establishing Oracle GoldenGate Credentials

To capture from an Autonomous Database only the `GGADMIN` account is used. The `GGADMIN` account is created inside the database when the Autonomous Database is provisioned and already has all the necessary permissions for both Extract and Replicat processes. This account is locked. It must be unlocked before it can be used with Oracle GoldenGate. This account is the same account used for both Extracts and Replicats in the Autonomous Database.

Run the `ALTER USER` command to unlock the `ggadmin` user and set the password for it. See Creating Users with Autonomous Database with Client-Side Tools.

This `ALTER USER` command must be run by the `admin` account user for Autonomous Databases.

```
ALTER USER ggadmin IDENTIFIED BY PASSWORD ACCOUNT UNLOCK;
```

Prerequisites for Configuring Oracle GoldenGate Extract to Capture from Autonomous Databases

Prior to configuring and starting the Extract process to capture from the Autonomous Database, make sure that the following requirements are met:

- Oracle Autonomous Database environment is provisioned and running.
- Autonomous Database-level supplemental logging should be enabled by the `ADMIN` or `GGADMIN`.

Configuring Autonomous Database Supplemental Logging for Extract

To add minimal supplemental logging to your Autonomous Database instance, log into the instance as `GGADMIN` or `ADMIN` account and execute the following commands:

```
ALTER PLUGGABLE DATABASE ADD SUPPLEMENTAL LOG DATA;
```

To `DROP` Autonomous Database-level supplemental logging incase you decide to stop capturing from that database instance:

```
ALTER PLUGGABLE DATABASE DROP SUPPLEMENTAL
        LOG DATA;
```

You can verify that the Autonomous Database-level supplemental logging is configured properly by issuing this SQL statement:

```
SELECT MINIMAL FROM dba_supplemental_logging;
```

The output for this statement is:

```
MINIMAL
-------
YES
```

The `MINIMAL` column will be `YES` if supplemental logging has been correctly set for this Autonomous Database instance.

Configure Extract to Capture from an Autonomous Database

Following are the steps to configure an Extract to capture from an Oracle Autonomous Database :

1. Install Oracle GoldenGate for your Oracle Autonomous Database instance.
2. Obtain Autonomous Database Client Credentials.

   To establish connection to your Oracle Autonomous Database instance, download the client credentials file. To download client credentials, you can use the Oracle Cloud Infrastructure Console or Database Actions Launchpad. See Downloading Client Credentials (Wallets).

   > **Note:**
   >
   > If you do not have administrator access to the Autonomous Database you should ask your service administrator to download and provide the credentials files to you.

   The following steps use the **Database Actions Launchpad** to download the client credentials.

   a. Log in to your Oracle Autonomous Database account.
   b. From the **Database Instance** page, click **Database Actions**. This launches the Database Actions Launchpad. The Launchpad attempts to log you into the database as ADMIN. If that is not successful, you will be prompted for your database ADMIN username and password.
   c. On the **Database Actions** Launchpad, under **Administration**, click **Download Client Credentials (Wallets)**.
   d. Enter a password to secure your Client Credentials zip file and click **Download**.

> **Note:**
>
> The password you provide when you download the wallet protects the downloaded Client Credentials wallet.

e. Save the credentials `zip` file to your local system.

The credentials `zip` file contains the following files:

- `cwallet.sso`
- `ewallet.p12`
- `keystore.jks`
- `ojdbc.properties`
- `sqlnet.ora`
- `tnsnames.ora`
- `truststore.jks`
- `ewallet.pem`
- `README.txt`

Refer and update (if required) the `sqlnet.ora` and `tnsnames.ora` files while configuring Oracle GoldenGate to work with the Autonomous Database instance.

3. Configure the server where Oracle GoldenGate is running to connect to the Autonomous Database instance.
   a. Log in to the server where Oracle GoldenGate was installed.
   b. Transfer the credentials `zip` file that you downloaded from Oracle Autonomous database instance to the Oracle GoldenGate server.
   c. In the Oracle GoldenGate server, unzip the credentials file into a new directory, for example: `/u02/data/adwc_credentials`. This is your key directory.
   d. To configure the connection details, open your `tnsnames.ora` file from the Oracle client location in the Oracle GoldenGate instance.
   e. Use the connection string with the `LOW` consumer group `*dbname*_low`, for example, `graphdb1_low`, and move it to your local `tnsnames.ora` file.

      See *Local Naming Parameters in the tnsnames.ora File* chapter in the *Oracle Database Net Services Reference* guide.

> **Note:**
>
> The `tnsnames.ora` file provided with the credentials file contains three database service names identifiable as:
>
> ```
> ADWC_Database_Name_low
> ADWC_Database_Name_medium
> ADWC_Database_Name_high
> ```
>
> Oracle recommends that you use *`ADWC_Database_Name_low`* with Oracle GoldenGate. See Predefined Database Service Names for Autonomous Database in the *Using Oracle Autonomous Database Serverless* guide or Predefined Database Service Names for Autonomous Databases for Oracle Autonomous Database on Dedicated Exadata Infrastructure.

f. Edit the `tnsnames.ora` file in the Oracle GoldenGate instance to include the connection details available in the `tnsnames.ora` file in your key directory (the directory where you unzipped the credentials `zip` file downloaded from the Autonomous Database).

```
Sample Connection String
adw1_low. = (description=
                 (retry_count=20)(retry_delay=3)
                 (address=(protocol=tcps)(port=1522)(host=adb-
preprod.us-phoenix-1.oraclecloud.com))

(connect_data=(service_name=okd2ybgcz4mjx94_graphdb1_low.adb.oraclecloud
.com))
                 (security=(ssl_server_cert_dn="CN=adwc-preprod.uscom-
east-1.oraclecloud.com,OU=Oracle BMCS US,O=Oracle Corporation,L=Redwood
City,ST=California,C=US"))
               )
```

If the database is within a firewall protected environment, you might not have direct access to the database. With an existing HTTP Proxy, you can pass the firewall with the following modifications to the `sqlnet.ora` and `tnsnames.ora`:

- *`sqlnet parameters`*
- *`address modification of tns_alias`*

If Extract becomes unresponsive due to a network timeout or connection loss, then you can add the following into the connection profile in the `tnsnames.ora` file:

```
(DESCRIPTION = (RECV_TIMEOUT=30) (ADDRESS_LIST =
      (LOAD_BALANCE=off)(FAILOVER=on)(CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
(ADDRESS = (PROTOCOL = TCP)(HOST = adb-preprod.us-
phoenix-1.oraclecloud.com)(PORT = 1522))
```

   g. To configure the wallet, create a `sqlnet.ora` file in the Oracle client location in the Oracle GoldenGate instance.

      ```
      cd /u02/data/oci/network/admin
      ls
      sqlnet.ora tnsnames.ora
      ```

      See Autonomous Database Client Credentials in *Using Oracle GoldenGate on Oracle Cloud Marketplace*.

   h. Edit this `sqlnet.ora` file to include your key directory.

      ```
      WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
      (DIRECTORY="/u02/data/adwc_credentials")))
      SSL_SERVER_DN_MATCH=yes
      ```

4. Start GGSCI.
   `./ggsci`
5. Create credentials for the Extract database (or a user with same privileges). In this case, `GGADMIN` is the user and will be used to connect to the Autonomous Database, and perform commands that require a database connection. It will also be used in the `USERIDALIAS` parameter for the Extract database connection.

   ```
   ALTER CREDENTIALSTORE ADD USER
   ggadmin@dbgraph1_low PASSWORD complex_password alias adb_alias
   ```

6. Connect to the database using `DBLOGIN`. The `DBLOGIN` user should be the `adb_alias` account user.

   ```
   DBLOGIN USERIDALIAS adb_alias
   ```

7. Configure supplemental logging on the tables, which you want to capture using `ADD TRANDATA` or `ADD SCHEMATRANDATA`. Remember that you are connected directly to the database instance, so there is no need to include the database name in these commands. Here's an example:

   ```
   ADD TRANDATA HR.EMP
   ```

   or

   ```
   ADD SCHEMATRANDATA HR
   ```

   See Prerequisites for Configuring Oracle GoldenGate Extract to Capture from Autonomous Databases.

8. Add heartbeat table.

   ```
   ADD HEARTBEATTABLE
   ```

9. Add and configure an Extract to capture from the Oracle Autonomous Database instance. See Add the Primary Extract for steps to create an Extract.

Oracle GoldenGate Extract is designed to work with the Oracle Autonomous Database instance to ensure that it only captures from a specific database instance. This means that the database instance name is not needed for any `TABLE` or `MAP` statements.

The following example creates an Extract (required for capturing from an Oracle Autonomous Database) called `exte`, and instructs it to begin now.

```
ADD EXTRACT exte, INTEGRATED TRANLOG, BEGIN NOW
```

To capture specific tables, use the two part object names.. For example, to capture from the table `HR.EMP`, in your Oracle Autonomous Database instance, use this entry in the Extract parameter file.

```
TABLE HR.EMP;
```

If you want to replicate `HR.EMP` into `COUNTRY.EMPLOYEE`, then your map statement would look like this:

```
MAP HR.EMP, TARGET COUNTRY.EMPLOYEE;
```

10. Register Extract with the Oracle Autonomous Database instance. For example, to register an Extract named `exte`, use the following command:

    ```
    REGISTER EXTRACT exte DATABASE
    ```

11. You can now start your Extract and perform data replication to the Oracle Autonomous Database instance. Here's an example:

    ```
    START EXTRACT exte
    ```

    This completes the process of configuring an Extract for Oracle Autonomous Database and you can use it like any other Extract process.

Configuring Replicat to Apply to an Autonomous Database

You can replicate into the Autonomous Database from any source database or platform that is certified by Oracle GoldenGate.

Topics:

Prerequisites for Configuring Oracle GoldenGate Replicat to an Autonomous Database

Learn about the prerequisites for configuring Oracle GoldenGate data replication to Autonomous Databases.

You should have the following details available with you:

- Your source database with Oracle GoldenGate Extract processes configured and writing trails to where the Replicat is running to apply data to the Autonomous Database target.
- Oracle Autonomous Database environment is provisioned and running.

To deliver data to the Autonomous Database instance using Oracle GoldenGate, perform the following tasks:

Configure Oracle GoldenGate Replicat for an Autonomous Database

Learn the steps to configure Oracle GoldenGate Replicat for an Autonomous Databases.

Here are the steps to complete the configuration tasks:

> **Note:**
>
> Instructions are based on the assumption that the source environment is already configured. Learn the steps required to configure replication into the Autonomous Database environment.

1. For Oracle GoldenGate on-premises, make sure that Oracle GoldenGate is installed.

   Oracle GoldenGate Classic Architecture support Autonomous Database capture using Marketplace for Oracle Autonomous Database Serverless.
2. Start GGSCI.
   ```
   ./ggsci
   ```
3. The Autonomous Database instance has a pre-created user created for Oracle GoldenGate on-premise called `ggadmin`. The `ggadmin` user has been granted the required privileges for Replicat to work. This is the user where any objects used for Oracle GoldenGate processing will be stored, like the checkpoint table and heartbeat objects. By default, this user is locked. To unlock the `ggadmin` user, connect to the Oracle Autonomous Database instance as the `ADMIN` user using any SQL client tool. See Create Users on Autonomous Database with Database Actions.
4. Run the `ALTER USER` command to unlock the `ggadmin` user and set the password for it. This will be used in GGSCI for any `DBLOGIN` operations on the Autonomous Database. It will be used in Replicat to allow Oracle GoldenGate to connect to the Autonomous Database and apply data. See Create Users on Autonomous Database with Database Actions.
   ```
   ALTER USER ggadmin IDENTIFIED BY p0$$word ACCOUNT UNLOCK;
   ```

Obtain the Autonomous Database Client Credentials

Learn how to establish connection to your Autonomous Databases.

To establish a connection with an Oracle Autonomous Database instance, you need to download the client credentials files. There are two ways to download the client credentials files: the Oracle Cloud Infrastructure Console or Database Actions Launchpad. See Downloading Client Credentials (Wallets).

> **Note:**
>
> If you do not have administrator access to the Oracle Autonomous Database, you should ask your service administrator to download and provide the credentials files to you.

The following steps use the **Database Actions Launchpad** to download the client credentials files.

1. Log into your Autonomous Database account.
2. From the **Database Instance** page, click **Database Actions**. This launches the Database Actions Launchpad. The Launchpad attempts to log you into the database as ADMIN. If that is not successful, you will be prompted for your database ADMIN username and password.
3. On the **Database Actions** Launchpad, under **Administration**, click **Download Client Credentials (Wallets)**.
4. Enter a password to secure your Client Credentials zip file and click **Download**.

> **Note:**
>
> The password you provide when you download the wallet protects the downloaded Client Credentials wallet.

5. Save the credentials ZIP file to your local system. The credentials ZIP file contains the following files:

- `cwallet.sso`
- `ewallet.p12`
- `keystore.jks`
- `ojdbc.properties`
- `sqlnet.ora`
- `tnsnames.ora`
- `truststore.jks`
- `ewallet.pem`
- `README.txt`

Refer and update (if required) the `sqlnet.ora` and `tnsnames.ora` files while configuring Oracle GoldenGate to work with the Oracle Autonomous Database instance.

Configure Replicat to Apply to an Autonomous Database

This section assumes that the source environment is already configured and provides the steps required to establish replication in the Oracle Autonomous Database environment.

In the Oracle GoldenGate instance, you need to complete the following:

1. Follow the steps given in Prerequisites for Configuring Oracle GoldenGate Replicat to an Autonomous Database.
2. Follow the steps given in Configure Oracle GoldenGate Replicat for an Autonomous Database.
3. Follow the steps given in Obtain the Autonomous Database Client Credentials.
4. Log into the server where Oracle GoldenGate was installed.
5. Transfer the credentials `zip` file that you downloaded from Oracle Autonomous Database to your Oracle GoldenGate instance.
6. In the Oracle GoldenGate instance, unzip the credentials file into a new directory `/u02/data/adwc_credentials`. This is your key directory.

7. To configure the connection details, open your `tnsnames.ora` file from the Oracle client location in the Oracle GoldenGate instance.

```
cd /u02/data/adwc_credentials
ls
tnsnames.ora
```

8. Edit the `tnsnames.ora` file in the Oracle GoldenGate instance to include the connection details available in the `tnsnames.ora` file in your key directory (the directory where you unzipped the credentials `zip` file downloaded from Oracle Autonomous Database).

```
Sample Connection String
graphdb1_low = (description=
                (retry_count=20)(retry_delay=3)(address=(protocol=tcps)
(port=1522)(host=adb-preprod.us-phoenix-1.oraclecloud.com))

(connect_data=(service_name=okd2ybgcz4mjx94_graphdb1_low.adb.oraclecloud.co
m))
                (security=(ssl_server_cert_dn="CN=adwc-preprod.uscom-
east-1.oraclecloud.com,OU=Oracle BMCS US,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))
```

If Replicat becomes unresponsive due to a network timeout or lost connection, then you can add the following into the connection profile in the `tnsnames.ora` file:

```
(DESCRIPTION =  (RECV_TIMEOUT=120)   (ADDRESS_LIST =
      (LOAD_BALANCE=off)(FAILOVER=on)(CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
      (ADDRESS = (PROTOCOL = TCP)(HOST = adb-preprod.us-
phoenix-1.oraclecloud.com)(PORT = 1522))
```

> **Note:**
>
> The `tnsnames.ora` file provided with the credentials file contains three database service names identifiable as:
>
> ```
> ADWC_Database_Name_low
> ADWC_Database_Name_medium
> ADWC_Database_Name_high
> ```
>
> For Oracle GoldenGate replication, use *`ADWC_Database_Name`*`_low`.

9. To configure the wallet, create a `sqlnet.ora` file in the Oracle client location in the Oracle GoldenGate instance.

```
cd /u02/data/oci/network/admin
ls
sqlnet.ora tnsnames.ora
```

10. Edit this `sqlnet.ora` file to include your key directory.

```
WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
(DIRECTORY="/u02/data/adwc_credentials")))
SSL_SERVER_DN_MATCH=yes
```

11. Use GGSCI to log into the Oracle GoldenGate deployment.
12. Create a credential to store the `GGADMIN` user and password for the Replicat to use. For example:

```
ADD CREDENTIALSTORE ALTER CREDENTIALSTORE ADD USER
ggadmin@databasename_low PASSWORD complex_password alias adb_alias
```

13. Add and configure a Replicat to deliver to Oracle Autonomous Database. For setting up your Replicat and other processes, see Add a Replicat.

    The following example creates a Replicat (required to replicat to an Oracle Autonomous Database) called `rauto`, and instructs it to begin now.

```
ADD REPLICAT rauto, PARALLEL INTEGRATED, EXTTRAIL ./dirdat/et
```

    If you want to replicate `HR.EMP` into `COUNTRY.EMPLOYEE`, then your map statement would look like this:

```
MAP HR.EMP, TARGET COUNTRY.EMPLOYEE;
```

> **Note:**
>
> You can use classic Replicat, coordinated Replicat, and parallel Replicat in non-integrated mode. Parallel Replicat in integrated mode is also supported for Oracle Autonomous Database.

14. You can now start your Replicat and perform data replication to the Autonomous Database. Here's an example:

```
START REPLICAT rauto
```

> **Note:**
>
> Oracle Autonomous Database times out and disconnects the Replicat when it is idle for more than 60 minutes. When Replicat tries to apply changes (when it gets new changes) after being idle, it encounters a database error and abends. Oracle recommends that you configure Oracle GoldenGate with the `AUTORESTART` profile to avoid having to manually restart a Replicat when it times out.

12

Upgrade

Learn about the tasks required for upgrading Oracle GoldenGate Classic Architecture.

Upgrading Oracle GoldenGate Classic Architecture

These instructions are for upgrading to Oracle GoldenGate Classic Architecture for Oracle databases.

Overview of the Upgrade Procedure

Learn about the complete upgrade procedure for Oracle GoldenGate Classic.

Prerequisites

Before performing the upgrade procedure, read the *Release Notes for Oracle GoldenGate* to determine whether the new release affects the following in your configuration:

- New default process behavior.
- Parameters that changed or were deprecated.
- Parameters that were added to support a desired new feature or database type.
- Parameter default values that have changed.
- New data type support that might require changes to `TABLE` or `MAP` statements.
- Interaction with native database components that might require database change.

As a best practice, perform a minimal upgrade first, so that you can troubleshoot more easily in the event that any problems arise. When you verify that your environment is upgraded successfully, you can implement the new functionality.

When upgrading your database and Oracle GoldenGate simultaneously, you must upgrade the database first. However, ensure that the Oracle GoldenGate version is equal to or higher than the database version.

For Oracle Database, if you are using symbolic links that point to the Oracle GoldenGate directories, such as `dirprm` and `dirdat`, you need to use the parameter `ALLOWOUTPUTDIR` within GLOBALS.

You can prevent startup delays that can cause lag by having any required parameter changes made ahead of time, so that they are ready when you restart the processes. You should not make parameter changes while a process is running, but you can:

1. Make a copy of the parameter file.
2. Make edits to the copy.
3. After you shut down the processes during the upgrade procedure, copy the old parameter file to a new name (to save it as backup).
4. Copy the new parameter file to the old parameter file's name.

Take into account the following pre-upgrade requirements:

- Only if the trail file format is being changed, allow the Oracle GoldenGate processes to finish processing all current DML and DDL data in the Oracle GoldenGate trails.
- When upgrading your database and Oracle GoldenGate simultaneously, you must upgrade the database first.

Upgrade Considerations if Using Character-Set Conversion

Both the `TRAILCHARSET` and `SOURCEDEF` parameters are deprecated because Extract writes the source database character set with the column level. By default, these parameters are ignored but use the database character set and column character set from the table metadata. `SOURCECHARSET` parameter is only required when you need to override the source database character set and must use it with the `OVERRIDE` option.

Upgrade Considerations if Using Quoted Object Names

Oracle GoldenGate treats strings that are within single quotes as literals. Oracle GoldenGate has supported double-quoted object names since release 11.2 but did not fully implement the rule of single quotes for literals until release 12.1. Supporting double quotes for object names and single quotes for literals brings Oracle GoldenGate into compliance with SQL-92 rules and is now enabled by default. The `USEANSISQLQUOTES` parameter, which forced the SQL-92 standard in previous releases, is now deprecated.

The change to default SQL-92 rules affects object names in parameter files, input to `SQLEXEC` clauses, `OBEY` files, conversion functions, user exits, and commands. You have the following options as a result of this change:

- Retain non-SQL-92 quote rules: Oracle GoldenGate retains backward compatibility to enable the retention of current parameter files that do not conform to SQL-92 rules. To retain non-SQL-92 rules, add the `NOUSEANSISQLQUOTES` parameter to the `GLOBALS` file before you perform the upgrade and retain that parameter going forward. `NOUSEANSISQLQUOTES` affects Extract, Replicat, DEFGEN, and GGSCI.
- Upgrade your parameter files to use SQL-92 rules: Oracle GoldenGate provides the `convprm` conversion tool which you can run to convert your parameter files to be in conformance with SQL-92 rules. Run the `convprm` tool before you start the upgrade process.

Overview of the `convprm` Tool

The following describes the `convprm` tool:

- It is a command line program which can be run either manually or scripted.
- It converts string literals from double-quoted character strings to single-quoted character strings, but leaves double-quoted object names intact. It can distinguish between an object name and a string literal even when both are represented as a sequence of characters delimited with double quotes.
- It escapes quotation marks. Quotation marks must be escaped when the character that is used to delimit the string appears in the literal string itself. For example, the sentence "This character "" is a double quote" contains an escaped quote mark. The sentence 'This character '' is a single quote' contains an escaped single quote mark. When converting from double quotes to single quotes, `convprm` removes one of the repeated double quotes from escaped double quotes and escapes the single quotes that are embedded in the character sequence.

- It issues a warning message if `NOUSEANSISQLQUOTES` is specified in the `GLOBALS` file. The message states that the converted parameter file is incompatible with `NOUSEANSISQLQUOTES`, but the parameter file was updated anyway.
- It recursively converts the files that are included through an `OBEY` or `INCLUDE` parameter.
- It creates a backup of the initial parameter file in the same directory as the original file. The backup has the name of the original file with the `.bck` suffix. The creation of a backup file can be disabled when you run the `convprm` tool.
- It converts the character set. The character set for the new parameter file is taken from the `CHARSET` parameter in the original parameter file. Absent that parameter, the character set is taken from the `CHARSET` parameter in the `GLOBALS` file. Absent a `GLOBALS` parameter, the new parameter file is written in the character set of the local operating system.

#unique_1134/unique_1134_Connect_42_CACFBJGF provides examples of the conversion outcome.

Table 12-1 Comparison of Input Requirements for [NO]USEANSISQLQUOTES

Input variable	String literal with old syntax	String literal with new syntax
Double quotes are escaped in the old syntax but not in the new syntax.	"abc""def"	'abc"def'
Single quotes are escaped in the new syntax but not in the old syntax.	"abc'def"	'abc''def'

Running convprm

To use the `convprm` tool:

1. Run `convprm` with the following syntax:

   ```
   convprm [options] input_files
   ```

   where:

   - `{-h | --help}` displays usage.
   - `{-v |--version}` displays version information.
   - `{-i | --follow-include}` recursively converts files included through an `OBEY` or `INCLUDE` parameter.
   - `{-n |--no-backup}` does not create a copy of the original file.
   - `{-s | --silent}` does not display status messages.
   - `{-q | --quotes}` performs quote conversion. This is the default.
   - `{-d | --dry-run}` does not change the parameter file or create a backup file. It only prints out what would happen as the result of the conversion.
   - *`input_files`* is a list of the parameter files that are to be converted. Separate each file name with a white space, for example:

     ```
     convprm [options] extfin extacct exthr
     ```

2. Examine the parameter file to make certain the conversion completed successfully. Status messages are displayed at the beginning, during, or at the end of the file conversion process.

On errors, the process abends in the same way as other Oracle GoldenGate executables. All error messages that cause the converter to fail are sent to the Oracle GoldenGate error log.

If you are currently using the `USEANSISQLQUOTES` parameter, you may remove it or leave it in the parameter files. It is now the default.

Obtaining the Oracle GoldenGate Distribution

To obtain Oracle GoldenGate, follow these steps:

1. Go to edelivery: edelivery.oracle.com

   Also see MOS note 1645495.1 and 2193391.1 for more information.

   To access Oracle Technology Network, go to https://www.oracle.com/middleware/technologies/goldengate.html
2. Find the Oracle GoldenGate 21c release and download the ZIP file onto your system.

For more information about locating and downloading Oracle Fusion Middleware products, see the Oracle Fusion Middleware Download, Installation, and Configuration Readme Files on Oracle Technology Network.

Upgrading Oracle GoldenGate Classic Architecture for Oracle Database

These instructions contain the procedure for performing the minimal upgrade.

> **Note:**
>
> Trigger-based DDL capture has been desupported from 21c onward, so you need to upgrade to native DDL capture.

> **Note:**
>
> If you are using integrated capture and plan to upgrade from trigger-based DDL capture to new native DDL capture, **do not** remove the DDL trigger until prompted. Extract needs to mine DDL to the point where the redo `COMPATIBLE` level. For example, if Extract is behind by a week when the database is upgraded, Extract does not immediately switch to native DDL capture. It must be allowed to process the previous redo first, then Extract upgrades to native DDL capture automatically.

1. Use the following command in GGSCI to determine the oldest archive log that you might need to restore when Extract starts. The `Recovery Checkpoint` field shows the oldest log needed for recovery.

   ```
   GGSCI> INFO EXTRACT group, SHOWCH
   ```

   It's best to perform upgrade activities outside of the peak hours. If there are large and long running transactions, you may consider that on the source system, the new Extract might need to start processing from the normal recovery checkpoint, rather than the bounded recovery checkpoint, if the first record of the oldest open transaction at the time that you stop Extract is in a log that is not on the system.

   You have two options:

- You can restore the archives back to, and including, the one shown in the recovery checkpoint shown with `INFO EXTRACT`.
- You can clear out the long-running transactions that apply to the Extract that you are upgrading. This can be done by skipping the transactions or by forcing them to the trail as committed transactions. Skipping a transaction may cause data loss, and forcing a transaction to the trail may add unwanted data to the trail if the transaction is rolled back. To skip or force a transaction, follow these steps:

    a. View open transactions with the following command in GGSCI. Record the transaction ID of any transaction that you want to clean up.

    ```
    GGSCI> SEND EXTRACT group, SHOWTRANS
    ```

    b. Clean up old transactions with the `SEND EXTRACT` command, using either the `SKIPTRANS` option to skip a transaction or the `FORCETRANS` option to force a transaction in its current state to the trail as a committed transaction.

    ```
    GGSCI> SEND EXTRACT group, {SKIPTRANS | FORCETRANS} transaction_ID
    [THREAD n] [FORCE]
    ```

    c. After you are finished cleaning up the long-running transactions, force a Bounded Recovery checkpoint.

    ```
    GGSCI> SEND EXTRACT group, BR BRCHECKPOINT IMMEDIATE
    ```

    > **Note:**
    >
    > A forced checkpoint is necessary because the skipped or forced transaction is not cleaned up from the Bounded Recovery checkpoint and will be shown if `SHOWTRANS` is issued again. This is a known issue. For more information about `SEND EXTRACT`, see *Reference for Oracle GoldenGate*.

2. (Target systems) In GGSCI, stop all Replicat processes.

    ```
    GGSCI> STOP REPLICAT group
    ```

3. (Source and target systems) In GGSCI, stop Manager on the source and target systems.

    ```
    GGSCI> STOP MANAGER
    ```

4. When updating target systems only, or if updating the target side before the source side, you *must* use `STOP` to stop all data pumps and any primary Extracts that write directly to those targets on any source running on this target. Any static collectors that may have been started that must be stopped, as well. To verify that there are no `server` processes running, use process checking shell commands, such as `ps` and `grep`.
5. You need to use an out-of-place upgrade, which implies that you retain the existing installation in parallel while you run the upgrade. See Install and Patch for details.

6. In GGSCI, start the Oracle GoldenGate processes on the source and target systems in the following order.

   ```
   GGSCI> START MANAGER
   GGSCI> START EXTRACT group
   GGSCI> START EXTRACT pump
   GGSCI> START REPLICAT group
   ```

   If you need to restore any log files, Extract abends with an error that indicates the log to restore. Restore the logs back to, and including that log, and then restart Extract.

If you made copies of the parameter files to make parameter changes, move the new parameter files into the Oracle GoldenGate directory where the old parameter files were stored then rename them to the same names as the old parameter files. If you are using case-sensitivity support, ensure that you either add `NOUSEANSISQLQUOTES` to your parameter files, or that you ran the `convprm` utility to convert the quotes as required. See "Upgrade Considerations if Using Character-Set Conversion" for more information.

Upgrade Considerations for Older Oracle GoldenGate Releases

- To accomodate the changes in the checkpoint table and heartbeat table, it is recommended that you upgrade the heartbeat table (source and target) and the checkpoint table (target only). If there is no change related to the objects, then the command returns with an informational message only. This step updates the table definition to add columns that support the Oracle GoldenGate 18c (18.1.0) release.

  ```
  GGSCI> DBLOGIN USERIDALIAS [alias] |
  GGSCI> UPGRADE CHECKPOINTTABLE [owner.table]
  ```

  In case of SQL Server and MySQL, you need to specifyc the `SOURCEDB` data source with `DBLOGIN`. See DBLOGIN command for details.

  `owner.table` can be omitted if the checkpoint table was created with the name listed with `CHECKPOINTTABLE` in the GLOBALS file. If the checkpoint table is already upgraded, then this command doesn't perform any further updates.

Upgrading Oracle GoldenGate from OUI

You can use Oracle Universal Installer (OUI) on any of the Linux, UNIX, and Windows platforms that OUI supports and which Oracle GoldenGate supports. OUI is supported for Oracle versions 11g and later. An instance of Oracle GoldenGate can be installed for only one Oracle version in any given Oracle home. You can install multiple instances of Oracle GoldenGate for the same or different database versions on the same host.

The installer registers the Oracle GoldenGate home directory with the central inventory that is associated with the selected database. The inventory stores information about all Oracle software products installed on a host, provided the product was installed using OUI.

To perform the upgrade using OUI, perform the following steps:

1. Unzip and untar the installation file.
2. From the unzipped directory, run the **runInstaller** program on UNIX or Linux, or `run setup.exe` on Windows.

3. On the **Select Installation Option** page, select the Oracle GoldenGate build to install, and then click **Next** to continue.
4. On the **Specify Installation Details** page, specify the following:
   - For **Software Location**, specify the Oracle GoldenGate installation directory. It can be a new or existing directory. The default location is under installing user's home directory, but Oracle recommends changing it to a local directory that is not mounted and has no quotas. The specified directory cannot be a registered home in the Oracle central inventory. If installing in a cluster, install Oracle GoldenGate on shared storage that is accessible by all of the cluster nodes.
   - (Optional) Select **Start Manager** to perform configuration functions, such as creating the Oracle GoldenGate sub-directories in the installation folder, setting library paths, and starting Manager on the specified port number. To proceed, a database must exist on the system. When Start Manager is selected, the **Database Location and Manager Port** fields are displayed.
     - The database must have a registered home in the Oracle central inventory. The installer registers the Oracle GoldenGate home directory with the central inventory.
     - For Manager Port, accept the default port number or enter a different unreserved, unrestricted port number for the Manager process to use for inter-process communication. The default port is the first available one starting with 7809. If you are installing multiple instances of Oracle GoldenGate on the same system, each must use a different port number.
5. Click **Next** to continue. In case of upgrading existing Oracle GoldenGate Installation, OUI prompts that the selected Software location has files or directories. Click on **Yes**.
6. The **Create Inventory** page is displayed if this is the first Oracle product to be installed from OUI on a host that does not have a central inventory.
   - For **Inventory Directory**, specify a directory for the central inventory. It can be a new directory or an existing directory that is empty and has the amount of disk space shown on the screen. The directory cannot be on a shared drive.
   - Select an operating system group in which the members have write permission to the inventory directory. This group is used to add inventory information to the Oracle GoldenGate sub-folder.
7. On the **Summary** page, confirm that there is enough space for the installation and that the installation selections are correct. Optionally, click **Save Response File** to save the installation information to a response file. You can run the installer from the command line with this file as input to duplicate the results of a successful installation on other systems. You can edit this file or create a new one from a template.
8. Click Install to begin the installation or **Back** to go back and change any input specifications. When Upgrading existing Oracle GoldenGate Installation, OUI will notify that the software location has files or directories. Click **Yes** to continue. You are notified when the installation is finished.
9. If you created a central inventory directory, you are prompted to run the `INVENTORY_LOCATION/orainstRoot.sh` script. This script must be executed as the root operating system user. This script establishes the inventory data and creates sub-directories for each installed Oracle product (in this case, Oracle GoldenGate).

Upgrading Oracle GoldenGate using OUI – Silent

These instructions apply to new installations, as well as upgrades.

You can perform a silent installation from the command console if the system has no X-Windows interface or to perform an automated installation. Silent installations can ensure that multiple users in your organization use the same installation options when they install your Oracle products.

You perform a silent installation by running a response file. You can create a response file by selecting the **Save Response File** option during an interactive OUI session or by editing a template.

1. To run a response file, use the following command:

   ```
   runInstaller -silent -nowait -responseFile absolute_path_to_response_file
   ```

   The response files and the template are stored in the response subdirectory of the Oracle GoldenGate installation directory. The Oracle GoldenGate response file contains a standard set of Oracle configuration parameters in addition to parameters that are specific to Oracle GoldenGate. These parameters correspond to the fields in the interactive session.

   > **Note:**
   >
   > If you are upgrading an existing Oracle GoldenGate installation with the silent option, then you might get the following warning:
   >
   > ```
   > WARNING:OUI-10030:You have specified a non-empty directory to install this
   > product. It is recommended to specify either an empty or a non-existent
   > directory.
   > You may, however, choose to ignore this message if the directory contains
   > Operating System generated files or subdirectories like lost+found. Do you
   > want to proceed with installation in this Oracle Home?
   > ```

2. Press **ENTER** to continue.

Upgrading a Configuration That Includes DDL Support

This section contains considerations and steps you should take when DDL support is active in the Oracle GoldenGate environment.

In the Integrated mode, if the source database is Oracle 11.2.0.4 or later, DDL capture support is integrated into the database logmining server as long as the database `COMPATIBLE` parameter is set to 11.2.0.4 or higher.

Table 12-2 shows possible DDL upgrade paths and guidelines.

Table 12-2 Possible Upgrade Paths to Oracle GoldenGate and Requirements for DDL Support

Upgrade from:	To: Integrated capture, no trigger
Integrated capture using trigger method (all 11.2.1 database versions)	Can be used for a container database. 1. Source database must be 11.2.0.4 or higher. 2. Source database `COMPATIBLE` setting must be 11.2.0.4 or higher. 3. Upgrade Oracle GoldenGate per these upgrade instructions.

[1]

Upgrading Configuration that includes Berkeley Database - Oracle GoldenGate 12.2 or later

When you are upgrading Oracle GoldenGate from release 12.1.2.1 to 12.3.0.1 and have enabled monitoring and the datastore is created by the Performance Metrics server, the best practice is to purge the data store before performing the upgrade. After the upgrade, the datastore is recreated. For more information about purging a datastore, see How to Purge the Datastore. in the *Using the Oracle GoldenGate Microservices Architecture*.

From Oracle GoldenGate 12.3.0.1 onward, all operations related to the datastore have been removed and are taken care of by the Performance Metrics server. To know more, see Monitoring Performance.

Upgrading Oracle GoldenGate for Non-Oracle Databases

Learn how to upgrade Oracle GoldenGate for Non-Oracle databases.

Oracle GoldenGate Upgrade Considerations

Before you start the upgrade, review the information about upgrading Extract and Replicat.

Even though you may only be upgrading the source or target installations, rather than both, all processes are involved in the upgrade. All processes must be stopped in the correct order for the upgrade, regardless of which component you upgrade, and the trails must be processed until empty.

Oracle recommends that you begin your upgrade with the target rather than the source to avoid the necessity of adjusting the trail file format.

Installation Binaries and Deployments

With Microservice Architecture, there is a strong separation between where the software is installed and the deployment directory structure for the Oracle GoldenGate instance, which contains the parameter files, report files, and trail files. For both these areas, the software binaries and deployment, are strictly separated. So, there is no interference between the old and new software installations related to the deployments. During a software upgrade, the new software will be installed independently. The deployment working with the old software will be stopped. Then, the deployment environment will be adjusted to the new software and the deployment will be restarted.

If you have a reverse proxy configuration on your host machine generated with `OGG_HOME/lib/utl/reverseproxy/ReverseProxySettings`, then consider reconfiguring it to leverage the enhanced `ReverseProxySetting` utility available with Oracle GoldenGate 21c (21.3) and higher releases.

Considerations for Upgrading Service Manager and other Deployments

When upgrading Oracle GoldenGate, the Service Manager must be updated first. The software version of the Service Manager must be higher or equal to the version of the deployments. There are no issues having a Service Manager running on the highest version and having deployments with lower versions.

---

[1] An upgrade of the database to 11.2.0.4 or 12.1 automatically takes a data dictionary snapshot in the redo stream as part of the patch set upgrade.

After completing the upgrade, run the `UPGRADE HEARTBEATTABLE` command to add extra columns for tables and lag views. These extra columns are used to track the Extract restart position. See `UPGRADE HEARTBEATTABLE` to know more.

Extract Upgrade Considerations

Running Extract in classic mode (non-integrated Extract) with the Oracle database has been desupported. Before upgrading Extracts running in classic mode (non-integrated Extract) with the Oracle database, you need to upgrade the Extracts to run in integrated mode.

Upgrading Classic Extract to Integrated Extract for Oracle

As classic Extract is desupported, you must upgrade the classic Extract to an integrated Extract. Stop the classic Extract, which were configured with DDL capture, and remove the DDL triggers prior to upgrading to an integrated Extract.

Remove the DDL triggers:

- `ddl_disable`
- `ddl_remove`
- `marker_remove`

To upgrade to Extract, use the following command:

```
ALTER EXTRACT group_name to UPGRADE INTEGRATED TRANLOG
```

Also, see Switching Extract from Classic Mode to Integrated Mode in *Administering Oracle GoldenGate*.

Replicat Upgrade Considerations

All Replicat installations should be upgraded at the same time. It is critical to ensure that all trails leading to all Replicat groups on all target systems are processed until empty, according to the upgrade instructions.

When upgrading from releases prior to 19c release of Oracle GoldenGate, ensure that you do not use the `SOURCEDEF` parameter in Replicat, otherwise the Replicat will abend. However, if the trail file format is pre-12.2, then `SOURCEDEF` is still required because no metadata exists in the trail file.

Because the `TIMEZONE` datatype is managed differently with Oracle GoldenGate 21c, you may need to run the `ALTER REPLICAT` *`extseqno`* command to synchronize with newer trail files after consuming the old trail file written by the Extract.

Upgrading Oracle GoldenGate for Non-Oracle Databases

These instructions are for upgrading Oracle GoldenGate Classic Architecture in the supported Non-Oracle database environments.

Overview of the Upgrade Procedure for Non-Oracle Databases

The upgrade performs a minimal feature upgrade to deploy only the core Oracle GoldenGate functionality, without implementing any of the major new features. It ensures easy troubleshooting of any upgrade related issues that may occur after the upgrade. After upgrading the Oracle GoldenGate environment successfully, you can implement the new functionality.

If you are upgrading multiple Extract processes that operate in a consolidated configuration (many sources to one target), you must upgrade one Extract at a time. All Replicat installations must be upgraded at the same time. It is critical to ensure that all trails leading to all Replicat groups on all target systems are processed until empty.

> **Note:**
>
> The hash calculation used by the `@RANGE` function to partition data among Replicat processes has been changed. This change is transparent, and no re-partitioning of rows in the parameter files is required. To ensure data continuity, ensure that you allow all Replicat processes on all systems to finish processing all the data in their trails before stopping those processes. If the Replicat processes are not upgraded all at the same time, or the trails are not cleaned out prior to the upgrade, rows may shift partitions as a result of the new hash method, which may result in collision errors.

Obtaining the Oracle GoldenGate Distribution

To obtain Oracle GoldenGate, follow these steps:

1. Go to edelivery: edelivery.oracle.com

   To access Oracle Technology Network, go to https://www.oracle.com/middleware/technologies/goldengate.html
2. Find the Oracle GoldenGate 19c (19.1.0) release and download the ZIP file onto your system.

For more information about locating and downloading Oracle Fusion Middleware products, see the Oracle Fusion Middleware Download, Installation, and Configuration Readme Files on Oracle Technology Network.

Upgrading Oracle GoldenGate Classic Architecture for Non-Oracle Databases

Even though you may only be upgrading Extract or Replicat, rather than both, all processes are involved in the upgrade. All processes must be stopped in the correct order for the upgrade, regardless of which component you upgrade, and the trails must be processed until empty.

1. (Source and target systems) Back up the current Oracle GoldenGate installation directory on the source and target systems, and any working directories that you have installed on a shared drive in a cluster (if applicable).
2. (Source and target systems, as applicable) Expand the Oracle GoldenGate upgrade build into a new directory on each system (not the current Oracle GoldenGate directory). Do not create the sub-directories; just complete the steps to the point where the installation files are expanded.

   However, this step doesn't apply to PostgreSQL.

   Oracle GoldenGate for PostgreSQL upgrade only works if you install the latest version (version to be upgraded) in the same `$OGG_HOME` directory as the current Oracle GoldenGate version.

For PostgreSQL, Oracle GoldenGate upgrade doesn't work if you install the latest version (version to be upgraded) in a different `$OGG_HOME` directory (21.3) and repoint the new `$OGG_HOME` to the latest version.

For PostgreSQL, use the same `$OGG_HOME` as current version directory for the latest Oracle GoldenGate binary. Make sure to take backup of existing `$OGG_HOME` before beginning the upgrade.

3. Stop all user activity that generates DML and DDL on objects in your Oracle GoldenGate configuration and ensure that there are no outstanding open transactions against the database.

   For SQL Server CDC Extract on a Source system, manually stop the CDC Capture job for the database.

4. (Source system) In GGSCI on the source system, issue the `SEND EXTRACT` command with the `LOGEND` option until it shows `YES`, indicating that there is no more data in the transaction log to process.

   For SQL Server CDC Extract on Source system, monitor that the current read position of the Extract is no longer updating, by repeatedly running `SEND EXTRACT` group `STATUS` for a few seconds, and observe that the LSN value for the current read position is no longer updating.

   ```
   GGSCI> SEND EXTRACT group LOGEND
   ```

5. (Source system) In GGSCI, stop Extract and data pumps.

   ```
   GGSCI> STOP EXTRACT group
   ```

6. (Target systems) In GGSCI on each target system, issue the `SEND REPLICAT` command with the `STATUS` option until it shows a status of "`At EOF`" to indicate that it finished processing all of the data in the trail. This must be done on all target systems until all Replicat processes return "`At EOF`."

   ```
   GGSCI> SEND REPLICAT group STATUS
   ```

7. (Target systems) In GGSCI, stop all Replicat processes.

   ```
   GGSCI> STOP REPLICAT group
   ```

8. (Source and target systems) In GGSCI, stop Manager on the source and target systems and close GGSCI.

   ```
   GGSCI> STOP MANAGER
   ```

9. If you want to upgrade the source or target database, or both, do so at this time according to the upgrade instructions provided for that database. Ensure that you start the databases after the upgrade, but do not permit transactions on the objects in the Oracle GoldenGate configuration.

   For MySQL, if you upgrade from Oracle GoldenGate 19c (19.1.0) and the database is MySQL 5.7, then no change is required. However, if you upgrade from Oracle GoldenGate 19c (19.1.0) and the database is MySQL 8.0, then you need to perform the following steps:

   a. Enable logging of full metadata because it's mandatory for MySQL 8.0 and higher, regardless of DDL or DML replication. Logging of full metadata can be enabled by setting the value of MySQL server variable `binlog_row_metadata` to `FULL` inside the MySQL configuration file (`my.cnf` for Linux and `my.ini` for Windows). You need to restart the database service after changing the configuration file for the settings to take effect.

   b. Run the DDL uninstall scripts to disable old DDL solutions if DDL replication was enabled previously.

   The script name:

   For Windows: `ddl_install.bat`

   For Linux: `ddl_install.sh`

   c. To uninstall, run the following script:

   ```
   bash$ ./ddl_install.sh uninstall mysql userid password port
   ```

10. (Source and target systems) Move the expanded Oracle GoldenGate files from the new directory to your existing Oracle GoldenGate directory on the source and target systems overwriting the existing files.
11. (DB2 for i) Run `ggos400install` without arguments. For an upgrade, no arguments are necessary. However, if you change the library, the old library is left on the system until you remove it.
12. (Source and target systems) Start GGSCI.
13. (Target systems, if upgrading Replicat from version 11.2.1.0.0 or earlier only) In GGSCI on each target system, issue the following commands to upgrade the Replicat checkpoint tables on those systems. This step updates the table definition.

    ```
    GGSCI> DBLOGIN {
            [SOURCEDB data_source] |
            [, database@host:port] |USERID {/ | userid}
            [, PASSWORD password]
            [algorithm ENCRYPTKEY {keyname | DEFAULT}] |USERIDALIAS alias
    [DOMAIN domain] |
            [SYSDBA | SQLID sqlid]
            [SESSIONCHARSET character_set]}


    GGSCI> UPGRADE CHECKPOINTTABLE [owner.table]
    ```

    > **Note:**
    >
    > *`owner.table`* can be omitted if the checkpoint table was created with the name listed with `CHECKPOINTTABLE` in the `GLOBALS` file.

14. (SQL Server CDC Extract on Source system) Run `ADD TRANDATA` again on any tables previously enabled with it, including any filter table and the `gg_heartbeat` and `gg_heartbeat_seed` tables if using the Oracle GoldenGate heartbeat implementation.

    ```
    GGSCI> DBLOGIN {[SOURCEDB data_source] | |USERID {/ | userid}[, PASSWORD
    password] |USERIDALIAS alias [DOMAIN domain]
    GGSCI> ADD TRANDATA schema.tablename
    ```

15. (Target system) If upgrading the target Oracle GoldenGate installation that is the recipient of trails from a source system running Oracle GoldenGate prior to version 11.2.1, then add the `SOURCECHARSET` parameter to the Replicat and specify the character set of the source database.
16. (Source system) By default, after upgrading, the Extract will continue to write trail files in the version of Oracle GoldenGate prior to the upgrade. To force the Extract to write in the upgraded trail version, use the `FORMAT RELEASE` parameter in the Extract, specifying the

new version, or alternately, perform an `ETROLLOVER` of the Extract and manually reposition the downstream processes to start reading at the new trail sequence.

```
{EXTTRAIL | RMTTRAIL} file_name FORMAT RELEASE major.minor
```

17. If you made copies of the parameter files to make parameter changes, move the new parameter files into the Oracle GoldenGate directory where the old parameter files were stored, and give them the same names as the old parameter files. If using case-sensitivity support, make certain that you either added `NOUSEANSISQLQUOTES` to your parameter files, or that you ran the `convprm` utility to convert the quotes as required.
18. Upgrade the heartbeat table configuration if it was previously implemented, before restarting all the processes.

```
GGSCI> DBLOGIN {[SOURCEDB data_source] | |USERID {/ | userid}[, PASSWORD
password]
|USERIDALIAS alias [DOMAIN domain]
GGSCI> UPGRADE HEARTBEATTABLE
```

19. You also need to modify the `BATCHSQL` parameter to double the value of `BATCHESPERQUEUE`. You must do this before starting Replicat.

> **Note:**
>
> If you are upgrading from Oracle GoldenGate version 12.1 to any later version and using the `INSERTALLRECORDS` parameter, it is recommended that you should double the value of `BYTESPEERQUEUE` option of the `BATCHSQL` parameter. For example, if you are using the `BYTESPEERQUEUE` option with its default value, which is 20 MB, then increase the value to 40 MB. However, if you are not using the default value for the `BYTESPEERQUEUE` option, then double the value specified during the Oracle GoldenGate version 12.1 installation.

For example:

```
BATCHSQL BATCHESPERQUEUE 40000000
```

20. For SQL Server CDC Extract on a Source system, manually restart the CDC Capture job for the database.
21. In GGSCI, start the Oracle GoldenGate processes on the source and target systems in the following order.

```
GGSCI> START MANAGER
GGSCI> START EXTRACT group
GGSCI> START EXTRACT pump
GGSCI> START REPLICAT group
```

Performing Application Patches

Application patches and application upgrades typically perform DDL such as adding new objects or changing existing objects. To apply applications patches or upgrades in an Oracle GoldenGate environment, you can do one of the following:

- If Oracle GoldenGate supports DDL replication for your database type, you can use it to replicate the DDL without stopping replication processes. To use this method, the source and target table structures must be identical.
- You can apply the patch or upgrade manually on both source and target after taking the appropriate steps to ensure replication continuity.

To Use Oracle GoldenGate to Replicate Patch DDL

1. If you have not already done so, dedicate some time to learn, install, and configure the Oracle GoldenGate DDL support. See the instructions for your database in this documentation. Once the DDL environment is in place, future patches and upgrades will be easier to apply.
2. If the application patch or upgrade adds new objects that you want to include in data replication, make certain that you include them in the `DDL` parameter statement. To add new objects to your `TABLE` and `MAP` statements, see the procedure on Adding Tables to the Oracle GoldenGate Configuration.
3. If the application patch or upgrade installs triggers or cascade constraints, disable those objects on the target to prevent collisions between DML that they execute on the target and the same DDL that is replicated from the source trigger or cascaded operation.

To Apply a Patch Manually on the Source and Target

1. Stop access to the source database.
2. Allow Extract to finish capturing the transaction data that remains in the transaction log. To determine when Extract is finished, issue the following command in GGSCI until it returns `At EOF`.

   ```
   SEND EXTRACT group GETLAG
   ```

3. Stop Extract.

   ```
   STOP EXTRACT group
   ```

4. Start applying the patch on the source.
5. Wait until the data pump (if used) and Replicat are finished processing the data in their respective trails. To determine when they are finished, use the following commands until they return `At EOF`.

   ```
   SEND EXTRACT group GETLAG
   SEND REPLICAT group GETLAG
   ```

6. Stop the data pump and Replicat.

   ```
   STOP EXTRACT group
   STOP REPLICAT group
   ```

   At this point, the data in the source and target should be identical, because all of the replicated transactional changes from the source have been applied to the target.

7. Apply the patch on the target.
8. If the patches changed table definitions, run DEFGEN for the source tables to generate updated source definitions, and then replace the old definitions with the new ones in the existing source definitions file on the target system.
9. Start the Oracle GoldenGate processes whenever you are ready to begin capturing user activity again.

13

Appendix

Learn about additional details required for supporting Oracle GoldenGate on different databases.

Supported Character Sets

This appendix lists the character sets that Oracle GoldenGate supports when converting data from source to target.

The identifiers that are shown should be used for Oracle GoldenGate parameters or commands when a character set must be specified, instead of the actual character set name. Currently Oracle GoldenGate does not provide a facility to specify the database-specific character set.

Supported Character Sets - Oracle

Table 13-1 Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`al32utf8`	Unicode 9.0 Universal Character Set (UCS), UTF-8 encoding scheme
`ar8ados710t`	Arabic MS-DOS 710 8-bit Latin/Arabic
`ar8ados710`	Arabic MS-DOS 710 Server 8-bit Latin/Arabic
`ar8ados720t`	Arabic MS-DOS 720 8-bit Latin/Arabic
`ar8ados720`	Arabic MS-DOS 720 Server 8-bit Latin/Arabic
`ar8aptec715t`	APTEC 715 8-bit Latin/Arabic
`ar8aptec715`	APTEC 715 Server 8-bit Latin/Arabic
`ar8arabicmacs`	Mac Server 8-bit Latin/Arabic
`ar8arabicmact`	Mac 8-bit Latin/Arabic
`ar8arabicmac`	Mac Client 8-bit Latin/Arabic
`ar8asmo708plus`	ASMO 708 Plus 8-bit Latin/Arabic
`ar8asmo8x`	ASMO Extended 708 8-bit Latin/Arabic
`ar8ebcdic420s`	EBCDIC Code Page 420 Server 8-bit Latin/Arabic
`ar8ebcdicx`	EBCDIC XBASIC Server 8-bit Latin/Arabic
`ar8hparabic8t`	HP 8-bit Latin/Arabic
`ar8iso8859p6`	ISO 8859-6 Latin/Arabic
`ar8mswin1256`	MS Windows Code Page 1256 8-Bit Latin/Arabic
`ar8mussad768t`	Mussa'd Alarabi/2 768 8-bit Latin/Arabic

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`ar8mussad768`	Mussa'd Alarabi/2 768 Server 8-bit Latin/Arabic
`ar8nafitha711t`	Nafitha International 711 Server 8-bit Latin/Arabic
`ar8nafitha711`	Nafitha Enhanced 711 Server 8-bit Latin/Arabic
`ar8nafitha721t`	Nafitha International 721 8-bit Latin/Arabic
`ar8nafitha721`	Nafitha International 721 Server 8-bit Latin/Arabic
`ar8sakhr706`	SAKHR 706 Server 8-bit Latin/Arabic
`ar8sakhr707t`	SAKHR 707 8-bit Latin/Arabic
`ar8sakhr707`	SAKHR 707 Server 8-bit Latin/Arabic
`ar8xbasic`	XBASIC 8-bit Latin/Arabic
`az8iso8859p9e`	ISO 8859-9 Azerbaijani
`bg8mswin`	MS Windows 8-bit Bulgarian Cyrillic
`bg8pc437s`	IBM-PC Code Page 437 8-bit (Bulgarian Modification)
`blt8cp921`	Latvian Standard LVS8-92(1) Windows/Unix 8-bit Baltic
`blt8ebcdic1112s`	EBCDIC Code Page 1112 8-bit Server Baltic Multilingual
`blt8ebcdic1112`	EBCDIC Code Page 1112 8-bit Baltic Multilingual
`blt8iso8859p13`	ISO 8859-13 Baltic
`blt8mswin1257`	MS Windows Code Page 1257 8-bit Baltic
`blt8pc775`	IBM-PC Code Page 775 8-bit Baltic
`bn8bscii`	Bangladesh National Code 8-bit BSCII
`cdn8pc863`	IBM-PC Code Page 863 8-bit Canadian French
`ce8bs2000`	Siemens EBCDIC.DF.04-2 8-bit Central European
`cel8iso8859p14`	ISO 8859-13 Celtic
`ch7dec`	DEC VT100 7-bit Swiss (German/French)
`cl8bs2000`	Siemens EBCDIC.EHC.LC 8-bit Latin/Cyrillic-1
`cl8ebcdic1025c`	EBCDIC Code Page 1025 Client 8-bit Cyrillic
`cl8ebcdic1025r`	EBCDIC Code Page 1025 Server 8-bit Cyrillic
`cl8ebcdic1025s`	EBCDIC Code Page 1025 Server 8-bit Cyrillic
`cl8ebcdic1025`	EBCDIC Code Page 1025 8-bit Cyrillic
`cl8ebcdic1025x`	EBCDIC Code Page 1025 (Modified) 8-bit Cyrillic
`cl8ebcdic1158r`	EBCDIC Code Page 1158 Server 8-bit Cyrillic
`cl8ebcdic1158`	EBCDIC Code Page 1158 8-bit Cyrillic
`cl8iso8859p5`	ISO 8859-5 Latin/Cyrillic
`cl8isoir111`	SOIR111 Cyrillic
`cl8koi8r`	RELCOM Internet Standard 8-bit Latin/Cyrillic

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`cl8koi8u`	KOI8 Ukrainian Cyrillic
`cl8maccyrillics`	Mac Server 8-bit Latin/Cyrillic
`cl8maccyrillic`	Mac Client 8-bit Latin/Cyrillic
`cl8mswin1251`	MS Windows Code Page 1251 8-bit Latin/Cyrillic
`d7dec`	DEC VT100 7-bit German
`d7siemens9780x`	Siemens 97801/97808 7-bit German
`d8bs2000`	Siemens 9750-62 EBCDIC 8-bit German
`d8ebcdic1141`	EBCDIC Code Page 1141 8-bit Austrian German
`d8ebcdic273`	EBCDIC Code Page 273/1 8-bit Austrian German
`dk7siemens9780x`	Siemens 97801/97808 7-bit Danish
`dk8bs2000`	Siemens 9750-62 EBCDIC 8-bit Danish
`dk8ebcdic1142`	EBCDIC Code Page 1142 8-bit Danish
`dk8ebcdic277`	EBCDIC Code Page 277/1 8-bit Danish
`e7dec`	DEC VT100 7-bit Spanish
`e7siemens9780x`	Siemens 97801/97808 7-bit Spanish
`e8bs2000`	Siemens 9750-62 EBCDIC 8-bit Spanish
`ee8bs2000`	Siemens EBCDIC.EHC.L2 8-bit East European
`ee8ebcdic870c`	EBCDIC Code Page 870 Client 8-bit East European
`ee8ebcdic870s`	EBCDIC Code Page 870 Server 8-bit East European
`ee8ebcdic870`	EBCDIC Code Page 870 8-bit East European
`ee8iso8859p2`	ISO 8859-2 East European
`ee8macces`	Mac Server 8-bit Central European
`ee8macce`	Mac Client 8-bit Central European
`ee8maccroatians`	Mac Server 8-bit Croatian
`ee8maccroatian`	Mac Client 8-bit Croatian
`ee8mswin1250`	MS Windows Code Page 1250 8-bit East European
`ee8pc852`	IBM-PC Code Page 852 8-bit East European
`eec8euroasci`	EEC Targon 35 ASCI West European/Greek
`eec8europa3`	EEC EUROPA3 8-bit West European/Greek
`el8dec`	DEC 8-bit Latin/Greek
`el8ebcdic423r`	IBM EBCDIC Code Page 423 for RDBMS server-side
`el8ebcdic875r`	EBCDIC Code Page 875 Server 8-bit Greek
`el8ebcdic875s`	EBCDIC Code Page 875 Server 8-bit Greek
`el8ebcdic875`	EBCDIC Code Page 875 8-bit Greek

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`el8gcos7`	Bull EBCDIC GCOS7 8-bit Greek
`el8iso8859p7`	ISO 8859-7 Latin/Greek
`el8macgreeks`	Mac Server 8-bit Greek
`el8macgreek`	Mac Client 8-bit Greek
`el8mswin1253`	MS Windows Code Page 1253 8-bit Latin/Greek
`el8pc437s`	IBM-PC Code Page 437 8-bit (Greek modification)
`el8pc737`	IBM-PC Code Page 737 8-bit Greek/Latin
`el8pc851`	IBM-PC Code Page 851 8-bit Greek/Latin
`el8pc869`	IBM-PC Code Page 869 8-bit Greek/Latin
`et8mswin923`	MS Windows Code Page 923 8-bit Estonian
`f7dec`	DEC VT100 7-bit French
`f7siemens9780x`	Siemens 97801/97808 7-bit French
`f8bs2000`	Siemens 9750-62 EBCDIC 8-bit French
`f8ebcdic1147`	EBCDIC Code Page 1147 8-bit French
`f8ebcdic297`	EBCDIC Code Page 297 8-bit French
`hu8abmod`	Hungarian 8-bit Special AB Mod
`hu8cwi2`	Hungarian 8-bit CWI-2
`i7dec`	DEC VT100 7-bit Italian
`i7siemens9780x`	Siemens 97801/97808 7-bit Italian
`i8ebcdic1144`	EBCDIC Code Page 1144 8-bit Italian
`i8ebcdic280`	EBCDIC Code Page 280/1 8-bit Italian
`in8iscii`	Multiple-Script Indian Standard 8-bit Latin/Indian
`is8macicelandics`	Mac Server 8-bit Icelandic
`is8macicelandic`	Mac Client 8-bit Icelandic
`is8pc861`	IBM-PC Code Page 861 8-bit Icelandic
`iw7is960`	Israeli Standard 960 7-bit Latin/Hebrew
`iw8ebcdic1086`	EBCDIC Code Page 1086 8-bit Hebrew
`iw8ebcdic424s`	EBCDIC Code Page 424 Server 8-bit Latin/Hebrew
`iw8ebcdic424`	EBCDIC Code Page 424 8-bit Latin/Hebrew
`iw8iso8859p8`	ISO 8859-8 Latin/Hebrew
`iw8machebrews`	Mac Server 8-bit Hebrew
`iw8machebrew`	Mac Client 8-bit Hebrew
`iw8mswin1255`	MS Windows Code Page 1255 8-bit Latin/Hebrew
`iw8pc1507`	IBM-PC Code Page 1507/862 8-bit Latin/Hebrew

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`ja16dbcs`	IBM EBCDIC 16-bit Japanese
`ja16ebcdic930`	IBM DBCS Code Page 290 16-bit Japanese
`ja16euctilde`	Same as `ja16euc` except for the way that the wave dash and the tilde are mapped to and from Unicode
`ja16euc`	EUC 24-bit Japanese
`ja16eucyen`	EUC 24-bit Japanese with '\' mapped to the Japanese yen character
`ja16macsjis`	Mac client Shift-JIS 16-bit Japanese
`ja16sjistilde`	Same as `ja16sjis` except for the way that the wave dash and the tilde are mapped to and from Unicode.
`ja16sjis`	Shift-JIS 16-bit Japanese
`ja16sjisyen`	Shift-JIS 16-bit Japanese with '\' mapped to the Japanese yen character
`ja16vms`	JVMS 16-bit Japanese
`ko16dbcs`	IBM EBCDIC 16-bit Korean
`ko16ksc5601`	KSC5601 16-bit Korean
`ko16ksccs`	KSCCS 16-bit Korean
`ko16mswin949`	MS Windows Code Page 949 Korean
`la8iso6937`	ISO 6937 8-bit Coded Character Set for Text Communication
`la8passport`	German Government Printer 8-bit All-European Latin
`lt8mswin921`	MS Windows Code Page 921 8-bit Lithuanian
`lt8pc772`	IBM-PC Code Page 772 8-bit Lithuanian (Latin/Cyrillic)
`lt8pc774`	IBM-PC Code Page 774 8-bit Lithuanian (Latin)
`lv8pc1117`	IBM-PC Code Page 1117 8-bit Latvian
`lv8pc8lr`	Latvian Version IBM-PC Code Page 866 8-bit Latin/Cyrillic
`lv8rst104090`	IBM-PC Alternative Code Page 8-bit Latvian (Latin/Cyrillic)
`n7siemens9780x`	Siemens 97801/97808 7-bit Norwegian
`n8pc865`	IBM-PC Code Page 865 8-bit Norwegian
`ndk7dec`	DEC VT100 7-bit Norwegian/Danish
`ne8iso8859p10`	ISO 8859-10 North European
`nee8iso8859p4`	ISO 8859-4 North and North-East European
`nl7dec`	DEC VT100 7-bit Dutch
`ru8besta`	BESTA 8-bit Latin/Cyrillic
`ru8pc855`	IBM-PC Code Page 855 8-bit Latin/Cyrillic
`ru8pc866`	IBM-PC Code Page 866 8-bit Latin/Cyrillic
`s7dec`	DEC VT100 7-bit Swedish
`s7siemens9780x`	Siemens 97801/97808 7-bit Swedish

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`s8bs2000`	Siemens 9750-62 EBCDIC 8-bit Swedish
`s8ebcdic1143`	EBCDIC Code Page 1143 8-bit Swedish
`s8ebcdic278`	EBCDIC Code Page 278/1 8-bit Swedish
`se8iso8859p3`	ISO 8859-3 South European
`sf7ascii`	ASCII 7-bit Finnish
`sf7dec`	DEC VT100 7-bit Finnish
`th8macthais`	Mac Server 8-bit Latin/Thai
`th8macthai`	Mac Client 8-bit Latin/Thai
`th8tisascii`	Thai Industrial Standard 620-2533 - ASCII 8-bit
`th8tisebcdics`	Thai Industrial Standard 620-2533 - EBCDIC Server 8-bit
`th8tisebcdic`	Thai Industrial Standard 620-2533 - EBCDIC 8-bit
`tr7dec`	DEC VT100 7-bit Turkish
`tr8dec`	DEC 8-bit Turkish
`tr8ebcdic1026s`	EBCDIC Code Page 1026 Server 8-bit Turkish
`tr8ebcdic1026`	EBCDIC Code Page 1026 8-bit Turkish
`tr8macturkishs`	Mac Server 8-bit Turkish
`tr8macturkish`	Mac Client 8-bit Turkish
`tr8mswin1254`	MS Windows Code Page 1254 8-bit Turkish
`tr8pc857`	IBM-PC Code Page 857 8-bit Turkish
`us7ascii`	ASCII 7-bit American
`us8bs2000`	Siemens 9750-62 EBCDIC 8-bit American
`us8icl`	ICL EBCDIC 8-bit American
`us8pc437`	IBM-PC Code Page 437 8-bit American
`vn8mswin1258`	MS Windows Code Page 1258 8-bit Vietnamese
`vn8vn3`	VN3 8-bit Vietnamese
`we8bs2000e`	Siemens EBCDIC.DF.04-F 8-bit West European with Euro symbol
`we8bs2000l5`	Siemens EBCDIC.DF.04-9 8-bit WE & Turkish
`we8bs2000`	Siemens EBCDIC.DF.04-1 8-bit West European
`we8dec`	DEC 8-bit West European
`we8dg`	DG 8-bit West European
`we8ebcdic1047e`	Latin 1/Open Systems 1047
`we8ebcdic1047`	EBCDIC Code Page 1047 8-bit West European
`we8ebcdic1140c`	EBCDIC Code Page 1140 Client 8-bit West European
`we8ebcdic1140`	EBCDIC Code Page 1140 8-bit West European

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`we8ebcdic1145`	EBCDIC Code Page 1145 8-bit West European
`we8ebcdic1146`	EBCDIC Code Page 1146 8-bit West European
`we8ebcdic1148c`	EBCDIC Code Page 1148 Client 8-bit West European
`we8ebcdic1148`	EBCDIC Code Page 1148 8-bit West European
`we8ebcdic284`	EBCDIC Code Page 284 8-bit Latin American/Spanish
`we8ebcdic285`	EBCDIC Code Page 285 8-bit West European
`we8ebcdic37c`	EBCDIC Code Page 37 8-bit Oracle/c
`we8ebcdic37`	EBCDIC Code Page 37 8-bit West European
`we8ebcdic500c`	EBCDIC Code Page 500 8-bit Oracle/c
`we8ebcdic500`	EBCDIC Code Page 500 8-bit West European
`we8ebcdic871`	EBCDIC Code Page 871 8-bit Icelandic
`we8ebcdic924`	Latin 9 EBCDIC 924
`we8gcos7`	Bull EBCDIC GCOS7 8-bit West European
`we8hp`	HP LaserJet 8-bit West European
`we8icl`	ICL EBCDIC 8-bit West European
`we8iso8859p15`	ISO 8859-15 West European
`we8iso8859p1`	ISO 8859-1 West European
`we8iso8859p9`	ISO 8859-9 West European & Turkish
`we8isoicluk`	ICL special version ISO8859-1
`we8macroman8s`	Mac Server 8-bit Extended Roman8 West European
`we8macroman8`	Mac Client 8-bit Extended Roman8 West European
`we8mswin1252`	MS Windows Code Page 1252 8-bit West European
`we8ncr4970`	NCR 4970 8-bit West European
`we8nextstep`	NeXTSTEP PostScript 8-bit West European
`we8pc850`	IBM-PC Code Page 850 8-bit West European
`we8pc858`	IBM-PC Code Page 858 8-bit West European
`we8pc860`	IBM-PC Code Page 860 8-bit West European
`we8roman8`	HP Roman8 8-bit West European
`yug7ascii`	ASCII 7-bit Yugoslavian
`zhs16cgb231280`	CGB2312-80 16-bit Simplified Chinese
`zhs16dbcs`	IBM EBCDIC 16-bit Simplified Chinese
`zhs16gbk`	GBK 16-bit Simplified Chinese
`zhs16maccgb231280`	Mac client CGB2312-80 16-bit Simplified Chinese
`zht16big5`	BIG5 16-bit Traditional Chinese

Table 13-1 (Cont.) Supported Oracle Character Sets

Identifier to use in parameter files and commands	Character Set
`zht16ccdc`	HP CCDC 16-bit Traditional Chinese
`zht16dbcs`	IBM EBCDIC 16-bit Traditional Chinese
`zht16dbt`	Taiwan Taxation 16-bit Traditional Chinese
`zht16hkscs31`	MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (character set conversion to and from Unicode is based on Unicode 3.1)
`zht16hkscs`	MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (character set conversion to and from Unicode is based on Unicode 3.0)
`zht16mswin950`	MS Windows Code Page 950 Traditional Chinese
`zht32euc`	EUC 32-bit Traditional Chinese
`zht32sops`	SOPS 32-bit Traditional Chinese
`zht32tris`	TRIS 32-bit Traditional Chinese

Supported Character Sets - Non-Oracle

Identifier to use in parameter files and commands	Character set
`UTF-8`	ISO-10646 UTF-8, surrogate pairs are 4 bytes per character
`UTF-16`	ISO-10646 UTF-16
`UTF-16BE`	UTF-16 Big Endian
`UTF-16LE`	UTF-16 Little Endian
`UTF-32`	ISO-10646 UTF-32
`UTF-32BE`	UTF-32 Big Endian
`UTF-32LE`	UTF-32 Little Endian
`CESU-8`	Similar to UTF-8, correspond to UCS-2 and surrogate pairs are 6 bytes per character
`US-ASCII`	US-ASCII, ANSI X34-1986
`windows-1250`	Windows Central Europe

Identifier to use in parameter files and commands	Character set
`windows-1251`	Windows Cyrillic
`windows-1252`	Windows Latin-1
`windows-1253`	Windows Greek
`windows-1254`	Windows Turkish
`windows-1255`	Windows Hebrew
`windows-1256`	Windows Arabic
`windows-1257`	Windows Baltic
`windows-1258`	Windows Vietnam
`windows-874`	Windows Thai
`cp437`	DOS Latin-1
`ibm-720`	DOS Arabic
`cp737`	DOS Greek
`cp775`	DOS Baltic
`cp850`	DOS multilingual
`cp851`	DOS Greek-1
`cp852`	DOS Latin-2
`cp855`	DOS Cyrillic
`cp856`	DOS Cyrillic / IBM
`cp857`	DOS Turkish
`cp858`	DOS Multilingual with Euro
`cp860`	DOS Portuguese

Identifier to use in parameter files and commands	Character set
`cp861`	DOS Icelandic
`cp862`	DOS Hebrew
`cp863`	DOS French
`cp864`	DOS Arabic
`cp865`	DOS Nordic
`cp866`	DOS Cyrillic / GOST 19768-87
`ibm-867`	DOS Hebrew / IBM
`cp868`	DOS Urdu
`cp869`	DOS Greek-2
`ISO-8859-1`	ISO-8859-1 Latin-1/Western Europe
`ISO-8859-2`	ISO-8859-2 Latin-2/Eastern Europe
`ISO-8859-3`	ISO-8859-3 Latin-3/South Europe
`ISO-8859-4`	ISO-8859-4 Latin-4/North Europe
`ISO-8859-5`	ISO-8859-5 Latin/Cyrillic
`ISO-8859-6`	ISO-8859-6 Latin/Arabic
`ISO-8859-7`	ISO-8859-7 Latin/Greek
`ISO-8859-8`	ISO-8859-8 Latin/Hebrew
`ISO-8859-9`	ISO-8859-9 Latin-5/Turkish
`ISO-8859-10`	ISO-8859-10 Latin-6/Nordic
`ISO-8859-11`	ISO-8859-11 Latin/Thai
`ISO-8859-13`	ISO-8859-13 Latin-7/Baltic Rim

Identifier to use in parameter files and commands	Character set
`ISO-8859-14`	ISO-8859-14 Latin-8/Celtic
`ISO-8859-15`	ISO-8859-15 Latin-9/Western Europe
`IBM037`	IBM 037-1/697-1 EBCDIC, Brazil, Canada, Netherlands, Portugal, US, and 037/1175 Traditional Chinese
`IBM01140`	IBM 1140-1/695-1 EBCDIC, Brazil, Canada, Netherlands, Portugal, US, and 1140/1175 Traditional Chinese
`IBM273`	IBM 273-1/697-1 EBCDIC, Austria, Germany
`IBM01141`	IBM 1141-1/695-1 EBCDIC, Austria, Germany
`IBM277`	IBM 277-1/697-1 EBCDIC, Denmark, Norway
`IBM01142`	IBM 1142-1/695-1 EBCIDC, Denmark, Norway
`IBM278`	IBM 278-1/697-1 EBCDIC, Finland, Sweden
`IBM01143`	IBM 1143-1/695-1 EBCDIC, Finland, Sweden
`IBM280`	IBM 280-1/697-1 EBCDIC, Italy
`IBM01144`	IBM 1144-1/695-1 EBCDIC, Italy
`IBM284`	IBM 284-1/697-1 EBCDIC, Latin America, Spain
`IBM01145`	IBM 1145-1/695-1 EBCDIC, Latin America, Spain
`IBM285`	IBM 285-1/697-1 EBCDIC, United Kingdom
`IBM01146`	IBM 1146-1/695-1 EBCDIC, United Kingdom
`IBM290`	IBM 290 EBCDIC, Japan (Katakana) Extended
`IBM297`	IBM 297-1/697-1 EBCDIC, France
`IBM01147`	IBM 1147-1/695-1 EBCDIC, France
`IBM420`	IBM 420 EBCDIC, Arabic Bilingual
`IBM424`	IBM 424/941 EBCDIC, Israel (Hebrew - Bulletin Code)

Identifier to use in parameter files and commands	Character set
`IBM500`	IBM 500-1/697-1 EBCDIC, International
`IBM01148`	IBM 1148-1/695-1 EBCDIC International
`IBM870`	IBM 870/959 EBCDIC, Latin-2 Multilingual
`IBM871`	IBM 871-1/697-1 EBCDIC Iceland
`IBM918`	IBM EBCDIC code page 918, Arabic 2
`IBM1149`	IBM 1149-1/695-1, EBCDIC Iceland
`IBM1047`	IBM 1047/103 EBCDIC, Latin-1 (Open Systems)
`ibm-803`	IBM 803 EBCDIC, Israel (Hebrew - Old Code)
`IBM875`	IBM 875 EBCDIC, Greece
`ibm-924`	IBM 924-1/1353-1 EBCDIC International
`ibm-1153`	IBM 1153/1375 EBCDIC, Latin-2 Multilingual
`ibm-1122`	IBM 1122/1037 EBCDIC, Estonia
`ibm-1157`	IBM 1157/1391 EBCDIC, Estonia
`ibm-1112`	IBM 1112/1035 EBCDIC, Latvia, Lithuania
`ibm-1156`	IBM 1156/1393 EBCDIC, Latvia, Lithuania
`ibm-4899`	IBM EBCDIC code page 4899, Hebrew with Euro
`ibm-12712`	IBM 12712 EBCDIC, Hebrew (max set including Euro)
`ibm-1097`	IBM 1097 EBCDIC, Farsi
`ibm-1018`	IBM 1018 EBCDIC, Finland Sweden (ISO-7)
`ibm-1132`	IBM 1132 EBCDIC, Laos
`ibm-1137`	IBM EBCDIC code page 1137, Devanagari

Identifier to use in parameter files and commands	Character set
`ibm-1025`	IBM 1025/1150 EBCDIC, Cyrillic
`ibm-1154`	IBM EBCDIC code page 1154, Cyrillic with Euro
`IBM1026`	IBM 1026/1152 EBCDIC, Latin-5 Turkey
`ibm-1155`	IBM EBCDIC code page 1155, Turkish with Euro
`ibm-1123`	IBM 1123 EBCDIC, Ukraine
`ibm-1158`	IBM EBCDIC code page 1158, Ukranian with Euro
`IBM838`	IBM 838/1173 EBCDIC, Thai
`ibm-1160`	IBM EBCDIC code page 1160, Thai with Euro
`ibm-1130`	IBM 1130 EBCDIC, Vietnam
`ibm-1164`	IBM EBCDIC code page 1164, Vietnamese with Euro
`ibm-4517`	IBM EBCDIC code page 4517, Arabic French
`ibm-4971`	IBM EBCDIC code page 4971, Greek
`ibm-9067`	IBM EBCDIC code page 9067, Greek 2005
`ibm-16804`	IBM EBCDIC code page 16804, Arabic
`KOI8-R`	Russian and Cyrillic (KOI8-R)
`KOI8-U`	Ukranian (KOI8-U)
`eucTH`	EUC Thai
`ibm-1162`	Windows Thai with Euro
`DEC-MCS`	DEC Multilingual
`hp-roman8`	HP Latin-1 Roman8
`ibm-901`	IBM Baltic ISO-8 CCSID 901

Identifier to use in parameter files and commands	Character set
`ibm-902`	IBM Estonia ISO-8 with Euro CCSID 902
`ibm-916`	IBM ISO8859-8 CCSID
`ibm-922`	IBM Estonia ISO-8 CCSID 922
`ibm-1006`	IBM Urdu ISO-8 CCSID 1006
`ibm-1098`	IBM Farsi PC CCSID 1098
`ibm-1124`	Ukranian ISO-8 CCSID 1124
`ibm-1125`	Ukranian without Euro CCSID 1125
`ibm-1129`	IBM Vietnamese without Euro CCSID 1129
`ibm-1131`	IBM Belarusi CCSID 1131
`ibm-1133`	IBM Lao CCSID 1133
`ibm-4909`	IBM Greek Latin ASCII CCSID 4909
`JIS_X201`	JIS X201 Japanese
`windows-932`	Windows Japanese
`windows-936`	Windows Simplified Chinese
`ibm-942`	IBM Windows Japanese
`windows-949`	Windows Korean
`windows-950`	Windows Traditional Chinese
`eucjis`	EUC Japanese
`EUC-JP`	IBM/MS EUC Japanese
`EUC-CN`	EUC Simplified Chinese, GBK
`EUC-KR`	EUC Korean

Identifier to use in parameter files and commands	Character set
`EUC-TW`	EUC Traditional Chinese
`ibm-930`	IBM 930/5026 Japanese
`ibm-933`	IBM 933 Korean
`ibm-935`	IBM 935 Simplified Chinese
`ibm-937`	IBM 937 Traditional Chinese
`ibm-939`	IBM 939/5035 Japanese
`ibm-1364`	IBM 1364 Korean
`ibm-1371`	IBM 1371 Traditional Chinese
`ibm-1388`	IBM 1388 Simplified Chinese
`ibm-1390`	IBM 1390 Japanese
`ibm-1399`	IBM 1399 Japanese
`ibm-5123`	IBM CCSID 5123 Japanese
`ibm-8482`	IBM CCSID 8482 Japanese
`ibm-13218`	IBM CCSID 13218 Japanese
`ibm-16684`	IBM CCSID 16684 Japanese
`shiftjis`	Japanese Shift JIS, Tilde 0x8160 mapped to U+301C
`gb18030`	GB-18030
`GB2312`	GB-2312-1980
`GBK`	GBK
`HZ`	HZ GB2312
`Ibm-1381`	IBM CCSID 1381 Simplified Chinese

Identifier to use in parameter files and commands	Character set
`Big5`	Big5, Traditional Chinese
`Big5-HKSCS`	Big5, HongKong ext.
`Big5-HKSCS2001`	Big5, HongKong ext. HKSCS-2001
`ibm-950`	IBM Big5, CCSID 950
`ibm-949`	CCSID 949 Korean
`ibm-949C`	IBM CCSID 949 Korean, has backslash
`ibm-971`	IBM CCSID 971 Korean EUC, KSC5601 1989
`x-IBM1363`	IBM CCSID 1363, Korean

Supported Locales

This appendix lists the locales that are supported by Oracle GoldenGate. The locale is used when comparing case-insensitive object names.

```
af
af_NA
af_ZA
am
am_ET
ar
ar_AE
ar_BH
ar_DZ
ar_EG
ar_IQ
ar_JO
ar_KW
ar_LB
ar_LY
ar_MA
ar_OM
ar_QA
ar_SA
ar_SD
ar_SY
ar_TN
ar_YE
```

```
as
as_IN
az
az_Cyrl
az_Cyrl_AZ
az_Latn
az_Latn_AZ
be
be_BY
bg
bg_BG
bn
bn_BD
bn_IN
ca
ca_ES
cs
cs_CZ
cy
cy_GB
da
da_DK
de
de_AT
de_BE
de_CH
de_DE
de_LI
de_LU
el
el_CY
el_GR
en
en_AU
en_BE
en_BW
en_BZ
en_CA
en_GB
en_HK
en_IE
en_IN
en_JM
en_MH
en_MT
en_NA
en_NZ
en_PH
en_PK
en_SG
```

```
en_TT
en_US
en_US_POSIX
en_VI
en_ZA
en_ZW
eo
es
es_AR
es_BO
es_CL
es_CO
es_CR
es_DO
es_EC
es_ES
es_GT
es_HN
es_MX
es_NI
es_PA
es_PE
es_PR
es_PY
es_SV
es_US
es_UY
es_VE
et
et_EE
eu
eu_ES
fa
fa_AF
fa_IR
fi
fi_FI
fo
fo_FO
fr
fr_BE
fr_CA
fr_CH
fr_FR
fr_LU
fr_MC
ga
ga_IE
gl
gl_ES
```

```
gu
gu_IN
gv
gv_GB
haw
haw_US
he
he_IL
hi
hi_IN
hr
hr_HR
hu
hu_HU
hy
hy_AM
hy_AM_REVISED
id
id_ID
is
is_IS
it
it_CH
it_IT
ja
ja_JP
ka
ka_GE
kk
kk_KZ
kl
kl_GL
km
km_KH
kn
kn_IN
ko
ko_KR
kok
kok_IN
kw
kw_GB
lt
lt_LT
lv
lv_LV
mk
mk_MK
ml
ml_IN
```

mr
mr_IN
ms
ms_BN
ms_MY
mt
mt_MT
nb
nb_NO
nl
nl_BE
nl_NL
nn
nn_NO
om
om_ET
om_KE
or
or_IN
pa
pa_Guru
pa_Guru_IN
pl
pl_PL
ps
ps_AF
pt
pt_BR
pt_PT
ro
ro_RO
ru
ru_RU
ru_UA
sk
sk_SK
sl
sl_SI
so
so_DJ
so_ET
so_KE
so_SO
sq
sq_AL
sr
sr_Cyrl
sr_Cyrl_BA
sr_Cyrl_ME
sr_Cyrl_RS

```
sr_Latn
sr_Latn_BA
sr_Latn_ME
sr_Latn_RS
sv
sv_FI
sv_SE
sw
sw_KE
sw_TZ
ta
ta_IN
te
te_IN
th
th_TH
ti
ti_ER
ti_ET
tr
tr_TR
uk
uk_UA
ur
ur_IN
ur_PK
uz
uz_Arab
uz_Arab_AF
uz_Cyrl
uz_Cyrl_UZ
uz_Latn
uz_Latn_UZ
vi
vi_VN
zh
zh_Hans
zh_Hans_CN
zh_Hans_SG
zh_Hant
zh_Hant_HK
zh_Hant_MO
zh_Hant_TW
```

About the Oracle GoldenGate Trail

This appendix contains information about the Oracle GoldenGate trail that you may need to know for troubleshooting, for a support case, or for other purposes. To view the Oracle GoldenGate trail records, use the Logdump utility.

Trail Recovery Mode

By default, Extract operates in *append mode*, where if there is a process failure, a recovery marker is written to the trail and Extract appends recovery data to the file so that a history of all prior data is retained for recovery purposes.

In append mode, the Extract initialization determines the identity of the last complete transaction that was written to the trail at startup time. With that information, Extract ends recovery when the commit record for that transaction is encountered in the data source; then it begins new data capture with the next committed transaction that qualifies for extraction and begins appending the new data to the trail. A data pump or Replicat starts reading again from that recovery point.

Overwrite mode is another version of Extract recovery that was used in versions of Oracle GoldenGate prior to version 10.0. In these versions, Extract overwrites the existing transaction data in the trail after the last write-checkpoint position, instead of appending the new data. The first transaction that is written is the first one that qualifies for extraction after the last read checkpoint position in the data source.

If the version of Oracle GoldenGate on the target is older than version 10, Extract will automatically revert to overwrite mode to support backward compatibility. This behavior can be controlled manually with the `RECOVERYOPTIONS` parameter.

Trail File Header Record

Each file of a trail contains a *file header record* that is stored at the beginning of the file. The file header contains information about the trail file itself. Previous versions of Oracle GoldenGate do not contain this header.

The file header is stored as a record at the beginning of a trail file preceding the data records. The information that is stored in the trail header provides enough information about the records to enable an Oracle GoldenGate process to determine whether the records are in a format that the current version of Oracle GoldenGate supports.

The trail header fields are stored as tokens, where the token format remains the same across all versions of Oracle GoldenGate. If a version of Oracle GoldenGate does not support any given token, that token is ignored. Depracated tokens are assigned a default value to preserve compatibility with previous versions of Oracle GoldenGate.

To ensure forward and backward compatibility of files among different Oracle GoldenGate process versions, the file header fields are written in a standardized token format. New tokens that are created by new versions of a process can be ignored by older versions, so that backward compatibility is maintained. Likewise, newer Oracle GoldenGate versions support older tokens. Additionally, if a token is deprecated by a new process version, a default value is assigned to the token so that older versions can still function properly. The token that specifies the file version is `COMPATIBILITY` and can be viewed in the Logdump utility and also by retrieving it with the `GGFILEHEADER` option of the `@GETENV` function.

A trail or Extract file must have a version that is equal to, or lower than, that of the process that *reads* it. Otherwise the process will abend. Additionally, Oracle GoldenGate forces the output trail or file of a data pump to be the same version as that of its input trail or file. Upon restart, Extract rolls a trail to a new file to ensure that each file is of only one version (unless the file is empty).

From Oracle GoldenGate 21c onward, for Oracle databases, you can specify a globally unique name for the database using the `DB_UNIQUE_NAME` parameter. If this database parameter is not set, then the `DB_UNIQUE_NAME` is the same as `DB_NAME`. This feature allows unique identification of the source of the trail data by viewing the trail file header.

See GETENV parameter to know about the use of the DbUniqueName token.

> **Note:**
>
> The `DbUniqueName` token will be written to trail files with 19.1 compatibility level, however prior Oracle GoldenGate releases supporting that compatibility level will ignore the new token. The token belongs to the Database Information group. The field will be limited to 65536 bytes, to allow fitting all possible values of `DB_UNIQUE_NAME`, limited to 30 characters.

Because the Oracle GoldenGate processes are decoupled and can be of different Oracle GoldenGate versions, the file header of each trail file contains a version indicator. By default, the version of a trail file is the current version of the process that created the file. If you need to set the version of a trail, use the `FORMAT` option of the `EXTTRAIL`, `EXTFILE`, `RMTTRAIL`, or `RMTFILE` parameter.

You can view the trail header with the `FILEHEADER` command in the Logdump utility. For more information about the tokens in the file header, see *Logdump Reference for Oracle GoldenGate*.

Trail Record Format

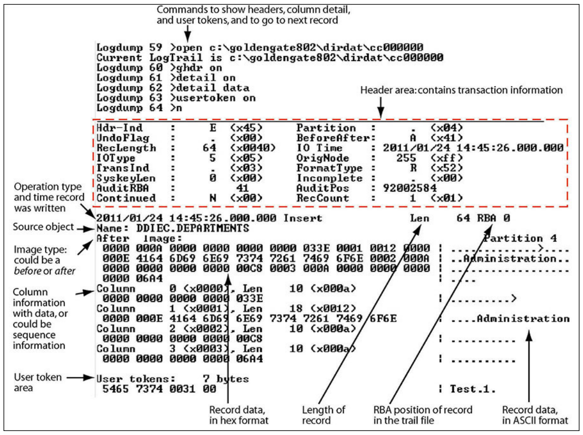
Each change record written by Oracle GoldenGate to a trail or extract file includes a header area, a data area, and possibly a user token area. The record header contains information about the transaction environment, and the data area contains the actual data values that were extracted. The token area contains information that is specified by Oracle GoldenGate users for use in column mapping and conversion.

Oracle GoldenGate trail files are unstructured. You can view Oracle GoldenGate records with the Logdump utility provided with the Oracle GoldenGate software. For more information, see Logdump Reference for Oracle GoldenGate.

> **Note:**
>
> As enhancements are made to the Oracle GoldenGate software, the trail record format is subject to changes that may not be reflected in this documentation. To view the current structure, use the Logdump utility.

Example of an Oracle GoldenGate Record

The following illustrates an Oracle GoldenGate record as viewed with Logdump. The first portion (the list of fields) is the header and the second portion is the data area. The record looks similar to this on all platforms supported by Oracle GoldenGate.


Commands to show headers, column detail, and user tokens, and to go to next record
Header area: contains transaction information
Operation type and time record was written
Source object
Image type: could be a *before* or *after*
Column information with data, or could be sequence information
User token area
Record data, in hex format
Length of record
RBA position of record in the trail file
Record data, in ASCII format

```
Logdump 59 >open c:\goldengate802\dirdat\cc000000
Current LogTrail is c:\goldengate802\dirdat\cc000000
Logdump 60 >ghdr on
Logdump 61 >detail on
Logdump 62 >detail data
Logdump 63 >usertoken on
Logdump 64 >n
Hdr-Ind     :     E  (x45)     Partition  :     .  (x04)
UndoFlag    :     .  (x00)     BeforeAfter:     A  (x41)
RecLength   :    64  (x0040)   IO Time    : 2011/01/24 14:45:26.000.000
IOType      :     5  (x05)     OrigNode   :   255  (xff)
TransInd    :     .  (x03)     FormatType :     R  (x52)
SyskeyLen   :     0  (x00)     Incomplete :     .  (x00)
AuditRBA    :         41       AuditPos   : 92002584
Continued   :     N  (x00)     RecCount   :     1  (x01)
2011/01/24 14:45:26.000.000 Insert               Len    64 RBA 0
Name: DDIEC.DEPARTMENTS
After  Image:                                              Partition 4
 0000 000A 0000 0000 0000 0000 033E 0001 0012 0000 | ..............>....
 000E 4164 6D69 6E69 7374 7261 7469 6F6E 0002 000A | ..Administration..
 0000 0000 0000 0000 00C8 0003 000A 0000 0000 0000 | ..................
 0000 06A4                                         | ....
Column     0 (x0000), Len    10 (x000a)
 0000 0000 0000 0000 033E                          | ..........>
Column     1 (x0001), Len    18 (x0012)
 0000 000E 4164 6D69 6E69 7374 7261 7469 6F6E      | ....Administration
Column     2 (x0002), Len    10 (x000a)
 0000 0000 0000 0000 00C8                          | ..........
Column     3 (x0003), Len    10 (x000a)
 0000 0000 0000 0000 06A4                          | ..........

User tokens:    7 bytes
 5465 7374 0031 00                                 | Test.1.
```

Record Header Area

The Oracle GoldenGate record header provides metadata of the data that is contained in the record and includes the following information.

- The operation type, such as an insert, update, or delete
- The before or after indicator for updates
- Transaction information, such as the transaction group and commit timestamp

Description of Header Fields

The following describes the fields of the Oracle GoldenGate record header. Some fields apply only to certain platforms.

Table 13-2 Oracle GoldenGate record header fields

Field	Description
`Hdr-Ind`	Should always be a value of `E`, indicating that the record was created by the Extract process. Any other value indicates invalid data.
`UndoFlag`	(NonStop) Conditionally set if Oracle GoldenGate is extracting aborted transactions from the TMF audit trail. Normally, `UndoFlag` is set to zero, but if the record is the backout of a previously successful operation, then `UndoFlag` will be set to 1. An undo that is performed by the disc process because of a constraint violation is not marked as an undo.
`RecLength`	The length, in bytes, of the record buffer.
`IOType`	The type of operation represented by the record. See #unique_1160/unique_1160_Connect_42_G997818 for a list of operation types.
`TransInD`	The place of the record within the current transaction. Values are: 0 — first record in transaction 1 — neither first nor last record in transaction 2 — last record in the transaction 3 — only record in the transaction
`SyskeyLen`	(NonStop) The length of the system key (4 or 8 bytes) if the source is a NonStop file and has a system key. If a system key exists, the first `Syskeylen` bytes of the record are the system key. Otherwise, `SyskeyLen` is 0.

Table 13-2 (Cont.) Oracle GoldenGate record header fields

Field	Description
`AuditRBA`	Identifies the transaction log identifier, such as the Oracle redo log sequence number.
`Continued`	(Windows and UNIX) Identifies whether or not the record is a segment of a larger piece of data that is too large to fit within one record. `LOB`s, `CLOBS`, and some `VARCHAR`s are stored in segments. Unified records that contain both before and after images in a single record (due to the `UPDATERECORDFORMAT` parameter) may exceed the maximum length of a record and may also generate segments. `Y` — the record is a segment; indicates to Oracle GoldenGate that this data continues to another record. `N` — there is no continuation of data to another segment; could be the last in a series or a record that is not a segment of larger data.
`Partition`	For Windows and UNIX records, this field will always be a value of 4 (`FieldComp` compressed record in internal format). For these platforms, the term `Partition does not` indicate that the data represents any particular logical or physical partition within the database structure. For NonStop records, the value of this field depends on the record type: • In the case of `BulkIO` operations, `Partition` indicates the number of the source partition on which the bulk operation was performed. It tells Oracle GoldenGate which source partition the data was originally written to. Replicat uses the `Partition` field to determine the name of the target partition. The file name in the record header will always be the name of the primary partition. Valid values for `BulkIO` records are 0 through 15. • For other non-bulk NonStop operations, the value can be either 0 or 4. A value of 4 indicates that the data is in `FieldComp` record format.
`BeforeAfter`	Identifies whether the record is a before (`B`) or after (`A`) image of an update operation. Records that combine both before and after images as the result of the `UPDATERECORDFORMAT` parameter are marked as after images. Inserts are always after images, deletes are always before images.
`IO Time`	The time when the operation occurred, in local time of the source system, in GMT format. This time may be the same or different for every operation in a transaction depending on when the operation occurred.
`OrigNode`	(NonStop) The node number of the system where the data was extracted. Each system in a NonStop cluster has a unique node number. Node numbers can range from 0 through 255. For records other than NonStop in origin, `OrigNode` is 0.
`FormatType`	Identifies whether the data was read from the transaction log or fetched from the database. `F` — fetched from database `R` — readable in transaction log
`Incomplete`	This field is obsolete.
`AuditPos`	Identifies the position in the transaction log of the data.
`RecCount`	(Windows and UNIX) Used for `LOB` data when it must be split into chunks to be written to the Oracle GoldenGate file. `RecCount` is used to reassemble the chunks.

Using Header Data

Some of the data available in the Oracle GoldenGate record header can be used for mapping by using the `GGHEADER` option of the `@GETENV` function or by using any of the following transaction elements as the source expression in a `COLMAP` statement in the `TABLE` or `MAP` parameter.

- `GGS_TRANS_TIMESTAMP`
- `GGS_TRANS_RBA`
- `GGS_OP_TYPE`
- `GGS_BEFORE_AFTER_IND`

Record Data Area

The data area of the Oracle GoldenGate trail record contains the following:

- The time that the change was written to the Oracle GoldenGate file
- The type of database operation
- The length of the record
- The relative byte address within the trail file
- The table name
- The data changes in hex format

The following explains the differences in record image formats used by Oracle GoldenGate on Windows, UNIX, Linux, and NonStop systems.

Full Record Image Format (NonStop Sources)

A *full record image* contains the values of all of the columns of a processed row. Full record image format is generated in the trail when the source system is HP NonStop, and only when the `IOType` specified in the record header is one of the following:

```
3 — Delete
5 — Insert
10 — Update
```

Each full record image has the same format as if retrieved from a program reading the original file or table directly. For SQL tables, datetime fields, nulls, and other data is written exactly as a program would select it into an application buffer. Although datetime fields are represented internally as an eight-byte timestamp, their external form can be up to 26 bytes expressed as a string. Enscribe records are retrieved as they exist in the original file.

When the operation type is `Insert` or `Update`, the image contains the contents of the record *after* the operation (the after image). When the operation type is `Delete`, the image contains the contents of the record *before* the operation (the before image).

For records generated from an Enscribe database, full record images are output unless the original file has the `AUDITCOMPRESS` attribute set to `ON`. When `AUDITCOMPRESS` is `ON`, compressed update records are generated whenever the original file receives an update operation. (A full image can be retrieved by the Extract process by using the `FETCHCOMPS` parameter.)

Compressed Record Image Format (Windows, UNIX, Linux Sources)

A *compressed record image* contains only the key (primary, unique, `KEYCOLS`) and the columns that changed in the processed row. By default, trail records written by processes on Windows and UNIX systems are always compressed. The format of a compressed record is as follows:

```
column_index column_length column_data[...]
```

Where:

- `column_index` is the ordinal index of the column within the source table (2 bytes).
- `colum_length` is the length of the data (2 bytes).
- `column_data` is the data, including `NULL` or `VARCHAR` length indicators.

Enscribe records written from the NonStop platform may be compressed. The format of a compressed Enscribe record is as follows:

```
field_offset field_length field_value[...]
```

Where:

- `field_offset` is the offset within the original record of the changed value (2 bytes).
- `field_length` is the length of the data (2 bytes).
- `field_value` is the data, including `NULL` or `VARCHAR` length indicators.

The first field in a compressed Enscribe record is the primary or system key.

Tokens Area

The trail record also can contain two areas for tokens. One is for internal use and is not documented here, and the other is the user tokens area. User tokens are environment values that are captured and stored in the trail record for replication to target columns or other purposes. If used, these tokens follow the data portion of the record and appear similar to the following when viewed with Logdump:

Parameter	Value
TKN-HOST	: syshq
TKN-GROUP	: EXTORA
TKN-BA_IND	: AFTER
TKN-COMMIT_TS	: 2011-01-24 17:08:59.000000
TKN-POS	: 3604496
TKN-RBA	: 4058
TKN-TABLE	: SOURCE.CUSTOMER
TKN-OPTYPE	: INSERT
TKN-LENGTH	: 57
TKN-TRAN_IND	: BEGIN

Oracle GoldenGate Operation Types

The following are some of the Oracle GoldenGate operation types. Types may be added as new functionality is added to Oracle GoldenGate. For a more updated list, use the `SHOW RECTYPE` command in the Logdump utility.

Table 13-3   Oracle GoldenGate Operation Types

Type	Description	Platform
1-Abort	A transaction aborted.	NSK TMF
2-Commit	A transaction committed.	NSK TMF
3-Delete	A record/row was deleted. A `Delete` record usually contains a full record image. However, if the `COMPRESSDELETES` parameter was used, then only key columns will be present.	All
4-EndRollback	A database rollback ended	NSK TMF
5-Insert	A record/row was inserted. An `Insert` record contains a full record image.	All
6-Prepared	A networked transaction has been prepared to commit.	NSK TMF
7-TMF-Shutdown	A TMF shutdown occurred.	NSK TMF
8-TransBegin	No longer used.	NSK TMF
9-TransRelease	No longer used.	NSK TMF
10-Update	A record/row was updated. An `Update` record contains a full record image. Note: If the partition indicator in the record header is 4, then the record is in `FieldComp` format (see below) and the update is compressed.	All
11-UpdateComp	A record/row in `TMF AuditComp` format was updated. In this format, only the changed bytes are present. A 4-byte descriptor in the format of *`2-byte_offset2-byte_length`* precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data.	NSK TMF
12-FileAlter	An attribute of a database file was altered.	NSK
13-FileCreate	A database file was created.	NSK
14-FilePurge	A database file was deleted.	NSK
15-FieldComp	A row in a SQL table was updated. In this format, only the changed bytes are present. Before images of unchanged columns are not logged by the database. A 4-byte descriptor in the format of *`2-byte_offset2-byte_length`* precedes each data fragment. The byte offset is the ordinal index of the column within the source table. The length is the length of the data. A partition indicator of 4 in the record header indicates `FieldComp` format.	All
16-FileRename	A file was renamed.	NSK
17-AuxPointer	Contains information about which AUX trails have new data and the location at which to read.	NSK TMF
18-NetworkCommit	A networked transaction committed.	NSK TMF
19-NetworkAbort	A networked transaction was aborted.	NSK TMF
90-(GGS)SQLCol	A column or columns in a SQL table were added, or an attribute changed.	NSK
100-(GGS)Purgedata	All data was removed from the file (`PURGEDATA`).	NSK
101-(GGS)Purge(File)	A file was purged.	NSK non-TMF

Table 13-3 (Cont.) Oracle GoldenGate Operation Types

Type	Description	Platform
102-(GGS)Create(File)	A file was created. The Oracle GoldenGate record contains the file attributes.	NSK non-TMF
103-(GGS)Alter(File)	A file was altered. The Oracle GoldenGate record contains the altered file attributes.	NSK non-TMF
104-(GGS)Rename(File)	A file was renamed. The Oracle GoldenGate record contains the original and new names.	NSK non-TMF
105-(GGS)Setmode	A `SETMODE` operation was performed. The Oracle GoldenGate record contains the `SETMODE` information.	NSK non-TMF
106-GGSChangeLabel	A `CHANGELABEL` operation was performed. The Oracle GoldenGate record contains the `CHANGELABEL` information.	NSK non-TMF
107-(GGS)Control	A `CONTROL` operation was performed. The Oracle GoldenGate record contains the `CONTROL` information.	NSK non-TMF
115 and 117 (GGS)KeyFieldComp(32)	A primary key was updated. The Oracle GoldenGate record contains the before image of the key and the after image of the key and the row. The data is in `FieldComp` format (compressed), meaning that before images of unchanged columns are not logged by the database.	Windows and UNIX
116-LargeObject 116-LOB	Identifies a `RAW`, `BLOB`, `CLOB`, or `LOB` column. Data of this type is stored across multiple records.	Windows and UNIX
132-(GGS) SequenceOp	Identifies an operation on a sequence.	Windows and UNIX
134-UNIFIED UPDATE 135-UNIFIED PKUPDATE	Identifies a unified trail record that contains both before and after values in the same record. The before image in a `UNIFIED UPDATE` contains all of the columns that are available in the transaction record for both the before and after images. The before image in a `UNIFIED PKUPDATE` contains all of the columns that are available in the transaction record, but the after image is limited to the primary key columns and the columns that were modified in the `UPDATE`.	Windows and UNIX
160 - DDL_Op	Identifies a DDL operation	Windows and UNIX
161- RecordFragment	Identifies part of a large row that must be stored across multiple records (more than just the base record).	Windows and UNIX
200-GGSUnstructured Block 200-BulkIO	A `BULKIO` operation was performed. The Oracle GoldenGate record contains the `RAW DP2` block.	NSK non-TMF
201 through 204	These are different types of NonStop trace records. Trace records are used by Oracle GoldenGate support analysts. The following are descriptions. • `ARTYPE_FILECLOSE_GGS 201` — the source application closed a file that was open for unstructured I/O. Used by Replicat • `ARTYPE_LOGGERTS_GGS 202` — Logger heartbeat record • `ARTYPE_EXTRACTERTS_GGS 203` — unused • `ARTYPE_COLLECTORTS_GGS 204` — unused	NSK non-TMF

Table 13-3 (Cont.) Oracle GoldenGate Operation Types

Type	Description	Platform
205-GGSComment	Indicates a comment record created by the Logdump utility. Comment records are created by Logdump at the beginning and end of data that is saved to a file with Logdump's `SAVE` command.	All
249 through 254	These are different types of NonStop trace records. Trace records are used by Oracle GoldenGate support analysts. The following are descriptions. • `ARTYPE_LOGGER_ADDED_STATS 249` — a stats record created by Logger when the source application closes its open on Logger (if `SENDERSTATS` is enabled and stats are written to the logtrail) • `ARTYPE_LIBRARY_OPEN 250` — written by `BASELIB` to show that the application opened a file • `ARTYPE_LIBRARY_CLOSE 251` — written by `BASELIB` to show that the application closed a file. • `ARTYPE_LOGGER_ADDED_OPEN 252` — unused • `ARTYPE_LOGGER_ADDED_CLOSE` 253 — unused • `ARTYPE_LOGGER_ADDED_INFO 254` — written by Logger and contains information about the source application that performed the I/O in the subsequent record (if `SENDERSTATS` is enabled and stats are written to the logtrail). The file name in the trace record is the object file of the application. The trace data has the application process name and the name of the library (if any) that it was running with.	NSK non-TMF

Oracle GoldenGate Trail Header Record

In addition to the transaction-related records that are in the Oracle GoldenGate trail, each trail file contains a file header.

The file header is stored as a record at the beginning of a trail file preceding the data records. The information that is stored in the trail header provides enough information about the records to enable an Oracle GoldenGate process to determine whether the records are in a format that the current version of Oracle GoldenGate supports.

The trail header fields are stored as tokens, where the token format remains the same across all versions of Oracle GoldenGate. If a version of Oracle GoldenGate does not support any given token, that token is ignored. Depracated tokens are assigned a default value to preserve compatibility with previous versions of Oracle GoldenGate.

You can view the trail header with the `FILEHEADER` command in the Logdump utility. For more information about the tokens in the file header, see *Logdump Reference for Oracle GoldenGate*.

About Checkpoints

This appendix provides information about checkpoints. When working with Oracle GoldenGate, you might need to refer to the checkpoints that are made by a process. Checkpoints save the state of the process for recovery purposes. Extract and Replicat use checkpoints.

About Extract Checkpoints

Extract checkpoint positions are composed of read checkpoints in the data source and write checkpoints in the trail. The following is a sampling of checkpoint information displayed with the `INFO EXTRACT` command with the `SHOWCH` option. In this case, the data source is an Oracle RAC database cluster, so there is thread information included in the output. You can view past checkpoints by specifying the number of them that you want to view after the `SHOWCH` argument.

Example 13-1   INFO EXTRACT with SHOWCH

```
EXTRACT    JC108XT Last Started 2011-01-01 14:15   Status ABENDED
Checkpoint Lag       00:00:00 (updated 00:00:01 ago)
Log Read Checkpoint  File /orarac/oradata/racq/redo01.log
            2011-01-01 14:16:45  Thread 1, Seqno 47, RBA 68748800
Log Read Checkpoint  File /orarac/oradata/racq/redo04.log
            2011-01-01 14:16:19  Thread 2, Seqno 24, RBA 65657408

Current Checkpoint Detail:

Read Checkpoint #1

    Oracle RAC Redo Log
    Startup Checkpoint (starting position in data source):
        Thread #: 1
        Sequence #: 47
        RBA: 68548112
        Timestamp: 2011-01-01 13:37:51.000000
        SCN: 0.8439720
        Redo File: /orarac/oradata/racq/redo01.log

Recovery Checkpoint (position of oldest unprocessed transaction in data source):
        Thread #: 1
        Sequence #: 47
        RBA: 68748304
        Timestamp: 2011-01-01 14:16:45.000000
        SCN: 0.8440969
        Redo File: /orarac/oradata/racq/redo01.log

    Current Checkpoint (position of last record read in the data source):
        Thread #: 1
        Sequence #: 47
        RBA: 68748800
        Timestamp: 2011-01-01 14:16:45.000000
        SCN: 0.8440969
        Redo File: /orarac/oradata/racq/redo01.log

Read Checkpoint #2

    Oracle RAC Redo Log

    Startup Checkpoint(starting position in data source):
        Sequence #: 24
        RBA: 60607504
        Timestamp: 2011-01-01 13:37:50.000000
        SCN: 0.8439719
        Redo File: /orarac/oradata/racq/redo04.log

Recovery Checkpoint (position of oldest unprocessed transaction in data source):
        Thread #: 2
        Sequence #: 24
        RBA: 65657408
        Timestamp: 2011-01-01 14:16:19.000000
        SCN: 0.8440613
        Redo File: /orarac/oradata/racq/redo04.log

    Current Checkpoint (position of last record read in the data source):
        Thread #: 2
        Sequence #: 24
        RBA: 65657408
        Timestamp: 2011-01-01 14:16:19.000000
        SCN: 0.8440613
        Redo File: /orarac/oradata/racq/redo04.log

Write Checkpoint #1

    GGS Log Trail

    Current Checkpoint (current write position):

        Sequence #: 2
        RBA: 2142224
        Timestamp: 2011-01-01 14:16:50.567638
        Extract Trail: ./dirdat/eh

    Header:
        Version = 2
        Record Source = A
        Type = 6
        # Input Checkpoints = 2
        # Output Checkpoints = 1

    File Information:
        Block Size = 2048
        Max Blocks = 100
        Record Length = 2048
        Current Offset = 0

    Configuration:
        Data Source = 3
        Transaction Integrity = 1
        Task Type = 0

    Status:
        Start Time = 2011-01-01 14:15:14
        Last Update Time = 2011-01-01 14:16:50
        Stop Status = A
        Last Result = 400
```

See Internal Checkpoint Information for information about the internal information that starts with the `Header` entry in the `SHOWCH` output.

About Extract read checkpoints

Extract places read checkpoints in the data source.

Startup Checkpoint

The startup checkpoint is the first checkpoint that is made in the data source when the process starts. This statistic is composed of the following:

- `Thread #`: The number of the Extract thread that made the checkpoint, if Oracle GoldenGate is running in an Oracle RAC environment. Otherwise, this statistic is not displayed.
- `Sequence #`: The sequence number of the transaction log where the checkpoint was made.
- `RBA`: The relative byte address of the record at which the checkpoint was made.
- `Timestamp`: The timestamp of the record at which the checkpoint was made.
- `SCN`: The system change number of the record at which the checkpoint was made.
- `Redo File`: The path name of the transaction log containing the record where the checkpoint was made.

Recovery Checkpoint

The recovery checkpoint is the position in the data source of the record containing the oldest transaction not yet processed by Extract. The fields for this statistic are the same as those of the other read checkpoint types.

Current Checkpoint

The current checkpoint is the position of the last record read by Extract in the data source. This should match the `Log Read Checkpoint` statistic shown in the summary and in the basic `INFO EXTRACT` command without options. The fields for this statistic are the same as those of the other read checkpoint types.

About Extract Write Checkpoints

Extract places a write checkpoint, known as the current checkpoint, in the trail. The current checkpoint is the position in the trail where Extract is currently writing. This statistic is composed of the following:

- `Sequence #`: The sequence number of the trail file where the checkpoint was written.
- `RBA`: The relative byte address of the record in the trail file at which the checkpoint was made.
- `Timestamp`: The timestamp of the record at which the checkpoint was made.
- `Extract trail`: The relative path name of the trail.
- `Trail Type`: Identifies the trail type. `EXTTRAIL` identifies the trail as a local trail, which means that it is directly accessible by Oracle GoldenGate processes through the host filesystem. `RMTTRAIL` identifies the trail as a remote trail, which means it is not directly accessible by Oracle GoldenGate processes through the host filesystem. A trail stored on a shared network device and accessible through NFS-like services are considered local because they are accessible transparently through the host filesystem.

Replicat Checkpoints

Replicat makes checkpoints in the trail file to mark its last read position. To view process checkpoints, use the `INFO REPLICAT` command with the `SHOWCH` option. The basic command shows current checkpoints. To view a specific number of previous checkpoints, type the value after the `SHOWCH` argument.

Example 13-2 INFO REPLICAT, SHOWCH

```
REPLICAT   JC108RP   Last Started 2011-01-12 13:10   Status RUNNING
Checkpoint Lag       00:00:00 (updated 111:46:54 ago)
Log Read Checkpoint  File ./dirdat/eh000000000
                     First Record  RBA 3702915
Current Checkpoint Detail:
    Read Checkpoint #1
    GGS Log Trail
    Startup Checkpoint(starting position in data source):
    Sequence #: 0
    RBA: 3702915
    Timestamp: Not Available
    Extract Trail: ./dirdat/eh
    Current Checkpoint (position of last record read in the data source):
    Sequence #: 0
    RBA: 3702915
    Timestamp: Not Available
    Extract Trail: ./dirdat/eh
    Header:
    Version = 2
    Record Source = A
    Type = 1
    # Input Checkpoints = 1
    # Output Checkpoints = 0
    File Information:
    Block Size = 2048
    Max Blocks = 100
    Record Length = 2048
    Current Offset = 0
    Configuration:
    Data Source = 0
    Transaction Integrity = -1
    Task Type = 0
    Status:
    Start Time = 2011-01-12 13:10:13
    Last Update Time = 2011-01-12 21:23:31
    Stop Status = A
    Last Result = 400
```

About Replicat Checkpoints

The following describes the detail of the Replicat checkpoints in the trail.

Startup Checkpoint

The *`startup checkpoint`* is the first checkpoint made in the trail when the process starts. Comprising this statistic are:

- `Sequence #:` The sequence number of the trail file where the checkpoint was written.
- `RBA:` The relative byte address of the record at which the checkpoint was made.
- `Timestamp:` The timestamp of the record at which the checkpoint was made.
- `Extract Trail:` The relative path name of the trail.

Current Checkpoint

The *`current checkpoint`* is the position of the last record read by Replicat in the trail. This should match the `Log Read Checkpoint` statistic shown in the summary and in the basic `INFO`

`REPLICAT` command without options. The fields for this statistic are the same as those of the `Startup Checkpoint`.

Internal Checkpoint Information

The `INFO` command with the `SHOWCH` option not only displays current checkpoint entries, but it also displays metadata information about the record itself. This information is not documented and is for use by the Oracle GoldenGate processes and by support personnel when resolving a support case. The metadata is contained in the following entries in the `SHOWCH` output.

```
Header:
Version = 2
Record Source = A
Type = 1
# Input Checkpoints = 1
# Output Checkpoints = 0
File Information:
Block Size = 2048
Max Blocks = 100
Record Length = 2048
Current Offset = 0
Configuration:
Data Source = 0
Transaction Integrity = -1
Task Type = 0
Status:
Start Time = 2011-01-12 13:10:13
Last Update Time = 2011-01-12 21:23:31
Stop Status = A
Last Result = 400
```

Oracle GoldenGate Checkpoint Tables

When database checkpoints are being used, Oracle GoldenGate creates a checkpoint table with a user-defined name in the database upon execution of the `ADD CHECKPOINTTABLE` command, or a user can create the table by using the `chkpt_`*`db`*`_create.sql` script (where *`db`* is an abbreviation of the type of database that the script supports).

There are two tables: the main checkpoint table and an auxiliary checkpoint table that is created automatically. The auxiliary table, known as the *transaction table*, bears the name of the primary checkpoint table appended with `_lox`. Each Replicat, or each thread of a coordinated Replicat, uses one row in the checkpoint table to store its progress information.

At checkpoint time, there typically are some number of transactions (among the total *n* transactions) that were applied, and the rest are still in process. For example, if Replicat is processing a group of *n* transactions ranging from CSN1 to CSN3. CSN1 is the high watermark and CSN3 is the low watermark. Any transaction with a CSN higher than the high watermark has not been processed, and any transaction with a CSN lower than the low watermark has already been processed. Completed transactions are stored in the `LOG_CMPLT_XID` column of the checkpoint table. Any overflow of these transactions is stored in the transaction table (auxiliary checkpoint table) in the `LOG_CMPLT_XID` column of that table.

Currently, Replicat (or each Replicat thread of a coordinated Replicat) applies transactions serially (not in parallel); therefore, the high watermark (the `LOG_CSN` value in the table) is always the same as the low watermark (the `LOG_CMPLT_CSN` value in the table), and there typically is only one transaction ID in the `LOG_CMPLT_XID` column. The only exception is when there are multiple transactions sharing the same CSN.

Do not change the names or attributes of the columns in these tables. You can change table storage attributes as needed.

Table 13-4 Checkpoint table definition

Column	Description
`GROUP_NAME (primary key)`	The name of a Replicat group using this table for checkpoints. There can be multiple Replicat groups using the same table. This column is part of the primary key.
`GROUP_KEY (primary key)`	A unique identifier that, together with `GROUPNAME`, uniquely identifies a checkpoint regardless of how many Replicat groups are writing to the same table. This column is part of the primary key.
`SEQNO`	The sequence number of the input trail that Replicat was reading at the time of the checkpoint.
`RBA`	The relative byte address that Replicat reached in the trail identified by `SEQNO`. `RBA` + `SEQNO` provide an absolute position in the trail that identifies the progress of Replicat at the time of checkpoint.
`AUDIT_TS`	The timestamp of the commit of the source transaction.
`CREATE_TS`	The date and time when the checkpoint table was created.
`LAST_UPDATE_TS`	The date and time when the checkpoint table was last updated.
`CURRENT_DIR`	The current Oracle GoldenGate home directory or folder.
`LOG_BSN`	The `LOG_BSN` provides information needed to set Extract back in time to reprocess transactions. Some filtering by Replicat is necessary because Extract will likely re-generate a small amount of data that was already applied by Replicat.
`LOG_CSN`	Stores the high watermark, or the upper boundary, of the CSNs. Any transaction with a CSN higher than this value has not been processed.
`LOG_XID`	Not used. Retained for backward compatibility.
`LOG_CMPLT_CSN`	Stores the low watermark, or the lower boundary, of the CSNs. Any transaction with a lower CSN than this value has already been processed.
`LOG_CMPLT_XIDS`	Stores the transactions between the high and low watermarks that are already applied.
`VERSION`	The version of the checkpoint table format. Enables future enhancements to be identified as version numbers of the table.

Table 13-5 Transaction table definition

Column	Description
`GROUP_NAME`	The name of a Replicat group using this table for checkpoints. There can be multiple Replicat groups using the same table. This column is part of the primary key of the transaction table.
`GROUP_KEY`	A unique identifier that, together with `GROUPNAME`, uniquely identifies a checkpoint regardless of how many Replicat groups are writing to the same table. This column is part of the primary key of the transaction table.

Table 13-5 (Cont.) Transaction table definition

Column	Description
`LOG_CMPLT_CSN`	The foreign key that references the checkpoint table. This column is part of the primary key of the transaction table.
`LOG_CMPLT_XIDS_SEQ`	Creates unique rows in the event there are so many overflow transactions that multiple rows are required to store them all. This column is part of the primary key of the transaction table.
`LOG_CMPLT_XIDS`	Stores the overflow of transactions between the high and low watermarks that are already applied.

Supporting Changes to XML Schemas

Learn about supporting changes to an XML schema. Extract does not support the capture of changes made to an XML schema.

Supporting RegisterSchema

`RegisterSchema` can be handled by registering the schema definition on both source and target databases before any table is created that references the XML schema.

Supporting DeleteSchema

Issue `DeleteSchema` on the source database first.

Once Replicat is caught up with the changes made to the source database, issue the `DeleteSchema` call on the target database.

Supporting CopyEvolve

The CopyEvolve procedure evolves, or changes, a schema and can modify tables by adding or removing columns.

The CopyEvolve procedure can also be used to change whether or not XML documents are valid. Handling `CopyEvolve` requires more coordination. Use the following procedure if you are issuing `CopyEvolve` on the source database.

1. Quiesce changes to dependent tables on the source database.
2. Execute the `CopyEvolve` on the primary or source database.
3. Wait for Replicat to finish applying all of the data from those tables to the target database.
4. Stop Replicat.
5. Apply the `CopyEvolve` on the target database.
6. Restart Replicat.

Preparing DBFS for an Active-Active Configuration

This appendix contains steps to configure Oracle GoldenGate to function within an active-active bidirectional or multi-directional environment where Oracle Database File System (DBFS) is in use on both (or all) systems.

Supported Operations and Prerequisites

This topic lists what is supported by Oracle GoldenGate for DBFS.

Oracle GoldenGate for DBFS supports the following:

- Supported DDL (like `TRUNCATE` or `ALTER`) on DBFS objects except for `CREATE` statements on the DBFS objects. `CREATE` on DBFS must be excluded from the configuration, as must any schemas that will hold the created DBFS objects. The reason to exclude `CREATES` is that the metadata for DBFS must be properly populated in the SYS dictionary tables (which itself is excluded from Oracle GoldenGate capture by default).
- Capture and replication of DML on the tables that underlie the DBFS file system.

The procedures that follow assume that Oracle GoldenGate is configured properly to support active-active configuration. This means that it must be:

- Installed according to the instructions in this guide.
- Configured according to the instructions in the Oracle GoldenGate Windows and UNIX Administrator's Guide.

Applying the Required Patch

Apply the Oracle DBFS patch for bug-9651229 on both databases.

To determine if the patch is installed, run the following query:

```
connect / as sysdba
select  procedure_name
from    dba_procedures
where   object_name = 'DBMS_DBFS_SFS_ADMIN'
and procedure_name = 'PARTITION_SEQUENCE';
```

The query should return a single row. Anything else indicates that the proper patched version of DBFS is not available on your database.

Examples Used in these Procedures

The following procedures assume two systems and configure the environment so that DBFS users on both systems see the same DBFS files, directories, and contents that are kept in synchronization with Oracle GoldenGate.

It is possible to extend these concepts to support three or more peer systems.

Partitioning the DBFS Sequence Numbers

DBFS uses an internal sequence-number generator to construct unique names and unique IDs.

These steps partition the sequences into distinct ranges to ensure that there are no conflicts across the databases. After this is done, further DBFS operations (both creation of new file systems and subsequent file system operations) can be performed without conflicts of names, primary keys, or IDs during DML propagation.

1. Connect to each database as `sysdba`.

   Issue the following query on each database.

   ```
   select last_number
   from dba_sequences
   where sequence_owner = 'SYS'
   and sequence_name = 'DBFS_SFS_$FSSEQ'
   ```

2. From this query, choose the maximum value of `LAST_NUMBER` across both systems, or pick a high value that is significantly larger than the current value of the sequence on either system.
3. Substitute this value ("`maxval`" is used here as a placeholder) in both of the following procedures. These procedures logically index each system as `myid=0` and `myid=1`.

   Node1

   ```
   declare
   begin
   dbms_dbfs_sfs_admin.partition_sequence(nodes => 2, myid => 0, newstart => :maxval);
   commit;
   end;
   /
   ```

   Node 2

   ```
   declare
   begin
   dbms_dbfs_sfs_admin.partition_sequence( nodes => 2, myid => 1, newstart => :maxval);
   commit;
   end;
   /
   ```

   > **Note:**
   >
   > Notice the difference in the value specified for the `myid` parameter. These are the different index values.

   For a multi-way configuration among three or more databases, you could make the following alterations:

   - Adjust the maximum value that is set for `maxval` upward appropriately, and use that value on all nodes.
   - Vary the value of `myid` in the procedure from 0 for the first node, 1 for the second node, 2 for the third one, and so on.

4. (Recommended) After (and only after) the DBFS sequence generator is partitioned, create a new DBFS file system on each system, and use only these file systems for DML propagation with Oracle GoldenGate.

> **Note:**
>
> If you must retain old file systems, open a service request with Oracle Support.

Configuring the DBFS file system

To replicate DBFS file system operations, use a configuration that is similar to the standard bi-directional configuration for DML.

Some guidelines to follow while configuring Oracle GoldenGate for DBFS are:

- Use matched pairs of identically structured tables.
- Allow each database to have write privileges to opposite tables in a set, and set the other one in the set to read-only. For example:
  - Node1 writes to local table `t1` and these changes are replicated to `t1` on Node2.
  - Node2 writes to local table `t2` and these changes are replicated to `t2` on Node1.
  - On Node1, `t2` is read-only. On Node2, `t1` is read-only.

DBFS file systems make this kind of table pairing simple because:

- The tables that underlie the DBFS file systems have the same structure.
- These tables are modified by simple, conventional DML during higher-level file system operations.
- The DBFS ContentAPI provides a way of unifying the namespace of the individual DBFS stores by means of mount points that can be qualified as read-write or read-only.

The following steps create two DBFS file systems (in this case named `FS1` and `FS2`) and set them to be read-write or read, as appropriate.

1. Run the following procedure to create the two file systems. (Substitute your store names for `FS1` and `FS2`.)
2. Run the following procedure to give each file system the appropriate access rights. (Substitute your store names for `FS1` and `FS2`.)

   In this example, note that on Node 1, store `FS1` is read-write and store `FS2` is read-only, while on Node 2 the converse is true: store `FS1` is read-only and store `FS2` is read-write.

   Note also that the read-write store is mounted as *local* and the read-only store is mounted as *remote*. This provides users on each system with an identical namespace and identical semantics for read and write operations. Local path names can be modified, but remote path names cannot.

Example 13-3

```
declare
dbms_dbfs_sfs.createfile system('FS1');
dbms_dbfs_sfs.createfile system('FS2');

dbms_dbfs_content.registerStore('FS1',
'posix', 'DBMS_DBFS_SFS');
```

```
dbms_dbfs_content.registerStore('FS2',
'posix', 'DBMS_DBFS_SFS');
commit;
end;
/
```

Example 13-4 Node 1

```
declare
dbms_dbfs_content.mountStore('FS1', 'local');
dbms_dbfs_content.mountStore('FS2', 'remote',
read_only => true);
commit;
end;
/
```

Example 13-5 Node 2

```
declare
dbms_dbfs_content.mountStore('FS1', 'remote',
read_only => true);
dbms_dbfs_content.mountStore('FS2', 'local');
commit;
end;
/
```

Mapping Local and Remote Peers Correctly

The names of the tables that underlie the DBFS file systems are generated internally and dynamically.

Continuing with the preceding example, there are:

- Two nodes (Node 1 and Node 2 in the example).
- Four stores: two on each node (`FS1` and `FS2` in the example).
- Eight underlying tables: two for each store (a table and a ptable). These tables must be identified, specified in Extract `TABLE` statements, and mapped in Replicat `MAP` statements.

1. To identify the table names that back each file system, issue the following query. (Substitute your store names for `FS1` and `FS2`.)

   The output looks like the following examples.

2. Identify the tables that are *locally read-write* to Extract by creating the following `TABLE` statements in the Extract parameter files. (Substitute your pluggable database names, schema names, and table names as applicable.)
3. Link changes on each remote file system to the corresponding local file system by creating the following `MAP` statements in the Replicat parameter files. (Substitute your pluggable database, schema and table names.)

   This mapping captures and replicates local read-write *source* tables to remote read-only peer tables:

   - file system changes made to `FS1` on Node 1 propagate to `FS1` on Node 2.
   - file system changes made to `FS2` on Node 2 propagate to `FS2` on Node1.

   Changes to the file systems can be made through the DBFS ContentAPI (package `DBMS_DBFS_CONTENT`) of the database or through `dbfs_client` mounts and conventional file systems tools.

All changes are propagated in both directions.

- A user at the virtual root of the DBFS namespace on each system sees identical content.
- For mutable operations, users use the `/local` sub-directory on each system.
- For read operations, users can use either of the `/local` or `/remote` sub-directories, depending on whether they want to see local or remote content.

Example 13-6

```
select fs.store_name, tb.table_name, tb.ptable_name
from table(dbms_dbfs_sfs.listTables) tb,
table(dbms_dbfs_sfs.listfile systems) fs
where   fs.schema_name = tb.schema_name
and fs.table_name = tb.table_name
and fs.store_name in ('FS1', 'FS2')
;
```

Example 13-7   Example output: Node 1 (Your Table Names Will Be Different.)

```
STORE NAME     TABLE_NAME     PTABLE_NAME
-------------  -------------  -------------
FS1            SFS$_FST_100   SFS$_FSTP_100
FS2            SFS$_FST_118   SFS$_FSTP_118
```

Example 13-8   Example output: Node 2 (Your Table Names Will Be Different.)

```
STORE NAME     TABLE_NAME     PTABLE_NAME
-------------  -------------  -------------
FS1            SFS$_FST_101   SFS$_FSTP_101
FS2            SFS$_FST_119   SFS$_FSTP_119
```

Example 13-9   Node1

```
TABLE [container.]schema.SFS$_FST_100
TABLE [container.]schema.SFS$_FSTP_100;
```

Example 13-10   Node2

```
TABLE [container.]schema.SFS$_FST_119
TABLE [container.]schema.SFS$_FSTP_119;
```

Example 13-11   Node1

```
MAP [container.]schema.SFS$_FST_119, TARGET [container.]schema.SFS$_FST_118;
MAP [container.]schema.SFS$_FSTP_119, TARGET [container.]schema.SFS$_FSTP_118
```

Example 13-12   Node2

```
MAP [container.]schema.SFS$_FST_100, TARGET [container.]schema.SFS$_FST_101;MAP
[container.]schema.SFS$_FSTP_100, TARGET [container.]schema.SFS$_FSTP_101;
```