ORACLE UNIVERSITY

Oracle GoldenGate 12*c*: Advanced Configuration for Oracle

Student Guide
D89113GC10
Edition 1.0 | July 2015 | D91916

Learn more from Oracle University at **oracle.com/education/**

ORACLE

Author
Elio Bonazzi

Technical Contributors and Reviewers
Joe Debuzna
Randall Richeson

Graphic Editors
Seema Bopaiah
Maheshwari Krishnamurthy
Rajiv Chandrabhanu

Editor
Vijayalakshmi Narasimhan

Publishers
Sumesh Koshy
Joseph Fernandez
Michael Sebastian Almeida
Jayanthy Keshavamurthy

Contents

6 Data Mapping, Data Selection/Filtering, and Data Transformation

1 Introduction

Course Objectives

After completing this course, you should be able to:

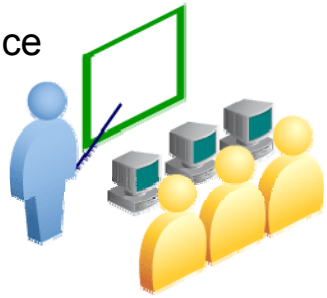
- Describe the Integrated Capture and the new Integrated Apply mechanisms to replicate complex data types and compressed data and improve replication performance
- Configure and set up local and downstream deployment modes
- Replicate data stored in multitenant container databases (Oracle 12*c*)
- Use Oracle GoldenGate with Oracle RAC, Oracle Clusterware, and Oracle Database File System (DBFS) to achieve Maximum Availability

Course Objectives

- Configure and implement the Oracle GoldenGate Event Marker system to enable data transformation
- Implement custom behavior by using user exits
- Perform zero-downtime database migration
- Set up and manage advanced deployment models to support bidirectional and three-node multi-master replication configurations
- Simulate node failure in an Oracle RAC environment that is configured for maximum availability and verify that Oracle GoldenGate survives the failure and continues to replicate

Target Audience

- This course is for DBAs and system administrators who use (or plan to use) the advanced features of Oracle GoldenGate.
- Prerequisites are either of the following:
  - *Oracle GoldenGate Fundamentals*
  - Extensive use of—and exposure to—Oracle GoldenGate concepts and implementations
- Helpful skills include:
  - Oracle Database administration experience
  - Oracle RAC experience



ORACLE

If you are concerned about whether your experience fulfills the course prerequisites, ask the instructor.

Introductions

- Introduce yourself.
- Tell us about:
  - Your company and role
  - Your experience with Oracle GoldenGate

Agenda: Day 1

- Lessons
  1. Introduction
  2. Oracle GoldenGate Integrated Capture/Apply
  3. Integrated Capture Deployment and Required Components
- Practices and optional workshops

ORACLE

- **Lessons:** The training involves a combination of lecture presentations, whiteboard technical diagrams, and demonstrations.
- **Practices:** You gain hands-on experience in installing, configuring, and managing a GoldenGate environment.
- **Workshops: (**Optional) Your instructor may choose to conduct workshops on various topics.

Agenda: Day 2

- Lessons
  4. Oracle GoldenGate with Oracle Real Application Clusters Configuration
  5. Oracle GoldenGate Event Marker System
  6. Data Mapping, Data Selection/Filtering, and Data Transformation
  7. Custom Behavior Through User Exits
- Practices and optional workshops

Agenda: Day 3

- Lessons
  8. Configuring Zero Down–Time Migration Replication
  9. Bidirectional Replication: Two-Node Configuration
- Practices and optional workshops

Agenda: Day 4

- Lessons
  10. Conflict Detection and Resolution: Custom Techniques
  11. Multi-Master Replication Topology: Three-Node Configuration
  12. Active Data Guard and Oracle GoldenGate: How to Achieve Maximum Availability
- Practices and optional workshops
- Questions and answers

Course Practices

- Most topics are reinforced with a hands-on activity.
- Many exercises include scripted solutions.

Classroom Guidelines

- The instructor starts each session at the scheduled time.
- Feel free to ask questions, but be respectful of the topic that is being discussed and the interests of other students.
- Ensure that cell phones and pagers are silent.

More Information

Topic	Website
Education	http://oracle.com/education
Product Documentation	http://www.oracle.com/technology/documentation
Product Downloads	http://www.oracle.com/technetwork/indexes/downloads
Product Articles	http://www.oracle.com/technetwork/articles
Product Support	http://www.oracle.com/support
Product Forums	http://forums.oracle.com
Product Tutorials/Demos	http://www.oracle.com/technetwork/tutorials/index.html
Online Learning Library (OLL)	www.oracle.com/goto/oll

ORACLE

After you complete the course, Oracle provides a variety of ways to access additional information.

Related Training

Course
Oracle GoldenGate 12*c* Fundamentals for Oracle
Oracle GoldenGate 12*c* Management Pack: Overview
Oracle GoldenGate 12*c* Troubleshooting and Tuning
Oracle GoldenGate Director 11*g*: Essentials
Oracle GoldenGate 12*c* Management Pack: Overview

Your instructor can provide additional information about the availability and content of the courses listed in the slide.

2

Oracle GoldenGate Integrated Capture/Apply

Objectives

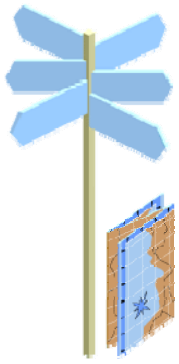
After completing this lesson, you should be able to:

- Discuss the role and function of an Oracle GoldenGate Extract process
- Compare and contrast Classic Capture with Integrated Capture
- Describe Oracle GoldenGate Integrated Capture features and functionality
- Identify the supported Integrated Capture features according to the source RDBMS release
- Describe Oracle GoldenGate Integrated Apply features and functionality

ORACLE

Roadmap

- Role and function of an Oracle GoldenGate Extract process
- Classic Capture versus Integrated Capture
- Oracle GoldenGate Integrated Capture features and functionality
- Supported Integrated Capture features according to the RDBMS releases
- Oracle GoldenGate Integrated Apply features and functionality



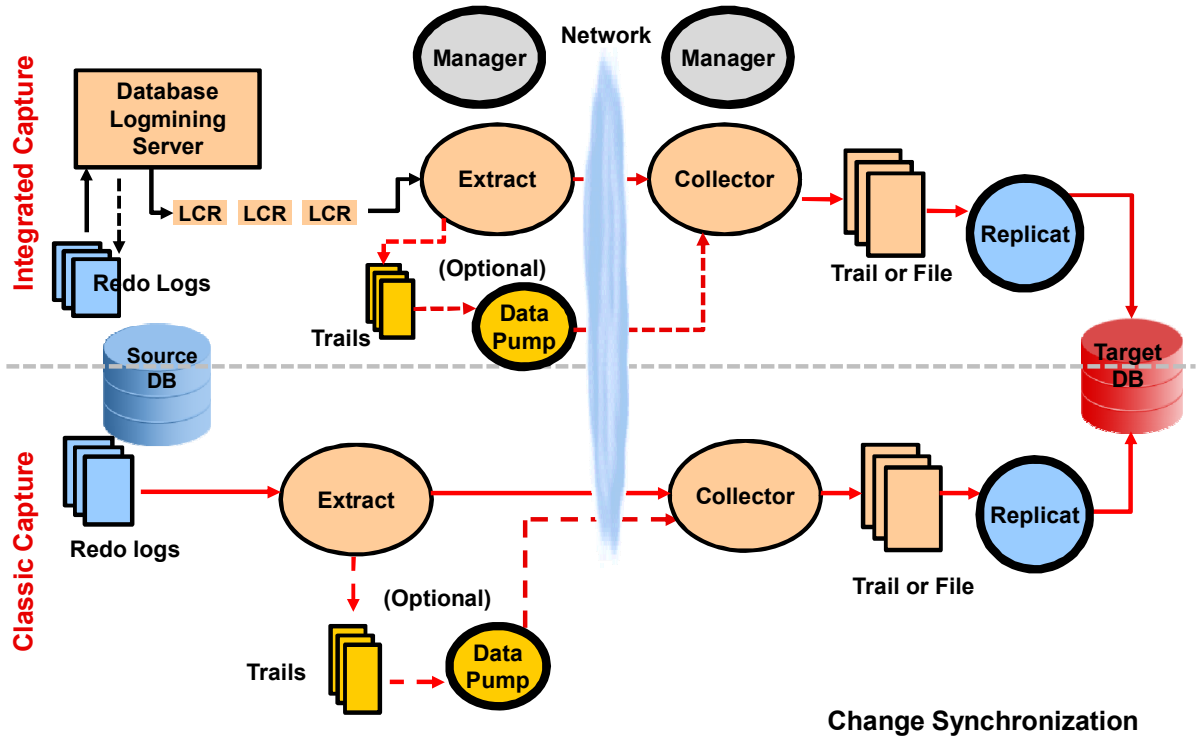
Extract: Overview

- Extract captures all the changes that are made to objects that you configure for synchronization.
- When a transaction is committed, Extract sends the data for that transaction to the trail for propagation to the target system.
- A primary Extract can be configured to:
  - Capture changed data from the Redo logs, the archive logs, or the Redo logs and logmining server (Integrated Capture)
  - Capture changed data from JMS, files, and web services
  - Send the data to be written to a local or remote trail or file
- A Secondary Extract, called a *Data Pump*, can be configured to distribute data from local trails to remote systems. Using Data Pumps is strongly recommended.

ORACLE

Extract captures all the changes that are made to objects that you configure for synchronization. Extract stores the changes until it receives commit records or rollbacks for the transactions that contain them. When a rollback is received, Extract discards the data for that transaction. When a commit is received, Extract sends the data for that transaction to the trail for propagation to the target system. All the log records for a transaction are written to the trail as a sequentially organized transaction unit. This design ensures both speed and data integrity.

Oracle GoldenGate: Logical Architecture



ORACLE

The diagram in the slide illustrates the logical architecture of Oracle GoldenGate for Integrated Capture (top part of the diagram) and for Classic Capture (bottom part of the diagram).

This is the basic configuration. Integrated Capture can be also deployed in “downstream mode,” where an additional computer is used as a standby collector of Redo logs (or archive Redo logs) to offload replication overhead to a computer that is not involved with production online transaction processing (OLTP).

In Oracle GoldenGate version 11.2.1.0.0, all network calls use IPv6-based APIs, but the software is able to fall back to IPv4 if IPv6 is not implemented.

Quiz

A Secondary Extract (or Data Pump Extract) is:

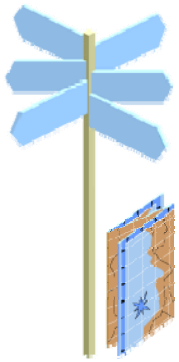
a. Optional but not recommended
b. Mandatory
c. Optional but strongly recommended
d. Included in the best practices for Oracle GoldenGate replication
e. Detrimental for performance

Answer: c, d

Using a Data Pump Extract is strongly recommended, because it provides resilience against network failures with minimal (almost negligible) impact on performance.

Roadmap

- Role and function of an Oracle GoldenGate Extract process
- Classic Capture versus Integrated Capture
- Oracle GoldenGate Integrated Capture features and functionality
- Supported Integrated Capture features according to the RDBMS releases
- Oracle GoldenGate Integrated Apply features and functionality



Classic Capture

- A Classic Capture Primary Extract scans the Redo log files (or the archive log files) looking for all committed transactions.
- Alternatively, a Classic Capture Primary Extract reads the shipped Redo logs when it is run in a standby system.
- Most Oracle data types are read directly from the Redo log files and transmitted to the trail for propagation to the target system. Exceptions are:
  - LOB modifications performed via `DBMS_LOB`
  - XML stored as binary
  - Abstract data types, `VARRAY`s, and nested tables. These data types are fetched (thus incurring a performance penalty) from the source table rather than from the Redo log.

ORACLE

Classic Capture: Limitations

- Classic Capture has some limitations, which preclude its usage for certain data types, such as:
  - Index-Organized Table (IOT) with a mapping table
  - Direct load inserts to IOT tables (cannot be sorted)
  - XML stored as object-relational or XMLType tables
  - LONG VARCHAR and invisible columns
- The most severe limitations, however, are the ones that affect compression. Classic Capture does not support:
  - Basic compression
  - OLTP compression
  - Exadata Hybrid Columnar Compression
- In addition, Classic Capture does not support parallel DML on RAC databases.

Integrated Capture

- The new capture system that is introduced with Oracle GoldenGate release 11.2 is called “integrated” because it operates seamlessly with other RDBMS facilities (such as Logminer), and is able to understand internal formats such as the Logical Change Record (LCR), which is also used by Streams.
- Because the Redo log files are accessed through the logmining server, Integrated Capture can switch automatically between different copies of archive logs or different mirrored versions of the online logs.

RMAN automatically retains the archive logs that are needed by the Integrated Extract. With Classic Capture, there is a disadvantage of managing the required archive logs.

Integrated Capture: Benefits

- Integrated Capture removes most of the limitations of Classic Capture.
  - Basic compression (in addition to OLTP compression) is supported.
  - Exadata Hybrid Columnar Compression is supported.
  - IOTs are supported without limitations.
  - XML stored as binary is captured from redo, whereas XML stored as object-relational is supported and captured from redo.
  - Parallel DML on RAC databases is now supported.
- Integrated Capture improves table filtering efficiency.
- Point-in-time recovery and RAC integration are also handled more efficiently.

Integrated Capture Extract can process through resetlogs smoothly, whereas Classic Extract may need some manual work to achieve the same. For example, an Integrated Extract can capture changes from a previous state of your source database automatically, whereas a Classic Extract will need to be placed in archive log only mode (ALO mode) to capture changes from a previous state of your source database. An Integrated Extract is able to process through a database incomplete recovery more smoothly than a Classic Extract.

For details on how an Integrated Extract capture changes with different states of the source database, see “Can GoldenGate Extract process through Oracle database resetlogs? (Doc ID 1633056.1)” on http://support.oracle.com.

Integrated Capture: Limitations

Integrated Capture also has a few limitations.

- SECUREFILE LOBs are captured from Redo logs only when the LOBs are not compressed, encrypted, or de-duplicated, and when they are stored out-of-row. Otherwise, they are captured from the source table.
- The following data types are *not* supported:
  - `ANYDATASET`
  - `ANYTYPE`
  - `BFILE`
  - `MLSLABEL`
  - `ORDDICOM`
  - `ROWID`
  - `TIMEZONE_ABBR`
  - `URITYPE`
  - `UROWID`

Integrated Capture and Classic Capture: Working Together

- Integrated Capture and Classic Capture are not mutually exclusive.
- On the same system, a set of tables can be replicated by using Classic Capture, whereas a different set can use Integrated Capture for replication.
- This arrangement ensures a gradual upgrade path toward Integrated Capture, which is now the preferred mode for replication.
- Established Oracle GoldenGate installations can continue using Classic Capture, while gradually introducing Integrated Capture to take advantage of compression and the ability to replicate data types that are not supported by Classic Capture.

Integrated Versus Classic Capture: Summary

- Integrated Capture overcomes most of the limitations of Classic Capture related to compression and data type.
- Future enhancements to the product will be included only in Integrated Capture. Current GoldenGate companies that replicate Oracle RDBMS instances should, therefore, start planning for an upgrade to Integrated Capture.
- Classic Capture, however, remains the only alternative for non-Oracle RDBMS vendors or heterogeneous environments where Oracle database is the replication target.
- Classic Capture will be enhanced for non-Oracle platforms to ensure heterogeneous replication, including the new features provided by database vendors.

Future enhancements to the product will be included only in Integrated Capture. For example, Integrated Capture is required to capture changes from a source container database (CDB). You cannot use Classic Capture if your source database is using the multi-tenant option.

New Parameter: ENABLE_GOLDENGATE_REPLICATION

To better track the Oracle GoldenGate product license, a new database initialization parameter, **ENABLE_GOLDENGATE_REPLICATION**, has been introduced in Oracle Database 11.2.0.4 and 12.1.0.2 and later.

- For some of the Oracle GoldenGate functionality to work, this parameter must be set to `true`. It enables access to Transparent Data Encryption (TDE) integration with Classic Extract, Integrated Capture, Integrated Apply, the DBLOGREADER functionality, the Replicat functionality such as suppression of triggers and deferring referential constraints, and other integration points.
- It is a dynamic parameter (no need to bounce the instance).

```
ALTER SYSTEM SET ENABLE_GOLDENGATE_REPLICATION = TRUE
SCOPE=BOTH;
```

ORACLE

For a detailed explanation of the new `ENABLE_GOLDENGATE_REPLICATION` parameter, refer to Doc ID 1568417.1.

If the parameter is not set to `TRUE`, Oracle error ORA-26947 (“Oracle GoldenGate replication is not enabled.”) is likely to be issued when you attempt replication by using Oracle GoldenGate functionality.

Quiz

Integrated Capture supplants Classic Capture, which will no longer be supported in the future.

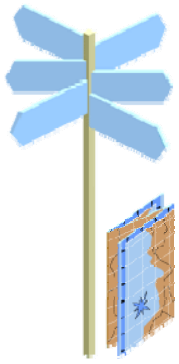
a. True
b. False

Answer: b

Classic Capture is the only option for non-Oracle databases. For Oracle database, Classic Capture will still be supported—but not enhanced—in the future releases of Oracle GoldenGate.

Roadmap

- Role and function of an Oracle GoldenGate Extract process
- Classic Capture versus Integrated Capture
- Oracle GoldenGate Integrated Capture features and functionality
- Supported Integrated Capture features according to the RDBMS releases
- Oracle GoldenGate Integrated Apply features and functionality



Integrated Capture Deployment Modes

- Integrated Capture requires Oracle RDBMS release 11.2.0.3 and later.
- Integrated Capture is available in two modes:
  - Local deployment
  - Downstream deployment
- Local deployment means that the source database and the logmining database are the same (that is, running in the same instance), which must be at least release 11.2.0.3.
- Downstream deployment means that the source database and the logmining database are *different* databases. The logmining server is created in the downstream instance. The Redo logs of the source database are continuously shipped to the downstream instance.

ORACLE

- **Local deployment:** The source database and the mining database are the same in this mode. The source database is the database for which you want to mine the redo stream to capture changes, and also the database 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 deployment:** The source and mining databases are different databases of the same platform. You create the logmining server in the downstream database. You configure redo transport in the source database to ship the Redo logs to the downstream mining database for capture at that location. Consider using a downstream mining server if you must offload overhead (for capture, transformations, and other processing) from the production server. Remember that this requires log shipping and other configuration.

Integrated Capture: Downstream Deployment

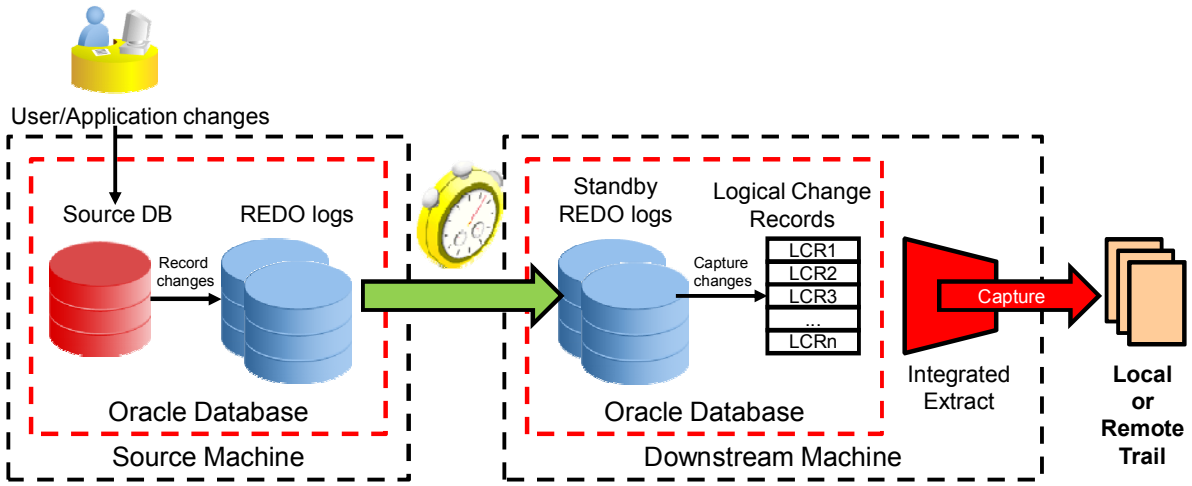
- It is important to note that the Oracle database on the downstream machine is used to hold minimal data or metadata that is specific to the Oracle internal log processing module.
- The standby Redo logs (or archive Redo logs) are *not* applied to the downstream database, but are processed by the Oracle internal log processing module and made available as Logical Change Records (LCRs) to the consumer thread of the Integrated Extract process.
- The latter is responsible for writing the processed data into the Oracle GoldenGate trail file.

The downstream logmining database can be configured to be small in size because data from the replication source is not stored (not even temporarily) on the downstream database.

An Oracle instance is needed only to run the internal log processing module to create the LCR entries that are published to the Integrated Extract process.

Real-Time Downstream Deployment

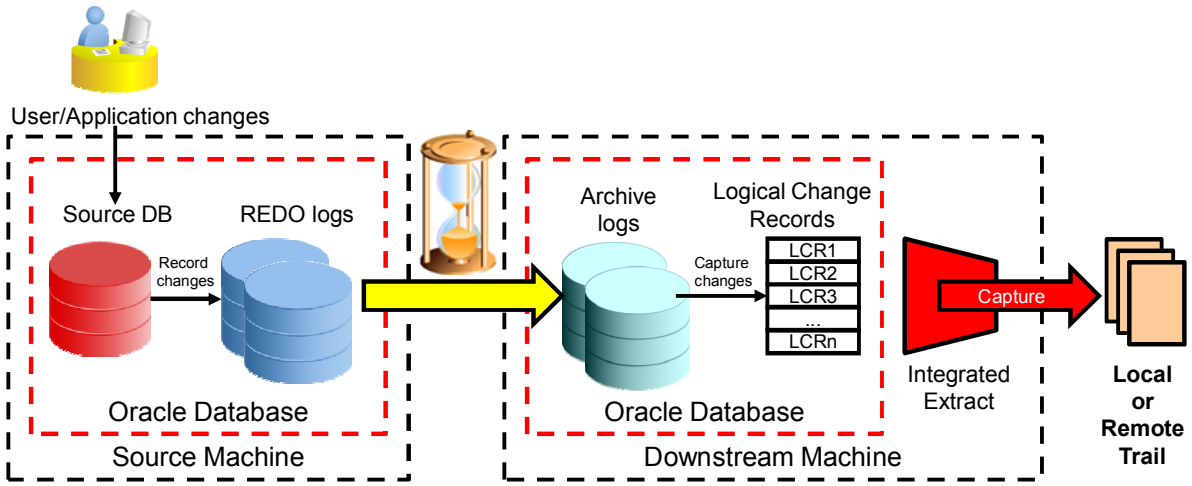
In this deployment configuration, Redo logs are *shipped continuous*ly to the downstream machine as "standby Redo logs."



Change data records from the source database are not persisted in the downstream database. As you learned earlier, the main purpose of the downstream database is to hold state-specific data or metadata that is minimal in nature and specific to the log processing module. Change data records from the Redo logs of the source database are transported or shipped continuously as standby Redo logs on the downstream machine.

Deferred Downstream Deployment

When replication SLAs enable a longer lag time between the source and target databases, and when synchronization can be deferred, archive redo files are shipped at regular intervals to the downstream mining database.



If the service-level agreements that govern replication permit deferred synchronization between the source and the target databases, the Redo logs that are generated at the source machine will be made available as archive logs in the downstream machine. This is similar to how you offload archive log processing by using Classic Extract.

Benefits of this configuration:

- Production databases that are performance-sensitive do not suffer from replication interference. No additional software is required to be installed on the production machine.
- Using a downstream mining database for data capture may be desirable where it is required to offload the capture overhead (and any other overhead) from transformation or other processing from the production server.
- Real-time capture can be used only when there is only *one* source database. If the downstream database must support more than one source database, the only option is to ship archive Redo logs.

Downstream Deployment: Advantages

- A downstream mining server offloads the capture and transformation process overhead from the production source database, thus improving replication performance.
- Downstream deployment is the *only* option for Integrated Capture from Oracle RDBMS instances that are earlier than 11.2.0.3.
- If it is used with a downstream mining server, the release of the source database must be at least 10.2.0.0.
- Downstream deployment offloads data capture and transformation, but it requires log shipping and other configuration changes.

Integrated Capture on Oracle instances earlier than 11.2.0.3 is possible only through a downstream deployment. The downstream database must be at least 11.2.0.3, and the earliest Oracle release that is supported for Integrated Capture is 10.2.

Quiz

When Integrated Capture is used with a downstream deployment option, the Redo logs (or the archive Redo logs) are applied to the downstream database.

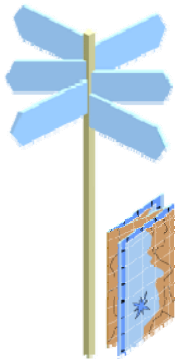
a. True
b. False

Answer: b

The Redo logs (or archive Redo logs) are shipped from the source database to the downstream computer, but they are not applied to the downstream database. They are processed by the Oracle internal log processing module. The data contained in the Redo logs is initially transformed into Logical Change Records (LCR) and subsequently transmitted to the Integrated Extract consumer, which checkpoints the data into the Oracle GoldenGate trail file.

Roadmap

- Role and function of an Oracle GoldenGate Extract process
- Classic Capture versus Integrated Capture
- Oracle GoldenGate Integrated Capture features and functionality
- Supported Integrated Capture features according to the RDBMS releases
- Oracle GoldenGate Integrated Apply features and functionality



Supported Features Based on Source DB Version

DB Version	12c	11.2.0.3+	11.2.0.2	11.1.0.7	10.2
Integrated Apply	Y	N	N	N	N
Large VARCHAR2	Y	N	N	N	N
Multitenant	Y	N	N	N	N
XML Types	Y	Y	N	N	N
Redo-based Secure Files	Y	Y	N	N	N
Fetch	Y	Y	Y	N	N
XA-RAC	Y	Y	Y	N	N
Compression	Y	Y	Y	N	N
TDE/TSE	Y	Y	Y	Y	N

Earlier releases have fewer features available for replication.

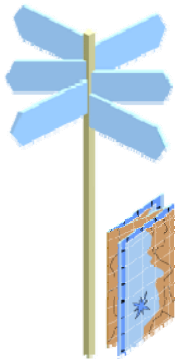
Source Oracle Database version:

- **12*c*:** All new features are supported, in particular, Integrated Apply, large VARCHAR2, and multitenant container database with pluggable databases.
- **11.2.0.3:** You lose the 12*c* features. This version of the database supports Exadata Hybrid Columnar Compression (EHCC) and other compression types such as OLTP and segment compression.
- **11.2.0.2:** You lose support for XML Binary and XML Object Relational (XML-OR), and support for Redo-based secure files.
- **11.1.0.7:** There is support only for Transparent Data Encryption (TDE) and Tablespace Encryption (TSE).
- **10.2:** There are no new features such as XMLTypes, Fetch support for secure files, ADTs, VARRAYS, Nested Tables, Object Tables, Compression, XA transactions, Capture support for Redo-based secure files, and TDE or TSE.

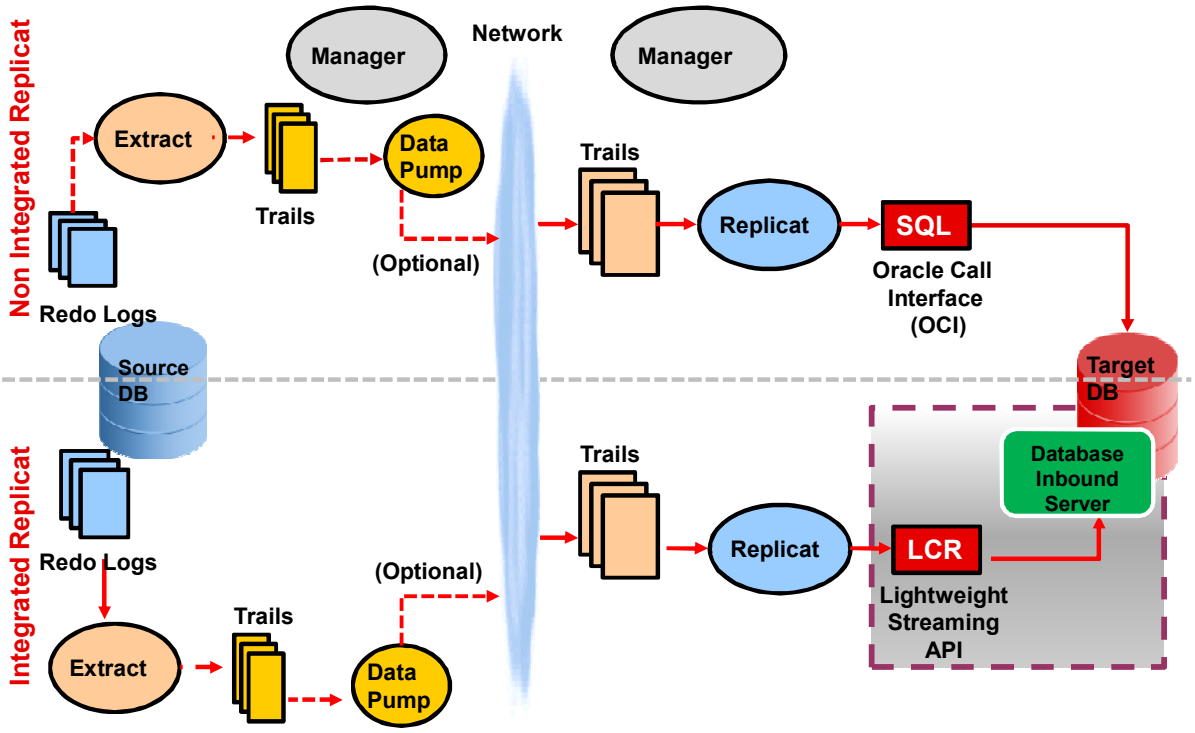
All source configurations presume a downstream Oracle Database version of 11.2.0.3 or later. The target database can be anything.

Roadmap

- Role and function of an Oracle GoldenGate Extract process
- Classic Capture versus Integrated Capture
- Oracle GoldenGate Integrated Capture features and functionality
- Supported Integrated Capture features according to the RDBMS releases
- Oracle GoldenGate Integrated Apply features and functionality



Integrated Replicat/Apply Architecture



The diagram in the slide shows how the new Integrated Replicat architecture (bottom half of the slide) differs from the Classic Replicat architecture (top half of the slide).

The major difference is that Classic Replicat uses the Oracle Call Interface (OCI) to store rows in the target database, whereas Integrated Replicat uses a new lightweight streaming API, which understands data in the LCR format.

Integrated Replicat: How It Works

Integrated Replicat (also knows as Integrated Apply or Integrated Delivery) provides the following features:

- Leverages database parallel apply servers via the inbound server for automatic, dependency-aware parallel apply
- Uses a streaming protocol with asynchronous processing
- Applies source transactions in parallel based on dependencies between transactions
- Has a single Replicat, so **`@RANGE`** or **`THREAD`** or other manual partitioning is not necessary
- Detects and resolves conflicts by using **`CDR`**, **`REPERROR`**, and **`HANDLECOLLISIONS`**

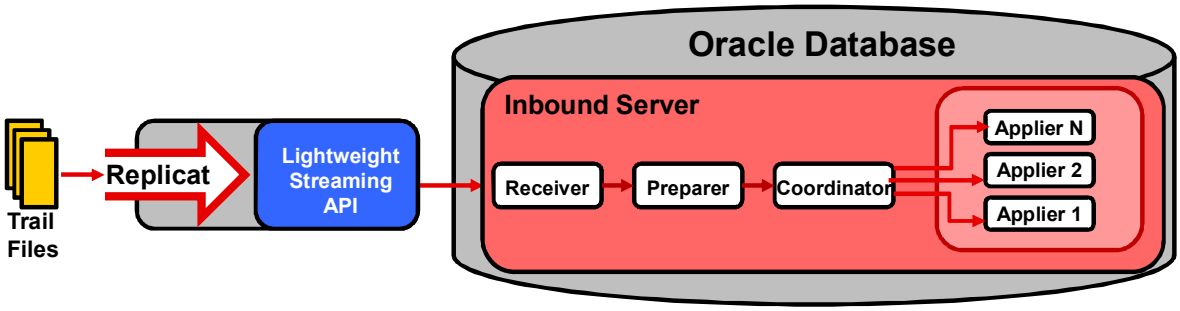
ORACLE

Integrated Delivery or Integrated Replicat uses the streaming protocol to avoid roundtrip times for applying SQL.

Key column constraints (Primary Key, Unique Index, and Foreign Key) can be used to help model the dependencies in the database. Oracle GoldenGate conflict detection and resolution, `REPERROR` handling, and `HANDLECOLLISIONS` can be configured in the Replicat parameter file, similar to a Classic Replicat, although the inbound server performs these tasks.

You can use the CDR parameters in the Replicat parameter file to perform conflict detection and resolution by using the inbound server, which also includes features for conflict resolution and error handling.

Integrated Delivery in Detail



Replicat

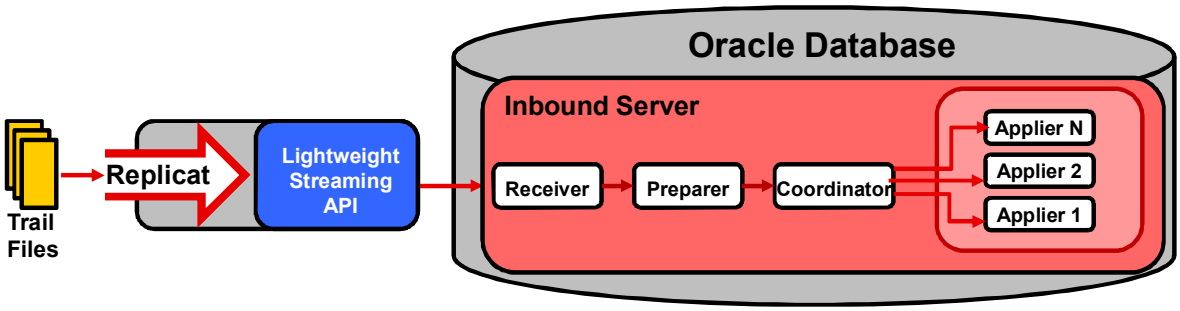
- Reads the trail file
- Constructs LCRs
- Transmits LCRs to Oracle Database via the lightweight streaming API

Inbound Server (Database Apply Process)

- **Receiver:** Reads the LCRs
- **Preparer:** Computes transaction dependencies
- **Coordinator:** Coordinates transactions and maintains the order between applier processes
- **Applier:** Performs changes for assigned transactions, including conflict detection and error handling

- The receiver is responsible for grouping transactions and sorting them in order of dependency.
- The coordinator coordinates the transactions to be applied (and maintains the applier processes in the correct order).
- The appliers perform changes for assigned transactions, including conflict detection and error handling.
- The Oracle database maintains statistics for each applier (apply server) process, including table statistics.

Streaming Protocol: Asynchronous Processing



- Uses one open connection to deliver multiple transactions
- Uses time-based asynchronous ACK to minimize roundtrips
- Uses separate processes for receiving and executing transactions
- Immediately processes each execution
- Handles mixed workloads (inserts, updates, and deletes) across multiple tables and schemas without roundtrips

Summary

In this lesson, you should have learned about:

- The role and function of an Oracle GoldenGate Extract process
- Integrated Capture and Classic Capture, their differences, and their characteristics
- Integrated Capture deployment modes and the advantages of a downstream deployment
- Integrated Capture features that are available depending on the release of the source RDBMS

Practice 2: Overview

The practices for this lesson cover configuring and deploying Integrated Capture and Integrated Apply on a pluggable database that is hosted in a multitenant container database.

Only Integrated Capture is supported when replicating data from a pluggable database. In such a configuration, Classic Capture does not work.

3

Integrated Capture Deployment and Required Components

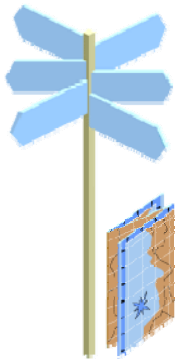
Objectives

After completing this lesson, you should be able to:

- Configure and deploy Oracle GoldenGate Integrated Capture by using the local deployment option
- Prepare a source database and a downstream mining database for real-time capture
- Configure Oracle GoldenGate Integrated Capture by using the downstream deployment option
- Deploy Integrated Capture in downstream mode

Roadmap

- Configuring and deploying Oracle GoldenGate Integrated Capture by using a local deployment option
- Configuring Oracle GoldenGate Integrated Capture by using a downstream deployment option
- Preparing a downstream mining database for real-time capture
- Deploying Integrated Capture in downstream mode



Local Deployment: New Syntax

In a local deployment, the logmining database is the same as the source database.

- The database release must be at least 11.2.0.3.
- The database must be running in archivelog mode.
  - Each Extract group process is registered with the logmining database by using the GGSCI **REGISTER** command.
- The Extract group parameter set has been extended to include **TRANLOGOPTIONS INTEGRATEDPARAMS** to enable Integrated Capture.
  - The **ADD EXTRACT** GGSCI command now supports the **INTEGRATED TRANLOG** clause to configure an Integrated Capture Extract group.

ORACLE

In INTEGRATED TRANLOG mode, Extract integrates with the database logmining server, which passes logical change records (LCR) directly to Extract.

Extract does not read the Redo log. Before using INTEGRATED TRANLOG, use the `DBLOGIN` or `MININGDBLOGIN` command to log in to the source database or the downstream mining database, and then use the `REGISTER EXTRACT` command to register the Extract with that database.

The `INTEGRATEDPARAMS` parameter for Extract passes several parameters to the Oracle Database logmining server when Extract is in Integrated Capture mode.

Local Deployment: Required Steps

To create a new Integrated Extract, perform the following steps:

1. Register the Extract group while in GGSCI, after connecting to the source database by using **DBLOGIN**.
2. In the parameter file for the Extract group, specify **TRANLOGOPTIONS INTEGRATEDPARAMS**.
3. Add the new Extract group by using the **INTEGRATED TRANLOG** clause.

INTEGRATEDPARAMS Options

INTEGRATEDPARAMS supports three options:

- max_sga_size
- parallelism
- downstream_real_time_mine

Example:

```
TRANLOGOPTIONS INTEGRATEDPARAMS (max_sga_size 200,
parallelism 3)
```


- **max_sga_size:** Is a positive integer that specifies (in megabytes) the amount of SGA memory that is used by the database logmining server. The default is 1 GB if the **streams_pool_size** parameter is greater than 1 GB; otherwise, it is 75% of **streams_pool_size**.
- **parallelism:** Is a positive integer that specifies the number of processes of the logmining server
- **downstream_real_time_mine:** Specifies whether Integrated Capture mines a downstream mining database in real-time mode. If it is set to **Y**, the logmining server uses Redo logs. If it is set to **N**, the logmining server uses the archive logs.

The shared memory that is used by the logmining server comes from the Streams pool portion of the System Global Area (SGA) in the database. Therefore, you must set **streams_pool_size** high enough to keep enough memory available for the number of Extract processes that you expect to run in Integrated Capture mode against a given database instance.

By default, one Integrated Capture Extract requests the logmining server to run with **max_sga_size** of 1 GB and **parallelism** of 2. Thus, if you are running three Extracts in Integrated Capture mode 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.

Local Deployment: Example

Start **GGSCI** and connect to the database by using **DBLOGIN**.

```
GGSCI> DBLOGIN USERID ggs_src, PASSWORD <password>
Successfully logged into database.
```

Register the new Extract group with the mining database.

```
GGSCI> REGISTER EXTRACT ehra DATABASE
2012-05-19 12:28:40 INFO OGG-02003  Extract EHRA
successfully registered with database at SCN 2029767.
```

In the parameter file, set values for **INTEGRATEDPARAMS**.

```
TRANLOGOPTIONS INTEGRATEDPARAMS (max_sga_size 200, &
parallelism 3)
```

In **GGSCI**, add the new Extract group.

```
GGSCI> ADD EXTRACT ehra, INTEGRATED TRANLOG, BEGIN NOW
```


It is a best practice not to end the name of Extract, Pumps, and Replicat groups with a number. The reason is that Oracle GoldenGate internally appends numbers to the Extract or Replicat names. To avoid problems that could occur while performing low-level troubleshooting, always end the Extract and Replicat names with letters.

With Release 12.1.2.1.0, an Integrated Extract for Oracle database now supports multiple extracts by using a shared LogMiner dictionary with the introduction of the `SHARE` option for `REGISTER EXTRACT`. Use of this option returns to an existing LogMiner data dictionary build with a specified SCN creating a clone. This allows for faster creation captures by leveraging existing dictionary builds.

The recommended way to provide database credentials to `DBLOGIN` is to use a wallet and its associated credential store. Providing a password in clear is still supported, but discouraged.

Integrated Capture: Upgrade from Classic

- Already existing Classic Capture groups can be upgraded to Integrated Capture with little effort:
  - If the Classic Extract group is running, it must be stopped.
  - In GGSCI, use **`DBLOGIN`** to connect to the logmining server.
- Register the Extract group that is being upgraded with the logmining server.
- Use **`ALTER EXTRACT <group name>, UPGRADE INTEGRATED TRANLOG`** to upgrade to Integrated Capture.
  - From this point onward, the Extract runs in Integrated Capture mode, enabling new data types and compression on the source database.

Integrated Capture Upgrade: Example

You are upgrading the Classic Capture Extract group **ehrb**:

```
GGSCI> STOP EXTRACT ehrb
Sending STOP request to EXTRACT EHRB...
Request processed.
```

Log in to the mining server to register the Extract group:

```
GGSCI> DBLOGIN USERID user01, PASSWORD <password>
Successfully logged into database.
GGSCI> register extract ehrb database
Extract EHRB successfully registered with database at SCN
229766
```

Alter the Extract **ehrb** to upgrade it to Integrated Capture:

```
GGSCI> ALTER EXTRACT ehrb, UPGRADE INTEGRATED TRANLOG
Extract EHRB successfully upgraded to integrated capture.
```


It is also possible to downgrade an Integrated Extract to Classic capture by using the DOWNGRADE keyword:

```
alter extract <group name>, DOWNGRADE INTEGRATED TRANLOG
```

Quiz

Already-existing Extract groups that are defined for Classic Capture must be dropped and re-created if you want to take advantage of the new Integrated Capture features.

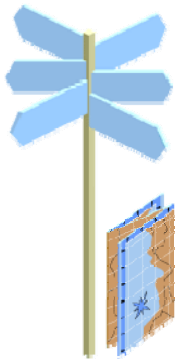
a. True
b. False

Answer: b

It is possible to upgrade already-existing Classic Capture groups by using the `UPGRADE INTEGRATED TRANLOG` syntax.

Roadmap

- Configuring and deploying Oracle GoldenGate Integrated Capture by using a local deployment option
- Configuring Oracle GoldenGate Integrated Capture by using a downstream deployment option
- Preparing a downstream mining database for real-time capture
- Deploying Integrated Capture in downstream mode



ORACLE

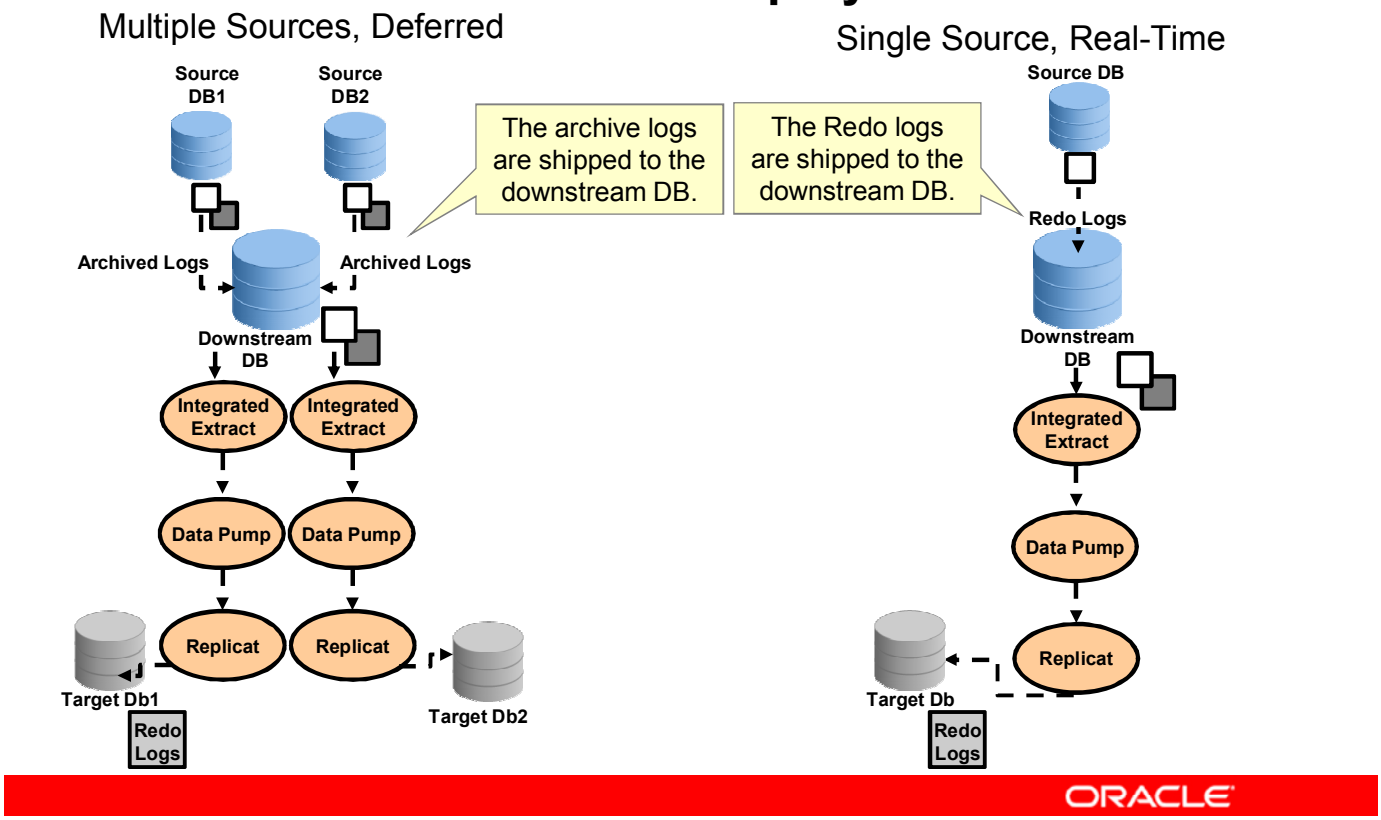
Downstream Deployment: One or Multiple Sources

- Downstream deployment is available with two options:
  - Real-time (operating on Redo logs)
  - Deferred (operating on archive logs)
- A real-time deployment can support only one source database.
- Multiple sources can use one downstream mining database if they ship their archive logs.

To employ real-time mining, you can use the downstream mining database to capture changes only from a single source database. You need to create standby Redo logs so that the source database can use Oracle Data Guard redo transport to send redo to the downstream mining database as it is written to the online Redo log at the source.

To use the downstream mining database to capture changes from multiple source databases, you must make sure that no standby Redo logs are present in the downstream mining database. All the Extract sessions that operate in Integrated Capture mode will capture changes from the archive logs that are sent from the various source databases to the downstream mining database.

Real-Time Versus Deferred Downstream Deployment

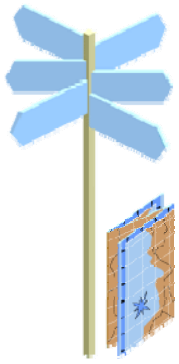


The diagram in the left section of the slide shows a deployment configuration where multiple source databases ship their archive log files to the downstream database.

The diagram in the right section of the slide shows a deployment configuration where a single source ships its Redo log files to the downstream database. In this configuration, only one source DB is supported.

Roadmap

- Configuring and deploying Oracle GoldenGate Integrated Capture by using a local deployment option
- Configuring Oracle GoldenGate Integrated Capture by using a downstream deployment option
- Preparing a downstream mining database for real-time capture
- Deploying Integrated Capture in downstream mode



Downstream Mining Server

- The downstream mining server must run in archivelog mode, and must be configured to archive the Redo (or archive) logs received from the source database.
  - The source database must be configured to send the Redo (or archive) logs to the mining database.
- Redo (or archive) log shipping is accomplished by Oracle Net Services and configured by using the **LOG_ARCHIVE_DEST_n** instance parameters.
  - Integrated Capture is set up on the host computer where the downstream database is installed. An Oracle GoldenGate instance must therefore be installed and activated on the downstream node.

If configured, each LOG_ARCHIVE_DEST_1 through LOG_ARCHIVE_DEST_10 destination must contain either a LOCATION or SERVICE attribute to specify a local disk directory or a remotely accessed database, respectively.

A redo transport destination is configured by setting the LOG_ARCHIVE_DEST_n parameter to a character string that includes one or more attributes.

The SERVICE attribute, which is a mandatory attribute for a redo transport destination, must be the first attribute specified in the attribute list. The SERVICE attribute is used to specify the Oracle Net service name that is used to connect to the redo transport destination. The service name must be resolvable through an Oracle Net naming method to an Oracle Net connect descriptor that matches the Oracle Net listeners at the redo transport destination.

Redo data can be transmitted to the downstream mining server either synchronously or asynchronously.

The SYNC attribute is used to specify that synchronous redo transport mode should be used to send redo data to a redo transport destination.

The ASYNC attribute is used to specify that asynchronous redo transport mode should be used to send redo data to a redo transport destination. Asynchronous redo transport mode is used if neither the SYNC nor the ASYNC attribute is specified.

Quiz

In a downstream deployment, Oracle GoldenGate should be installed on the source database and on the downstream node.

a. True
b. False

Answer: b

Integrated Capture is configured on the downstream node. The source database needs to only ship the redo (or archived) logs to the downstream server, which is accomplished by using the RDBMS native software (Oracle Net Services). One or more `LOG_ARCHIVE_DEST_n` destinations can be configured to ship the logs to the downstream server. No Oracle GoldenGate component is required to run on the source database. The Oracle GoldenGate instance is installed and runs on the downstream node.

Users and Privileges Required for Downstream Deployment

- A specific Oracle GoldenGate user should exist on the source database.
  - The user should be able to access the **V$DATABASE** view.
  - All required privileges should be granted to that user.
- A specific Oracle GoldenGate user should also be defined in the downstream database. This user's credentials are used by the Integrated Extract for synchronization purposes.

On the source database, a dedicated Oracle GoldenGate user should be created with select privileges on the `V$DATABASE` view. The `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure is used to grant this user the required privileges.

A specific user also exists in the downstream database. Integrated Extract uses the credentials of this user to retrieve logical change records from the logmining server at the downstream mining database.

`DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` is run to grant appropriate privileges to this user in the mining database, as well as select privileges on `V_$DATABASE`.

Downstream Node Configuration: Preparing the Mining DB to Archive Its Redo

- The downstream database must be run in archivelog mode.

```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
```

- `log_archive_dest_1` is set to archive local redo.

```
SQL> ALTER SYSTEM SET
     LOG_ARCHIVE_DEST_1='LOCATION=/u02/archivelog
     VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
```

- `log_archive_dest_1` is then enabled.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE;
```

Preparing the Mining Database to Archive Redo Received from the Source Database

- The downstream mining database `log_archive_dest_2` parameter is first set, and then enabled.

```
SQL> ALTER SYSTEM SET
     LOG_ARCHIVE_DEST_2='LOCATION=/u02/archive/srcdb
     VALID_FOR=(STANDBY_LOGFILE,ALL_ROLES)';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

- The `DG_CONFIG` parameter must also be set for the downstream mining database.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_CONFIG=
    'DG_CONFIG=(srcdb,dwnstrdb)';
```

Preparing the Mining Database: Adding the Standby Redo Log Files

The downstream mining database must be configured to create the standby Redo log files.

```
SQL> Alter Database add
     standby logfile group 4 ('/u02/dwnstrdb/redo/
     slog4a.rdo','/u02/dwnstrdb/redo/slog4b.rdo')
     size 60M;
SQL> Alter Database add
     standby logfile group 5 ('/u02/dwnstrdb/redo/
     slog5a.rdo','/u02/dwnstrdb/redo/slog5b.rdo')
     size 60M;
SQL> Alter Database add
     standby logfile group n ('/u02/dwnstrdb/redo/
     slognа.rdo','/u02/dwnstrdb/redo/slognb.rdo')
     size 60M;
```

ORACLE

The final step for configuring the downstream mining database is the creation of the standby Redo log files, which must be at least the same size as the original Redo log files from the source database. In addition, the number of the log file groups must be the number of the Redo log groups on the source database, plus one. So, if the source RDBMS has been configured with three Redo log file groups, the downstream mining database must have four Redo log file groups.

The size of each standby Redo log file must be at least the size of the corresponding Redo log file on the source database.

Preparing the Source Database to Send Redo to the Mining Database

- The `DG_CONFIG` parameter must also be set at the source database.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_CONFIG=
     'DG_CONFIG=(srcdb,dwnstrdb)';
```

- Set up redo transport at the source database.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=
     DWNSTRDB ASYNC OPTIONAL NOREGISTER VALID_FOR=
    (ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=
    dwnstrdb';
```

- Enable the downstream destination.

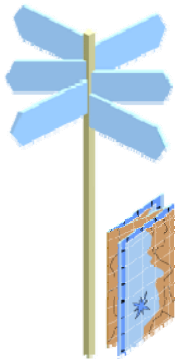
```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

The downstream database setup is now complete. The source database automatically ships the logs, which are read by Integrated Capture and checkpointed to the trail file.

ORACLE

Roadmap

- Configuring and deploying Oracle GoldenGate Integrated Capture by using a local deployment option
- Configuring Oracle GoldenGate Integrated Capture by using a downstream deployment option
- Preparing a downstream mining database for real-time capture
- Deploying Integrated Capture in downstream mode



Deploying Integrated Capture on Downstream Node

- After log shipping has been configured, you can register the Integrated Capture group on the downstream node.

```
GGSCI> DBLOGIN USERID gg_admsrc@srcdb PASSWORD <password>
GGSCI> MININGDBLOGIN USERID gg_dwnstr@dwnstrdb PASSWORD <pwd>
GGSCI> REGISTER EXTRACT ehr3 DATABASE
```

- The Extract group can be created in the downstream mining database.

```
GGSCI> ADD EXTRACT ehr3 INTEGRATED TRANLOG BEGIN NOW
```

- Edit the **ehr3.prm** Extract parameter file. To take advantage of real-time capture, the following lines are required:

```
USERID gg_admsrc@srcdb PASSWORD <password>
TRANLOGOPTIONS MININGUSER gg_dwnstr@dwnstrdb MININGPASSWORD &
 <password>
TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine Y)
```

ORACLE

Summary

In this lesson, you should have learned how to:

- Configure and deploy Oracle GoldenGate Integrated Capture by using the local deployment option
- Prepare a source database and a downstream mining database for real-time capture
- Configure Oracle GoldenGate Integrated Capture by using the downstream deployment option
- Deploy Integrated Capture in downstream mode

Practice 3: Overview

This practice cover the following topics:

- Configuring an Extract group by using local deployment
- Preparing the source and downstream databases for Integrated Capture
- Deploying Integrated Capture by using a downstream real-time mode
- Configuring and deploying Integrated Delivery

4

Oracle GoldenGate with Oracle Real Application Clusters Configuration

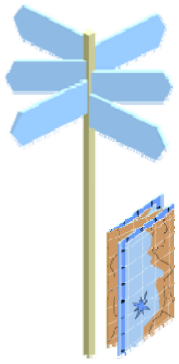
Objectives

After completing this lesson, you should be able to:

- Use Oracle GoldenGate with Oracle RAC to achieve Maximum Availability
- Understand the role played by Oracle Clusterware in ensuring Maximum Availability
- Describe the features and functionality provided by Oracle Database File System (DBFS)
- Follow best practices while setting up the various components of the Maximum Availability architecture

Roadmap

- Oracle GoldenGate + Oracle RAC = Maximum Availability
- Oracle Clusterware as a key component of Maximum Availability
- Oracle Database File System (DBFS)
- Best practices to follow when using Oracle GoldenGate with Oracle RAC



Oracle Real Application Clusters (Oracle RAC)

- Oracle Real Application Clusters (Oracle RAC) enables multiple instances that are linked by an interconnect to share access to an Oracle Database.
- In an Oracle RAC setup, the Oracle Database runs on two or more systems in a cluster, while concurrently accessing a single shared database.
- The result is a single database system that spans multiple hardware systems, enabling Oracle RAC to provide high availability and redundancy during failures in the cluster.
- Oracle RAC is the foundation for data center high availability (HA).

Oracle RAC and Business Continuity

- Oracle RAC is an integral component of Oracle's Maximum Availability Architecture.
- The key advantages of Oracle RAC is the inherent fault tolerance provided by the multiple physical servers that are used by the database.
- Because the servers in the cluster run independently, the failure of one or more servers does not affect the others.
- This architecture also allows a group of servers to be transparently put online or offline, while the rest of the system continues to provide uninterrupted database services.

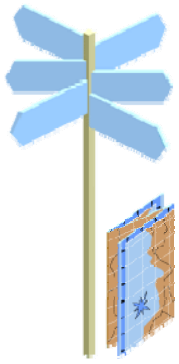
Oracle GoldenGate Used in Combination with Oracle RAC

Oracle GoldenGate is instrumental in achieving Maximum Availability in the following scenarios:

- To migrate to an Oracle Database by incurring minimal down time
- To deploy an application architecture that requires features such as an active-active database and zero or minimal down time during planned outages for system upgrades
- To implement a near real-time data warehouse or consolidated database on Oracle RAC, which is sourced from various, possibly heterogeneous source databases
- To capture data from an OLTP application that is running on Oracle RAC to support further downstream consumption such as a SOA type integration

Roadmap

- Oracle GoldenGate + Oracle RAC = Maximum Availability
- Oracle Clusterware as a key component of Maximum Availability
- Oracle Database File System (DBFS)
- Best Practices to follow when using Oracle GoldenGate with Oracle RAC



Oracle Clusterware

Oracle Clusterware enables servers to communicate with one another, so that they appear to function as a collective unit:

- This combination of servers is commonly known as a cluster.
- Although the servers are stand-alone servers, each server has additional processes that communicate with other servers, appearing as if they are one system to applications and end users.
- Oracle Clusterware provides the infrastructure that is necessary to run Oracle RAC; additionally, it also manages resources such as virtual IP (VIP) addresses, databases, listeners, services, and so on.

There are APIs to register an application and instruct Oracle Clusterware about the way an application is managed in a clustered environment.

One can use the APIs to register the Oracle GoldenGate Manager process as an application that is managed through Oracle Clusterware. The Manager process should then be configured to automatically start or restart other Oracle GoldenGate processes.

Oracle Clusterware Components

- Oracle Clusterware is run by Cluster Ready Services (CRS) consisting of two key components:
  - Voting Disk, which polls for consistent heartbeat information from all the nodes when the cluster is running, and acts as a tiebreaker during communication failures
  - Oracle Cluster Registry (OCR), which records and maintains the cluster and node membership information
- The CRS service has four components, each handling a variety of functions:
  - Cluster Ready Services daemon (CRSd.)
  - Oracle Cluster Synchronization Service Daemon (OCSSd.)
  - Event Volume Manager Daemon (EVMd.)
  - Oracle Process Clusterware Daemon (OPROCd.)

- The CRS daemon manages resources such as starting and stopping of services and failing-over application resources.
- The OCSS daemon provides basic “group services” support, such as synchronization services between nodes and access to the node membership information.
- The EVM daemon spawns a permanent child process called “evmlogger,” and generates events.
- The OPROC daemon provides the server fencing solution for Oracle Clusterware. Its main function is to ensure cluster integrity.

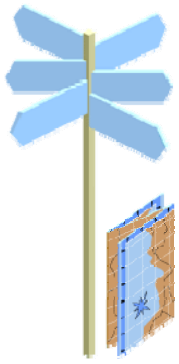
Oracle Clusterware Failover

- Failure or death of the CRS daemons can cause node failure.
- Node failure triggers automatic reboots of the nodes to avoid corruption of data (due to possible failure of communication between the nodes), also known as *fencing*.
  - The Clusterware Failover mechanism ensures the survivability of the cluster when one or more nodes suffer from an outage.
  - Oracle Clusterware automatically attempts to restart the failed component and also redirects operations to a surviving component.

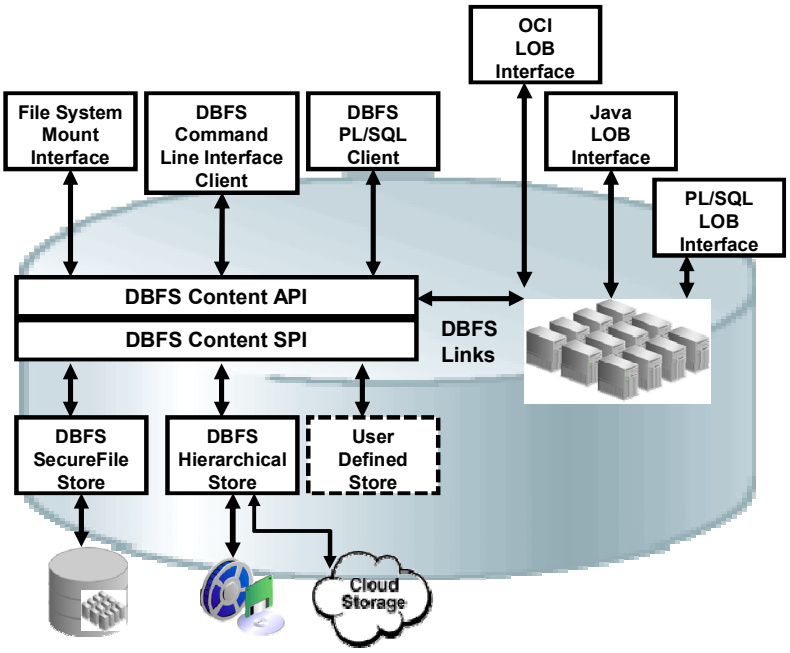
Oracle Clusterware includes a high availability framework for managing any application that runs on clusters. Oracle Clusterware manages applications to ensure that they start when the system starts. Oracle Clusterware also monitors the applications to make sure that they are always available. For example, if an application process fails, Oracle Clusterware attempts to restart the process based on scripts that interact with the Oracle Clusterware services through APIs.

Roadmap

- Oracle GoldenGate + Oracle RAC = Maximum Availability
- Oracle Clusterware as a key component of Maximum Availability
- Oracle Database File System (DBFS)
- Best practices to follow when using Oracle GoldenGate with Oracle RAC



Oracle Database File System (DBFS)



The Oracle Database File System (DBFS) leverages the features of a database to store files, and the strengths of a database in efficiently managing relational data, to implement a standard file system interface for the files stored in the database.

With this interface, storing files in the database is no longer limited to programs that are specifically written to use the BLOB and CLOB programmatic interfaces.

With DBFS, files in the database can be transparently accessed by using any operating system program that acts on files.

Oracle DBFS Client/Server

- The Oracle Database File System (DBFS) creates a standard file system interface on top of the files and directories that are stored in database tables.
- DBFS is similar to NFS in that it provides a shared network file system that looks like a local file system.
- Like NFS, there is a server component and a client component.
- In DBFS, the server is the Oracle Database:
  - Files are stored as SecureFiles LOBs in a database table.
  - A set of PL/SQL procedures implements the file system access primitives such as create, open, read, write, and list directory.

Oracle DBFS Content Store

- The implementation of the file system in the database is called the DBFS Content Store.
  - The DBFS Content Store allows each database user to create one or more file systems that can be mounted by clients.
  - Each file system has its own dedicated tables that hold the file system content.
- DBFS also has a client component that runs on each file system client machine called **`dbfs_client`**.
  - **`dbfs_client`** provides a command-line interface to allow files to be easily copied to and from the database from any host on the network.
  - It implements simple file system commands such as `list` and `copy`, which are similar to the shell utilities **`ls`** and **`rcp`**.

The command-line interface that is implemented by `DBFS_CLIENT` creates a direct connection to the database without requiring an OS mount of the DBFS.

The files in the DBFS Content Store can also be directly accessed by the database applications through the PL/SQL interface. The PL/SQL interface allows database transactions and read consistency to span relational and file data.

DBFS can migrate SecureFiles from existing tables to other storage by using DBFS Links.

Oracle DBFS Enhancements in Oracle 12*c*

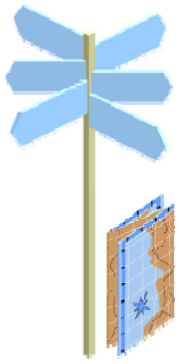
- The Oracle DBFS functionality was introduced in Oracle 11*g*.
  - Real file-system-style access was available only on the Linux platform due to the reliance on the FUSE project.
  - All other platforms were limited to using the client tool to access the file system.
- Oracle 12*c* introduces the following access:
  - **`HTTP/HTTPS`**
  - **`FTP`**
  - **`WebDAV`**
    - Access to the DBFS is via the `"/dbfs"` virtual directory in the XML DB repository.
- The HTTP access to the database can be configured to use Digest Authentication.

ORACLE

Digest Authentication provides encryption of user credentials (name, password, and so on) without the overhead of complete data encryption.

Roadmap

- Oracle GoldenGate + Oracle RAC = Maximum Availability
- Oracle Clusterware as a key component of Maximum Availability
- Oracle Database File System (DBFS)
- Best Practices to follow when using Oracle GoldenGate with Oracle RAC



Oracle GoldenGate with Oracle RAC

- It is a best practice to store the Oracle GoldenGate trail files, checkpoint files, bounded recovery, and configuration files in Oracle DBFS.
  - Such arrangement provides the best performance, scalability, recoverability, and failover capabilities in the event of a system failure.
- Using DBFS is fundamental to the continuing availability of the checkpoint and trail files in the event of a node failure.
  - Ensuring the availability of the checkpoint files clusterwide is essential to ensuring that after a failure occurs, the Extract process can continue mining from the last known archive redo log file position, and the Replicat processes can start applying from the same trail file position before a failure occurred.

Oracle GoldenGate and DBFS

- The use of DBFS allows one of the surviving database instances to be the source of an Extract/Data Pump process or the destination for the Replicat processes.
- It is recommended that you run the DBFS database in ARCHIVELOG mode, so that recoverability is not compromised in the event of media failures or corruptions.
- It is further recommended that you create a single file system for storing the Oracle GoldenGate trail files, checkpoint files, bounded recovery files, temp files, discard files, and parameter files.
  - Enough trail file disk space should be allocated to permit storage of up to 12 hours of trail files.

Enough space for 12 hours worth of trail files will give sufficient space for Extract trail file generation if a problem occurs with the target environment that prevents it from receiving new trail files. The amount of space needed for 12 hours can be determined only by testing trail file generation rates with real production data.

Oracle DBFS and Oracle Clusterware

- DBFS can be configured so that the DBFS instance and mount point resources are automatically started by Cluster Ready Services (CRS) after a node failure.
- The **`crsctl`** command-line utility is used to register the DBFS resource with the Cluster Ready Services, so that Oracle Clusterware is aware of the resource, and is able to mount the DBFS file system from a surviving node.
- The following Oracle GoldenGate directories should be placed in the shared DBFS drive:
  - **`dirchk`** (Checkpoint files)
  - **`dirpcs`** (Process status files)
  - **`dirprm`** (Parameter files)
  - **`dirdat`** (Extract data files)

ORACLE

The recommended way to store Oracle GoldenGate files in the DBFS file system is to create symbolic links. For example, if the DBFS file system is mounted on:

```
/mnt/gg_source/goldengate
```

Create the directories under that mount point:

```
$> mkdir /mnt/gg_source/goldengate/dirchk
$> mkdir /mnt/gg_source/goldengate/dirpcs
$> mkdir /mnt/gg_source/goldengate/dirprm
$> mkdir /mnt/gg_source/goldengate/dirdat
$> mkdir /mnt/gg_source/goldengate/BR
ln -s /mnt/gg_source/goldeng
```

And create the symbolic links:

```
ate/dirprm $GG_HOME/dirchk
ln -s /mnt/gg_source/goldengate/dirprm $GG_HOME/dirpcs
ln -s /mnt/gg_source/goldengate/dirprm $GG_HOME/dirprm
ln -s /mnt/gg_source/goldengate/dirprm $GG_HOME/dirdat
ln -s /mnt/gg_source/goldengate/BR $GG_HOME/BR
```

Oracle GoldenGate Bounded Recovery

- The Bounded Recovery (BR) feature was added to Extract in Oracle GoldenGate version 11*g*.
  - This feature guarantees efficient recovery after Extract stops for any reason, whether planned or unplanned, no matter how many open (uncommitted) transactions there are at the time that Extract stops and no matter how old they are.
- The Bounded Recovery checkpoint files should be placed on a shared file system such that in the event of a failover when there are open long-running transactions, Extract can use Bounded Recovery to reduce the time taken to perform recovery.
  - Bounded Recovery files should be placed on DBFS.
  - The Extract parameters **BR** must be used to specify the DBFS location of the Bounded Recovery file.

To set the Bounded Recovery file directory, use the following Extract parameter:

```
BR BRDIR /mnt/gg_source/goldengate/BR
```

Oracle GoldenGate Target Environment

- On the target environment where the Replicat processes read the trail files and apply the data to the target database, there is a requirement for two separate DBFS file systems to separate the different I/O requirements of the **trail** and **checkpoint** files.
  - Trail files are written by the Collection Server process on the target host by using consecutive serial I/Os from the start to the end of the file, sized according to the Data Pump configuration.
  - The same trail files are read by each Replicat process, also using consecutive serial I/O requests.
  - After a portion of the trail is read by a Replicat process, it will not normally be read a second time by the same process.
- The storage option for trail files is **`NOCACHE LOGGING`**.

ORACLE

When you use multiple Replicat processes that read from the same trail files, it is rare that they would remain in sync, especially when they read from the same portion of the trail file at the same time. Because of this, the best configuration for DBFS would be with the `NOCACHE LOGGING` storage options.

`NOCACHE LOGGING` is a database table option. `NOCACHE` is used to specify that LOB values are not brought into the buffer cache. `LOGGING` is used to specify that Oracle Database generates full redo for LOB data pages. `NOLOGGING` is intended to be used when a customer does not care about media recovery. Thus, if the disk, tape, or storage media fails, you cannot recover your changes from the log because the changes were never logged.

Oracle GoldenGate Checkpoint Files

- The checkpoint files are small (approximately 4 KB) but written to frequently, overwriting previous data.
- The file does not grow in size and is read only during process startup to determine the proper starting point for recovery or initiation.
  - Because the checkpoint file is written to over and over, performance is best when the file is stored in DBFS with the **`CACHE LOGGING`** storage option.
  - Setting the **`CACHE`** option causes the small amount of data that is being written to the checkpoint files to be written into the buffer cache of the DBFS instance, without issuing direct writes to disk that causes higher waits on I/O.
- Checkpoint performance increases by a factor of 2 to 5 times when the **`CACHE LOGGING`** configuration is used.

The second DBFS file system is used only for checkpoint files, so it can be sized less than 100 MB.

Performance Best Practices Summary

When using Oracle GoldenGate in an Oracle RAC environment, perform the following:

- Store the Oracle GoldenGate trail files, checkpoint files, and bounded recovery and configuration files in DBFS.
- Using DBFS is fundamental to the continuing availability of the checkpoint and trail files in the event of a node failure.
- Run the DBFS database in `ARCHIVELOG` mode.
- In the replication source environment, use one DBFS file system to store all files.
- In the replication target, create two DBFS file systems:
  - One for trail files (storage option **`NOCACHE LOGGING`**)
  - One for checkpoint files (storage option **`CACHE LOGGING`**)
- Register DBFS with Cluster Ready Services (**`crsctl`**) to make sure that DBFS is mounted by a surviving node.

Summary

In this lesson, you should have learned how to:

- Use Oracle GoldenGate with Oracle RAC to achieve Maximum Availability
- Understand the role played by Oracle Clusterware in ensuring Maximum Availability
- Describe the features and functionality provided by Oracle Database File System (DBFS)
- Follow best practices while setting up the various components of the Maximum Availability architecture

Practice 4: Overview

The practices for this lesson cover the following topics:

- Creating one DBFS file system for the replication source and two DBFS file systems to host the replication target
- Configuring Oracle GoldenGate by using the DBFS file systems
- Using the `crsctl` utility to register the DBFS resource with Oracle Clusterware

5

Oracle GoldenGate Event Marker System

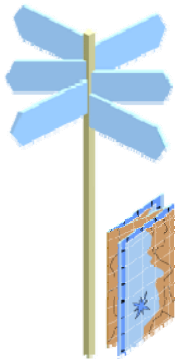
Objectives

After completing this lesson, you should be able to:

- Explain the Oracle GoldenGate Event Marker System
- Identify the type of actions that can be triggered via the Event Marker mechanism
- Combine **`EVENTACTIONS`** options to inject sophisticated extraction and delivery logic into data replication

Roadmap

- Explain the Oracle GoldenGate Event Marker System
- Identify the type of actions that can be triggered via the Event Marker mechanism
- Combine `EVENTACTIONS` options to inject sophisticated extraction and delivery logic into data replication



Event Marker Mechanism

- The Extract and Replicat processes can be used to perform specific actions following the occurrence of certain events in the transaction log or the trail file.
- Event Marker consists of two components:
  - Event Record: Is a specific type of record that satisfies certain selection criteria; is used to trigger actions. The event record is selected by a **FILTER** or **WHERE** clause included in a **TABLE** statement (Extract) or a **MAP** and **TABLE** statement (Replicat).
  - **EVENTACTIONS** clause: Determines what actions must be taken by the Extract or Replicat process. Such actions can be stopping the Extract or Replicat process, producing a statistics report, executing a shell command, and so on.

The Event Marker System is *not* supported for Initial Loads.

The `EVENTACTIONS` parameter must be included as a sub-parameter of the `TABLE` or `MAP` statement.

Normally the `TABLE` statement is applied to Extract processing. There is, however, a form of `TABLE` statement for Replicat, which is similar to that of the Replicat `MAP` statement, except that there is no mapping of the source table in the data record to a target table by means of a `TARGET` clause. `TABLE` for Replicat is solely a means of triggering a non-data action to be taken by Replicat when it encounters an event record.

Because there is no mapping between source and target, the event record must be ignored or discarded by using one of the `EVENTACTIONS`.

`EVENTACTIONS` is not supported if the source database is Teradata and Extract is configured in maximum performance mode.

EVENTACTIONS Options

- **EVENTACTIONS** options can be combined (and usually are).
- A few **EVENTACTIONS** options are incompatible with each other, and some take precedence over others while being evaluated.
- Several actions will occur before the record is written to the trail or applied to the target, whereas other actions will occur after the record has been written.
- It is possible to instruct the Extract or Replicat process to discard the event record, or even to ignore the entire transaction that includes the event record.

The ability to discard the triggering record is crucial when the information carried by the event record is used only as a marker to drive the Extract or Replicat processes, and has no business value.

For example, a specific string could be used to trigger an end-of-day process. That string would be inserted on the source database on a table that is normally used for business purposes.

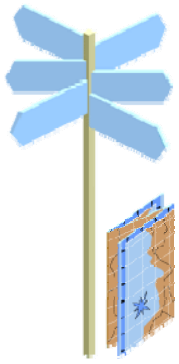
Example

```
MAP src.DEPT, TARGET trg.DEPT, &
  FILTER (@streq(DEPT_NAME = 'END_OF_DAY)=1), &
  EVENTACTIONS (IGNORE, LOG INFO, FORCESTOP);
```

Because the string 'END_OF_DAY' has no business meaning, and acts only as an event record, the entire record should not be applied to the target database. The IGNORE action forces the record not to be applied by Replicat to the target database.

Roadmap

- Explain the Oracle GoldenGate Event Marker System
- Identify the type of actions that can be triggered via the Event Marker mechanism
- Combine `EVENTACTIONS` options to inject sophisticated extraction and delivery logic into data replication



EVENTACTIONS Sequence

- **EVENTACTIONS** that are evaluated *before* the record is written:
  - **TRACE**
  - **LOG**
  - **CHECKPOINT BEFORE**
  - **IGNORE**
  - **DISCARD**
  - **SHELL**
  - **ROLLOVER**
- **EVENTACTIONS** that are evaluated *after* the record is written:
  - **REPORT**
  - **ABORT**
  - **CHECKPOINT AFTER**
  - **FORCESTOP**
  - **STOP**
- **EVENTACTIONS** are usually combined, as in the following example:

```
EVENTACTIONS (CP BEFORE, REPORT, LOG, IGNORE TRANSACTION)
```

ORACLE

EVENTACTIONS in the example in the slide forces the process to issue a checkpoint, log an informational message, ignore the entire transaction (without processing any of it), and as a result generate a report.

EVENTACTIONS Syntax

```
EVENTACTIONS (
[STOP | SUSPEND | ABORT | FORCESTOP]
[IGNORE [RECORD | TRANSACTION [INCLUDEVENT]]
[DISCARD]
[LOG [INFO | WARNING]]
[REPORT]
[ROLLOVER]
[SHELL "<command>" |
SHELL ("<command>", VAR <variable> = {<column name> |
<expression>}
[, ...][, ...]) ]
[TRACE[2] <trace file> [TRANSACTION] [DDL[INCLUDE] |
DDLONLY]
[PURGE | APPEND]]
[CHECKPOINT [BEFORE | AFTER | BOTH]]
[, ...]
)
```

Event Marker Actions

Action	Explanation
STOP	Causes the process to end gracefully after the current transaction completes
SUSPEND	Pauses the process, which can still receive the SEND command. To resume, send a RESUME command.
ABORT	Forces an abrupt end of the process. The transaction is not completed and a recovery is necessary at the next start.
FORCESTOP	Forces a graceful end of the process only if the event record is the last operation in the transaction
IGNORE	Ignores (either completely or partially) the current event record or the transaction
DISCARD	Causes the process to write the specified event record to the discard file, updating the statistics of discarded records
LOG	Causes the process to log the event when the specified event record is encountered

ORACLE

STOP This can be combined with other EVENTACTIONS options, except ABORT and FORCESTOP.

SUSPEND When the process is in SUSPENDED mode, the INFO command shows it as RUNNING and the RBA field shows the last checkpoint position. SUSPEND cannot be combined with ABORT but can be combined with all other options.

ABORT The event record is not processed. A fatal error is written to the log and the event record is written to the discard file if DISCARD is also specified. ABORT can be combined only with CHECKPOINT BEFORE, DISCARD, SHELL, and REPORT.

FORCESTOP This can be combined with other EVENTACTIONS options, except ABORT, STOP, CHECKPOINT AFTER, and CHECKPOINT BOTH. If it is used with ROLLOVER, the rollover occurs only if the process stops gracefully.

IGNORE RECORD, TRANSACTION, and INCLUDEEVENT can be specified with IGNORE. RECORD is the default, which implies that the event record is ignored, but the rest of the active transaction is not ignored. TRANSACTION forces the process to ignore the entire transaction, whereas TRANSACTION INCLUDEEVENT forces the propagation of the event record to the trail, but the remaining part of the transaction is ignored.

DISCARD After a DISCARD command, the process resumes processing with the next record in the trail. With this option, the DISCARDFILE parameter should be used to specify the name of a discard file. By default, a discard file is not created. DISCARD can be combined with all other EVENTACTIONS options, except IGNORE.

LOG INFO (default) or WARNING can be specified with LOG. WARNING signifies a high-severity message. LOG can be combined with all other EVENTACTIONS options, except ABORT.

Event Marker Actions

Action	Explanation
REPORT	Causes the process to generate a report file when the specified event record is encountered
ROLLOVER	Causes Extract to roll over the trail to a new file when the specified event record is encountered
SHELL	Causes the process to execute the specified shell command when the event record is encountered
TRACE[2]	Causes process trace information to be written to a trace file when the specified event record is encountered. TRACE provides step-by-step processing information. TRACE2 identifies the code segments on which the process is spending the most time.
CHECKPOINT	Causes the process to write a checkpoint when the specified event record is encountered

ORACLE

REPORT

The REPORT message occurs after the event record is processed (unless DISCARD, IGNORE, or ABORT are used), so the report data will include the event record. REPORT can be combined with all other EVENTACTIONS options.

ROLLOVER

The ROLLOVER action occurs before Extract writes the event record to the trail file, which causes the record to be the first one in the new file unless DISCARD, IGNORE, or ABORT are also used. ROLLOVER can be combined with all other EVENTACTIONS options, except ABORT.

SHELL

The SHELL "<command>" executes a basic shell command. The command string is taken at its literal value and sent to the system that way. The command is case-sensitive and must be enclosed within double quotation marks. The SHELL command supports a more sophisticated version, which allows for parameter passing. Variables, column names carrying the before or after image of a column value, and expressions can be passed to the shell invocation for execution:

```
SHELL ("<command>",
VAR <variable> = {<column
name> | <expression>}
 [, ...][, ...])
```

<variable> is the user-defined name of the placeholder variable where the runtime variable value will be substituted.

<column name> can be the before or after (current) image of a column value.

<expression> can be the following, depending on whether column data or DDL is being handled:

- **Column Data**
  - The value from a TOKENS clause in a TABLE statement
  - A return value from any Oracle GoldenGate column conversion function
  - A return value from a SQLEXEC query or procedure
- **DDL**
  - Return value from the @TOKEN function (Replicat only)
  - Return value from the @GETENV function
  - Return value from other functions that do not reference column data (for example, @DATENOW)
  - Return value from the @DDL function

TRACE[2]

This can be combined with all other EVENTACTIONS options, except ABORT.

CHECKPOINT

This can be invoked with the BEFORE or AFTER clause to specify the following:

- For Extract, a checkpoint is performed before the event record is written to the trail.
- For Replicat, the checkpoint is performed before the SQL operation that is contained in the record is applied to the target.
- For Extract, AFTER writes a checkpoint after Extract writes the event record to the trail.
- For Replicat, AFTER writes a checkpoint after Replicat applies the SQL operation that is contained in the record to the target.

A BOTH clause is available to request a checkpoint before and after the process handles the record. CHECKPOINT BEFORE can be combined with all EVENTACTIONS options, whereas CHECKPOINT AFTER and BOTH can be combined with all EVENTACTIONS options, except ABORT.

Quiz

All **EVENTACTIONS** can be combined with each other, as long as **ABORT** is not used:

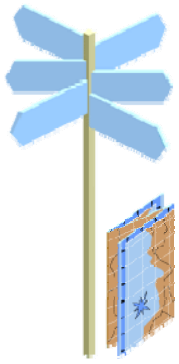
a. True
b. False

Answer: b

Several actions are incompatible with other actions. Therefore, it is not possible to mix and match all available actions, even if `ABORT` is not used.

Roadmap

- Explain the Oracle GoldenGate Event Marker System
- Identify the type of actions that can be triggered via the Event Marker mechanism
- Combine **`EVENTACTIONS`** options to inject sophisticated extraction and delivery logic into data replication

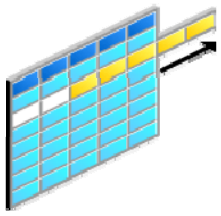


Example 1: Avoiding the Propagation of Deletes

In this Replicat example, all deletes that are performed against a specific table are logged but are not allowed in the target database.

```
MAP src.customer, TARGET trg.customer;

TABLE src.customer, FILTER (@GETENV('GGHEADER','OPTYPE')&
  = 'DELETE'), EVENTACTIONS (LOG INFO, IGNORE);
```



ORACLE

The `TABLE` statement for Replicat retrieves the operation that is performed. If the trail record pertains to a delete, the operation is logged but the record is discarded; the delete is not applied to the target table.

Example 2: SHELL Triggering a Remote Backup

In this example, a table called **EVENT_TRIGGER** exists on the source database. Inserting a row in this table, with a specific keyword, triggers an **RMAN** full backup to be performed on the target system.

```
MAP src.event_trigger, TARGET trg.event_trigger,
 FILTER (@streq(@GETENV('GGHEADER','OPTYPE'),'INSERT')
 AND @streq(TRIGGER_TYPE,'PERFORM BACKUP')),
 EVENTACTIONS (SHELL '/home/oracle/scripts/rman_bck.sh',
 REPORT);
```

ORACLE

The `EVENT_TRIGGER` table contains one column: `TRIGGER_TYPE`.

The `"insert into EVENT_TRIGGER values ('PERFORM BACKUP');"` SQL statement that is performed on the source database is written in the trail, and then shipped to the remote Replicat.

Replicat, because of the `FILTER` options, intercepts the `'PERFORM BACKUP'` value, and the corresponding `EVENTACTIONS` clause invokes the `SHELL` command, which calls the `rman_bck.sh` script.

Such an event is written in the report file on the target system.

Important: The return value from `'rman_bck.sh'` must be `0` (zero) to indicate success. A value different from `0` causes Replicat to abend.

Example 3: Stopping Replicat Upon Detection of Data Anomalies

In this example, a serious data anomaly is detected for which immediate action is required. The balance of one account is overdrawn by more than $15,000, and **ABORT** is used to stop Replicat.

```
MAP src.account, TARGET trg.account;
TABLE src.account, FILTER (balance < -15000), &
  EVENTACTIONS (ABORT);
```

In the example in the slide, 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, which does not need to be discarded.

Example 4: Tracing a Specific Data Insertion

In this example, government regulations mandate the tracking of specific customers residing in countries without reciprocal banking disclosure agreements. When a new customer is entered in the source database, and the country of residence is “CAYMAN ISLANDS,” a trace file is generated to monitor the transaction.

```
MAP src.customer, TARGET trg.customer;
TABLE src.customer,
 FILTER (@streq(@GETENV ('GGHEADER', 'OPTYPE'),'INSERT')
 AND  @streq(country, 'CAYMAN ISLANDS')),
EVENTACTIONS (TRACE country_risk.trc TRANSACTION);
```

In the example in the slide, the transaction is allowed to go through. A trace file that reports the insertion of a specific record is generated if it does not already exist; otherwise, the information is appended to the existing trace file name `country_risk.trc`.

Summary

In this lesson, you should have learned how to:

- Use the Event Marker System provided by Oracle GoldenGate
- Identify the type of actions that can be triggered via the Event Marker mechanism
- Combine the **`EVENTACTIONS`** options to trigger:
  - Tracing specific transactions
  - Preventing deletes from being applied to the target database
  - Performing remote tasks such as a database backup
  - Stopping the Replicat process when serious anomalies are detected

ORACLE

Practice 5: Overview

This practice covers the following topics:

- Triggering an end-of-day processing job
- Stopping processing after detecting data anomalies
- Tracing the insertion of rows in which specific columns carry abnormally high values
- Executing shell scripts to validate data; stopping processing if validation fails

6

Data Mapping, Data Selection/Filtering, and Data Transformation

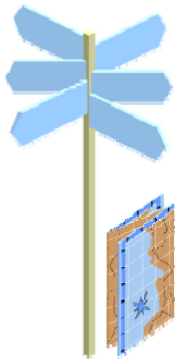
Objectives

After completing this lesson, you should be able to:

- Map columns between different schemas
- Select and filter data for replication
- Use built-in data transformation functions
- Use **SQLEXEC** to interact directly with a database

Roadmap

- Mapping columns between different schemas
- Selecting and filtering data for replication
- Using built-in data transformation functions
- Using `SQLEXEC` to interact directly with a database



Column Mapping: Overview

- GoldenGate provides the capability to map columns from one table to another.
- Data can be transformed between dissimilar database tables by:
  - Using `COLMAP` to map target columns from your source columns
  - Using `COLMATCH` for global column mapping
- GoldenGate automatically matches source to target column names with `USEDEFAULTS`.
- Mapping can be applied either when extracting or replicating data.

Extract and Replicat provide the capability to transform data between two dissimilarly structured database tables or files. These features are implemented with the `COLMAP` clause in the `TABLE` and `MAP` parameters.

Data Type Conversions

Numeric fields are converted from one type and scale to match the type and scale of the target. If the scale of the source is larger than that of the target, the number is truncated on the right. If the target scale is larger than the source, the number is padded with zeros.

Varchar and character fields can accept other character, varchar, group, and datetime fields, or string literals enclosed in quotation marks. If the target character field is smaller than that of the source, the character field is truncated on the right.

Column Mapping: COLMAP Syntax

Syntax for the COLMAP clause:

```
MAP <table spec>, TARGET <table spec>,
COLMAP (
[USEDEFAULTS, ]
<target column> = <source expression>
[, BINARYINPUT]
[, ...]
);
```


COLMAP Syntax

<table spec> is the source or target table.

<target column> =<source expression> explicitly defines a source-target column map.

<target column> is the name of the target column.

<source expression> can be any of the following:

- The name of a source column, such as ORD_DATE
- A numeric constant, such as 123
- A string constant within quotation marks, such as 'ABCD'
- An expression that uses a GoldenGate column-conversion function, such as @STREXT (COL1, 1, 3)

BINARYINPUT

Use BINARYINPUT when the target column is defined as a binary data type, such as RAW or BLOB, but the source input contains binary zeros in the middle of data. Use BINARYINPUT when replicating a full Enscribe record that is defined as a single column into a target column. The source input is handled as a binary input, and replacement of data values is suppressed.

USEDEFAULTS

This automatically maps the source and target columns that have the same name, if they are not specified in an explicit column map. Use an explicit map or `USEDEFAULTS`, but not both, for the same set of columns. Specify `USEDEFAULTS` before explicit column maps.

Note: If you set up global column mapping rules with `COLMATCH` parameters, you can map columns with different names to each other by using default mapping.

Column Mapping: COLMATCH Syntax

Syntax for the COLMATCH clause:

```
MAP HR.CONTACT, TARGET HR.PHONE,
 COLMATCH (USEDEFAULTS,
 NAME = CUST_NAME,
 PHONE_NUMBER = @STRCAT( '(', AREA_CODE, ')',
   PH_PREFIX, '-', PH_NUMBER ) );
```


- **NAMES <target column> = <source column>:** Maps based on column names
- **PREFIX <prefix>:** Ignores the specified name prefix
- **SUFFIX <suffix>:** Ignores the specified name suffix
- **RESET:** Turns off previously defined COLMATCH rules for subsequent TABLE or MAP statements

Global Column Mapping

- **COLMATCH** can be used to create global rules for column mapping, which apply to multiple tables.
- If a recurring column name in the source database must be consistently changed in the target database, using **COLMATCH** provides a more convenient syntax.
- Example:
  - Source database column name **ACCT**
  - Target database column name **ACCOUNT**

```
COLMATCH NAMES ACCT = ACCOUNT
```

- The **COLMATCH** line shown here can appear in the parameter file for either Extract or Replicat, and can remain active for all **TABLE** or **MAP** statements that appear after the **COLMATCH** clause.

If several tables in the source database are named with a prefix (or a suffix) and that prefix (or suffix) must be dropped in the target database, COLMATCH provides a convenient way to accomplish that.

Example: In the source database, several tables have the TAX_ prefix, but the corresponding tables in the target database have dropped that prefix. Using COLMATCH, you can establish the following mapping rule:

```
COLMATCH PREFIX TAX_
```

Important: A COLMATCH rule applies to multiple TABLE or MAP statements. All TABLE or MAP statements that appear after the COLMATCH clause in the parameter files are affected by that rule. COLMATCH provides the RESET clause to terminate the substitution rule when it is no longer needed.

Column Mapping: Example

```
MAP HR.CONTACT, TARGET HR.PHONE,
  COLMAP (USEDEFAULTS,
  NAME = CUST_NAME,
  PHONE_NUMBER = @STRCAT( '(', AREA_CODE, ')',
    PH_PREFIX, '-', PH_NUMBER ) );
```


The example in the slide:

- Moves the HR.CONTACT CUST_NAME column value to the HR.PHONE NAME column
- Concatenates HR.CONTACT AREA_CODE, PH_PREFIX, and PH_NUMBER with quotation marks and hyphen literals to derive the PHONE_NUMBER column value
- Automatically maps other HR.CONTACT columns to the HR.PHONE columns that have the same name

Column Mapping: Building History

This example uses special values to build history of operations data:

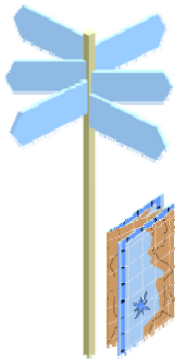
```
INSERTALLRECORDS
MAP SALES.ACCOUNT, TARGET REPORT.ACCTHISTORY,
  COLMAP (USEDEFAULTS,
  TRAN_TIME = @GETENV('GGHEADER','COMMITTIMESTAMP'),
  OP_TYPE = @GETENV('GGHEADER', 'OPTYPE'),
  BEFORE_AFTER_IND =
  @GETENV('GGHEADER', 'BEFOREAFTERINDICATOR'),
 );
```


`INSERTALLRECORDS` causes Replicat to insert every change operation made to a record as a new record in the database. The initial insert and subsequent updates and deletes are maintained as point-in-time snapshots.

- **COLMAP:** Uses the `@GETENV` function to get historical data from the GoldenGate trail header
- **TRAN_TIME:** Picks up the commit time stamp for the date of the transaction
- **OP_TYPE:** Stores whether it is an insert, an update, or a delete operation
- **BEFORE_AFTER_IND:** Indicates whether it is storing a “before” or “after” image

Roadmap

- Mapping columns between different schemas
- Selecting and filtering data for replication
- Using built-in data transformation functions
- Using `SQLEXEC` to interact directly with a database



Data Selection: Overview

GoldenGate provides the ability to select or filter out data based on a variety of levels and conditions.

Parameter	Component	Selection
TABLE or MAP	<table spec> WHERE FILTER	Table Row Row, Operation, Range
TABLE	COLS | COLSEXCEPT	Columns

Table Selection

The MAP (Replicat) or TABLE (Extract) parameter can be used to select a table.

```
MAP sales.tcustord, TARGET sales.tord;
```

Rows Selection

The following WHERE option can be used with MAP or TABLE to select rows for the 'AUTO' product type.

```
WHERE (PRODUCT_TYPE = 'AUTO');
```

Operations Selection

The following can be used with MAP or TABLE to select rows with amounts greater than zero only for update and delete operations.

```
FILTER (ON UPDATE, ON DELETE, amount > 0);
```

Columns Selection

The COLS and COLSEXCEPT options of the TABLE parameter allow selection of columns, as in the following example. Use COLS to select columns for extraction and use COLSEXCEPT to select all columns, except those designated by COLSEXCEPT:

```
TABLE sales.tcustord, TARGET sales.tord, COLSEXCEPT
   (facility_number);
```

Data Selection: WHERE Clause

- The WHERE clause is the simplest form of selection.
- The WHERE clause appears with either the MAP or TABLE parameter and must be surrounded by parentheses.
- The WHERE clause cannot:
  - Perform arithmetic operations
  - Refer to trail header and user token values
- Use the FILTER clause for more complex selections with built-in functions.

Examples using the WHERE clause:

```
MAP sales.tcustord, TARGET sales.cust_ord,
WHERE (PRODUCT_AMOUNT > 10000);

MAP sales.tcustord, TARGET sales.cust_ord,
WHERE (PRODUCT_TYPE = 'AUTO');
```

Data Selection: WHERE Clause

WHERE can perform an evaluation for:

Element Description	Example
Columns	PRODUCT_AMT
Comparison operators	=, <>, >, <, >=, <=
Numeric values	-123, 5500.123
Literal strings	'AUTO', 'Ca'
Field tests	@NULL,@PRESENT,@ABSENT
Conjunctive operators	AND, OR

Arithmetic operators and floating-point data types are not supported by WHERE.

Data Selection: WHERE Clause Examples

- Only rows where the state column has a value of CA are returned.

```
WHERE (STATE = 'CA');
```

- Only rows where the amount column has a value of NULL are returned. Note that if amount is not part of the update, the result is false.

```
WHERE (AMOUNT = @NULL);
```

- Only rows where amount is part of the operation and that have a value that is not null are returned.

```
WHERE (AMOUNT @PRESENT AND AMOUNT <> @NULL);
```

- Only rows where the account identifier is greater than CORP-ABC are returned.

```
WHERE (ACCOUNT_ID > 'CORP-ABC');
```

ORACLE

Selection: FILTER Clause

- The FILTER clause provides complex evaluations to include or exclude data selection.
- The FILTER clause appears on either the MAP or TABLE parameter and must be surrounded by parentheses.
- With FILTER, you can:
  - Deploy other GoldenGate built-in functions
  - Use multiple FILTERs on one statement
    - If any filter fails, the entire filter clause fails.
  - Include multiple option clauses (for example, on insert or update)
  - Raise a user-defined error for exception processing

When multiple filters are specified per a given TABLE or MAP entry, the filters are executed until one fails or until all are passed. The failure of any filter results in a failure of all filters.

Filters can be qualified with operation type, so you can specify different filters for inserts, updates, and deletes.

The FILTER RAISEERROR option creates a user-defined error number if the filter clause is true. In the following example, error 9999 is generated when the BEFORE time stamp is earlier than the CHECK time stamp. This also selects only update operations.

```
FILTER (ON UPDATE, BEFORE.TIMESTAMP < CHECK.TIMESTAMP,
RAISEERROR 9999);
```

Data Selection: FILTER Clause

Syntax:

```
FILTER (<option> [, <option]) ,
       [FILTER (<option> [, <option]) ] [, …]
```

Where <option> is one of the following:

- **<column specification>**
- **<field conversion function>**
- **<ON INSERT | UPDATE | DELETE >**
- **<IGNORE INSERT | UPDATE | DELETE>**
- **<RAISEERROR <error number> >**

ON INSERT | UPDATE | DELETE

Specifically limits the filter to be executed on an insert, an update, or a delete. More than one ON clause can be specified (for example, ON UPDATE, ON DELETE executes on updates and deletes, but not inserts).

IGNORE INSERT | UPDATE | DELETE

Specifically ignores the specified type of operation

RAISEERROR <error num>

Causes an error to be raised as if there was a database error in the map (RAISEERROR has no effect in the Extract program). In combination with REPERROR, RAISEERROR can be used to control what happens if a filter is not passed (the operation can be discarded, posted to an exceptions table, reported, and so on).

Data Selection: FILTER Clause Examples

- The following example includes rows where the price multiplied by the amount exceeds 10,000:

```
FILTER
((PRODUCT_PRICE*PRODUCT_AMOUNT)>10000);
```

- The following example includes rows containing a certain string:

```
FILTER (@STRFIND(CITY, 'NEW YORK')>0);
```

- Why is the preceding example not constructed like the one that follows? The following filter will fail!

```
FILTER(CITY = 'NEW YORK');
```


The equal sign (=) does not work with strings. To test for equality or inequality between strings, the function @STREQ must be used.

Data Selection: RANGE Function

The **@RANGE** function is used with Classic Replicat.

- Divides workload into multiple, randomly distributed groups of data
- Guarantees that the same row is always processed by the same process
- Determines which group the range falls in by computing a hash against the primary key or user-defined columns
- Syntax:

```
@RANGE (<my range>, <total ranges>
        [, <column> [, ...]])
```

- Example:

```
TABLE SALES.ACCOUNT, FILTER (@RANGE (1,3));
```

ORACLE

Integrated Replicat is now the preferred way to speed up data delivery to target databases. When the target database is not the Oracle database, the @RANGE function is still the recommended solution for parallel Replicat streams, which are likely to increase data delivery performance.

@RANGE helps divide workload into multiple, randomly distributed groups of data, while guaranteeing that the same row will always be processed by the same process. For example, @RANGE can be used to split the workload of different key ranges for a heavily accessed table into different Replicat processes.

The user specifies both a range that applies to the current process and the total number of ranges (generally the number of processes).

@RANGE computes a hash value of all the columns specified or, if no columns are specified, the primary key columns of the source table. A remainder of the hash and the total number of ranges is compared with the ownership range to determine whether or not @RANGE evaluates to true or false. Note that the total number of ranges will be adjusted internally to optimize even distribution across the number of ranges.

Restriction: @RANGE cannot be used if primary key updates are performed on the database.

Data Selection: RANGE Function Examples

- For transaction volume beyond the capacity of a single Replicat, the following example shows three Replicat groups, each processing one-third of the data.
- Hashing each operation by primary key to a particular Replicat guarantees the original sequence of operations.

Replicat #1

```
MAP SALES.ACCOUNT,
   TARGET SALES.ACCOUNT, FILTER (@RANGE (1,3));
```

Replicat #2

```
MAP SALES.ACCOUNT,
   TARGET SALES.ACCOUNT, FILTER (@RANGE (2,3));
```

Replicat #3

```
MAP SALES.ACCOUNT,
   TARGET SALES.ACCOUNT, FILTER (@RANGE (3,3));
```


The example in the slide demonstrates three Replicat processes, with each Replicat group processing one-third of the data in the GoldenGate trail based on the primary key.

Data Selection: RANGE Function Examples

- Two tables, REP and ACCOUNT, that are related by REP_ID require three Replicats to handle the transaction volumes.
- By hashing the REP_ID column, related rows are always processed to the same Replicat.

```
RMTTRAIL ./dirdat/aa
TABLE SALES.REP, FILTER (@RANGE (1,3));
TABLE SALES.ACCOUNT, FILTER (@RANGE (1,3,REP_ID));
```

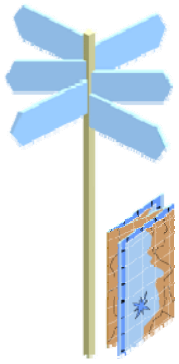
```
RMTTRAIL ./dirdat/bb
TABLE SALES.REP, FILTER (@RANGE (2,3));
TABLE SALES.ACCOUNT, FILTER (@RANGE (2,3,REP_ID));
```

```
RMTTRAIL ./dirdat/cc
TABLE SALES.REP, FILTER (@RANGE (3,3));
TABLE SALES.ACCOUNT, FILTER (@RANGE (3,3,REP_ID));
```


The REP_ID column is the primary key for the REP table. The TABLE statements specified in the parameter file for each Data Pump Extract split the replication of the REP and ACCOUNT tables in three groups. Each group processes the same REP_IDs. The REP table does not specify a column on which to base the range calculation, so by default Oracle GoldenGate will use the primary key column. The ACCOUNT table specifies the REP_ID column as the column on which the hash value is computed. Rows with the same REP_ID value in both tables will be processed by the same Replicat group on the replication target database, thus ensuring transactional integrity.

Roadmap

- Selecting and filtering data for replication
- Mapping columns between different schemas
- Using built-in data transformation functions
- Using `SQLEXEC` to interact directly with a database



Functions: Data Transformation

- GoldenGate provides the capability to transform columns by using a set of built-in functions.
- Transformation functions can be applied for either Extract or Replicat.
- If you require more, you also have the ability to call your own logic through user exits.

Functions: Overview

- Using column conversion functions, you can:
  - Perform string and number conversion
  - Extract portions of strings or concatenate columns
  - Compare strings or numbers
  - Perform a variety of date mappings
  - Use single or nested `IF` statements to evaluate numbers, strings, and other column values to determine the appropriate value and format for the target columns
- Functions are identified with the `@` prefix.

Functions: Example

```
MAP SALES.ACCOUNT, TARGET REPORT.ACCOUNT,
COLMAP ( USEDEFAULTS,
   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)
   );
```


The example in the slide uses `@DATE` to derive the `TRANSACTION_DATE` by converting the source date columns `YEAR` in the format `YY`, `DAY` in the format `DD`, and `MONTH` in the format `MM` to a target date with the format `YYYY-MM-DD`. The syntax for the `@DATE` function is:

```
@DATE ('<out descriptor>', '<in descriptor>', <source col>
[, '<in descriptor>', <source col>]
[, ...])
```

The example in the slide uses `@STREXT` to extract portions of a string field into three different columns. It takes the first through the third characters from the source's `PHONE-NO` to populate the target's `AREA_CODE`, characters 4 through 6 for `PHONE_PREFIX`, and characters 7 through 10 for `PHONE_NUMBER`. The syntax for the `@STREXT` function is:

```
@STREXT (<column or literal string>,
<begin position>, <end position>)
```

Functions: Performing Tests on Column Values

Function	Description
`CASE`	Allows a user to select a value depending on a series of value tests
`EVAL`	Allows a user to select a value depending on a series of independent tests
`IF`	Selects one of two values depending on whether a conditional statement returns `TRUE` or `FALSE`
`COLSTAT`	Returns whether a column value is missing, NULL, or invalid
`COLTEST`	Tests whether a column value is present, missing, NULL, or invalid
`VALONEOF`	Returns true if a column contains one of a list of values

ORACLE

The functions listed in the slide select a value based on tests against the current value.

Discussion Points: `IF` Function

Syntax:

```
@IF (<conditional expression>,
     <value if expression is non-zero>,
     <value if expression is zero>)
```

Non-zero is considered true and zero (`0`) is considered false.

1. What `IF` clause would you use to set the target column `AMOUNT_COL` to `AMT` only if `AMT` is greater than zero, and otherwise return zero?

1. `AMOUNT_COL = @IF (AMT > 0, AMT, 0)`

Discussion Points: `IF` Function

2. What `IF` clause would you use to set `ORDER_TOTAL` to `PRICE*QUANTITY` if both `PRICE` and `QUANTITY` are greater than zero, and otherwise return zero?

2. `ORDER_TOTAL = @IF (PRICE > 0 AND QUANTITY > 0, PRICE * QUANTITY, 0)`

Functions: Working with Dates

Function	Description
DATE	Returns a date from a variety of sources in a variety of output formats
DATEDIFF	Returns the difference between two dates or times
DATENOW	Returns the current date and time

The functions listed in the slide return dates in various formats and calculate the difference between two dates.

Discussion Points: DATE Function

Syntax:

```
@DATE ('<output format>',
       '<input format>', <source column>
    [, '<input format>', <source column>]
    [,...])
```

1. What DATE expression would you use to convert the year, month, and day columns to a date?

1. `date_col = @DATE ('YYYY-MM-DD', 'YY', date1_yy, 'MM', date1_mm, 'DD', date1_dd)`

Formats that are supported for both input and output are:

CC	century
YYYY	four-digit year
YY	two-digit year
MMM	alphanumeric month (such as APR)
MM	numeric month
DDD	numeric day of the year (for example: 001, 365)
DD	numeric day of month
HH	hour
MI	minute
SS	seconds
FFFFFF	fraction (up to microseconds)
DOW0	numeric day of the week (Sunday = 0)

DOW1	numeric day of the week (Sunday = 1)
DOWA	alphanumeric day of the week, (for example, SUN)
JUL	Julian day
JTS	GMT and JTS Julian time stamp
JTSLCT	Julian time stamp that is already local time, or to keep local time when converting to a Julian time stamp
STRATUS	Application time stamp
CDATE	C time stamp in seconds since the Epoch

The formats that are supported for input are:

TTS	NonStop 48-bit time stamp
PHAMIS	Application date format

Calculating the Century

When a two-digit year is supplied but a four-digit year is required in the output, the system can calculate the century (as 20 if the year is < 50). It can be hard-coded, or the `@IF` function can be used to set a condition.

Discussion Points: DATE Function

2. What DATE expression would you use to convert a numeric column that is stored as YYYYMMDDHHMISS to a Julian time stamp?

2. `julian_ts_col = @DATE ('JTS', 'YYYYMMDDHHMISS', numeric_date)`

Functions: Working with Strings and Numbers

Function	Description
COMPUTE	Returns the result of an arithmetic expression
NUMBIN	Converts a binary string into a number
NUMSTR	Converts a string into a number
STRCAT	Concatenates two or more strings
STRCMP	Compares two strings to determine if they are equal, or if the first is less or greater than the second
STREQ	Tests to see if two strings are equal; returns 1 if equal and 0 if not equal

Functions: Working with Strings and Numbers

Function	Description
STREXT	Extracts selected characters from a string
STRFIND	Finds the occurrence of a string within a string
STRLEN	Returns the length of a string
STRLTRIM	Trims the leading spaces in a column
STRNCAT	Concatenates one or more strings up to a specified number of characters per string
STRNCMP	Compares two strings up to a certain number of characters

ORACLE

Functions: Working with Strings and Numbers

Function	Description
STRNUM	Converts a number into a string, with justification and zero-fill options
STRRTRIM	Trims the trailing spaces in a column
STRSUB	Substitutes one string for another within a column
STRTRIM	Trims both leading and trailing spaces in a column
STRUP	Changes a string to uppercase

ORACLE

Discussion Points: STRCAT Function

Syntax:

```
@STRCAT (<string1>, <string2> [, ...])
```

The strings can be column names or literal values in quotation marks.

1. What STRCAT expression would you use to concatenate the columns LASTNAME and FIRSTNAME, separated by a semicolon?

1. `NAME = @STRCAT (LASTNAME, ';' ,FIRSTNAME)`

Discussion Points: STRCAT Function

2. What STRCAT expression would you use to concatenate a country code, an area code, and a local phone number into an international phone number with hyphens between the components?

2. `INTL_PHONE = @STRCAT (COUNTRY_CODE, '-', AREA_CODE, '-', LOCAL_PHONE)`

Discussion Point: STREXT Function

Syntax:

```
@STREXT (<column or literal string>,
         <begin position>, <end position>)
```

- What STREXT expressions would you use to split a phone number into three columns (area code, prefix, and phone number)?


```
AREA_CODE = @STREXT (PHONE, 1, 3), PREFIX = @STREXT (PHONE, 4, 6),
PHONE_NO = @STREXT (PHONE, 7, 10)
```

Functions: Other

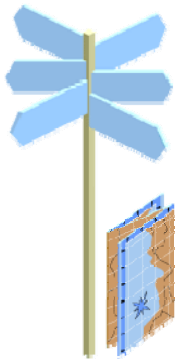
Function	Description
`BINARY`	Keeps source data in its original binary format in the target when the source column is defined as a character
`BINTOHEX`	Converts a binary string to a hexadecimal string
`GETENV`	Returns information about the GoldenGate environment, trail file header, trail record header, last replicated operation, and lag; can retrieve the commit time stamp in local time or GMT
`GETVAL`	Extracts parameters from a stored procedure as input to a `FILTER` or `COLMAP` clause

Functions: Other

Function	Description
HEXTOBIN	Converts a hexadecimal string to a binary string
HIGHVAL, LOWVAL	Emulates COBOL functions that allow you to set a numeric limit on string or binary data types
RANGE	Divides workload into multiple groups of data, while ensuring that the same row is always sent to the same process. Range uses a hash against primary key or user-defined columns.
TOKEN	Maps environmental values that are stored in the user token area to the target column

Roadmap

- Selecting and filtering data for replication
- Mapping columns between different schemas
- Using built-in functions
- Using `SQLEXEC` to interact directly with a database



SQLEXEC: Overview

- Extends GoldenGate capabilities by enabling Extract and Replicat to communicate with the application database through SQL queries or by running stored procedures
- Extends data integration beyond what can be done with GoldenGate functions

The SQLEXEC option enables both Extract and Replicat to communicate with the user's database, by using either SQL queries or stored procedures. SQLEXEC can be used to interface with a virtually unlimited set of functions that are supported by the underlying database.

Stored Procedure Capabilities

Stored procedures extend the functionality of popular databases such as Oracle, DB2, SQL Server, and Teradata. Users write stored procedures to perform custom logic, typically involving the database in some way, using languages such as Oracle's PL/SQL and Microsoft's Transact-SQL.

Extract and Replicat enable stored procedure capabilities to be leveraged for Oracle, SQL Server, and DB2. Tying together industry-standard stored procedure languages with extraction and replication functions brings a familiar, powerful interface to virtually unlimited functionality.

Stored procedures can also be used as an alternative method for inserting data into the database, aggregating data, denormalizing or normalizing data, or executing any other function that requires database operations as input. Extract and Replicat can support stored procedures that accept only input, or procedures that produce output as well. Output parameters can be captured and used in subsequent map and filter operations.

SQL Query Capabilities

In addition to stored procedures, Extract and Replicat can execute specified database queries that either return results (`SELECT` statements) or update the database (`INSERT`, `UPDATE`, and `DELETE` statements).

SQLEXEC: Basic Functionality

- Execute a stored procedure or SQL query by using the **SQLEXEC** clause of the **TABLE** or **MAP** parameter.
- (Optional) Extract output parameters from the stored procedure or SQL query as input to a **FILTER** or **COLMAP** clause by using the **@GETVAL** function.
- Use **SQLEXEC** at the root level (without input/output parameters) to call a stored procedure, run a SQL query, or issue a database command.

Before defining the SQLEXEC clause, a database logon must be established. This is done via the SOURCEDB, USERID, and PASSWORD parameters for Extract, and the TARGETDB, USERID, and PASSWORD parameter for Replicat.

When SQLEXEC is used, a mapping between one or more input parameters and source columns or column functions must be supplied.

When at least one SQLEXEC entry is supplied for a given Replicat map entry, a target table is not required.

SQLEXEC: Using with Lookup Stored Procedure

The following stored procedure performs a query to return a description when given a code:

```
CREATE OR REPLACE PROCEDURE LOOKUP
       (IN CODE_PARAM VARCHAR(20), OUT DESC_PARAM
VARCHAR(20))
BEGIN
       SELECT DESC_COL INTO DESC_PARAM
       FROM LOOKUP_TABLE
       WHERE CODE_COL = CODE_PARAM;
END;
```


Table Lookup Using a Stored Procedure

Mapping can be augmented with a simple database lookup procedure in Extract or Replicat. The example in the slide illustrates the stored procedure to perform a table lookup.

SQLEXEC: Using with Lookup Stored Procedure

The following parameter entry:

- Maps data from the ACCOUNT table to the NEWACCT table
- Forces Extract to execute the LOOKUP stored procedure when processing any rows from ACCOUNT before executing the column map
- Maps values returned in desc_param to the newacct_val column by using the @GETVAL function

```
MAP HR.ACCOUNT, TARGET HR.NEWACCT,
      SQLEXEC (spname lookup,
           params (code_param = account_code)),
      COLMAP (USEDEFAULTS, newacct_id = account_id,
            newacct_val = @GETVAL(lookup.desc_param));
```


The example in the slide illustrates how a stored procedure can be used for mapping in a Replicat parameter file.

SQLEXEC: Using with SQL Query

The following example performs a SQL query directly to return the description. **@GETVAL** is used to retrieve the return parameter.

```
MAP HR.ACCOUNT, TARGET HR.NEWACCT,
 SQLEXEC (id lookup,
   query 'select desc_param from lookup_table
    where code_col = :code_param',
    params (code_param = account_code)),
COLMAP (USEDEFAULTS, newacct_id = account_id,
    newacct_val = @GETVAL(lookup.desc_param));
```


The example parameter file entries in the slide illustrate a mapping by using a simple SQL query to look up the account description.

SQLEXEC: Syntax in a TABLE or MAP Statement

When the SQLEXEC parameter is used in a TABLE or MAP statement, the syntax is:

```
SQLEXEC (
{ SPNAME <sp name> | QUERY '<sql query>' }
[, ID <logical name>]
{ PARAMS <param spec> | NOPARAMS}
[, BEFOREFILTER | AFTERFILTER]
[, DBOP]
[, EXEC <frequency>]
[, ALLPARAMS <option>]
[, PARAMBUFSIZE <num bytes>]
[, MAXVARCHARLEN <num bytes>]
[, TRACE <option>]
[, ERROR <action>])
```


The SQLEXEC option is specified as an option in TABLE and MAP statements within EXTRACT and REPLICAT parameter files. Use either SPNAME (for stored procedure) or QUERY (for SQL query).

- **SPNAME <sp name>:** This is the name of the stored procedure in the database. This name can be used when extracting values from the procedure.
- **QUERY '<sql query>':** This is a query to execute against the database. The query must be a legitimate SQL statement for the particular database.
- **ID <logical name>:** This is required with the QUERY parameter to reference the column values returned by the query, or when you can invoke several instances of a stored procedure to reference each instance separately (for example, to invoke the same stored procedure to populate two separate target columns)
- **PARAMS <param spec>:** <param spec> is [OPTIONAL | REQUIRED] <sp param name> = <source column> | <source function>.
  For a stored procedure, <sp param name> is the name of any parameter in the stored procedure or query that can accept input.
  For an Oracle SQL query, <sp param name> is the name of any input parameter in the query without the leading colon. For example, if the parameter :param1 appears in the query, it is specified as param1 in the PARAMS clause.
- **NOPARAMS:** This specifies that there are no parameters.

- **BEFOREFILTER**: Use BEFOREFILTER to cause the stored procedure or query to execute before applying filters to a particular map. By default, stored procedures and queries are executed after the filtering logic has been applied.
- **AFTERFILTER**: Use AFTERFILTER (the default) to cause the stored procedure or query to execute after applying filters to a particular map. This enables you to skip stored procedure or query processing unless the map is actually executed.
- **DBOP:** Use DBOP if the stored procedure or query updates the database and you want the process to execute transaction commit logic.
- **EXEC <frequency>:** This determines the frequency of execution for the stored procedure or query.
- **MAP:** This executes the stored procedure or query once for each source-target table map for which it is specified. MAP renders the results invalid for any subsequent maps that have the same source table. For example, if a source table is being synchronized with more than one target table, the results are valid for only the first source-target map. MAP is the default.
- **ONCE:** This executes the stored procedure or query once during the course of a GoldenGate run, at the first invocation of the associated FILE or MAP statement. The results remain valid for as long as the process remains running.
- **TRANSACTION:** This executes the stored procedure or query once per source transaction. The results remain valid for all operations of the transaction.
- **SOURCEROW:** This executes the stored procedure or query once per source row operation. Use this option when you are synchronizing a source table with more than one target table, so that the results of the procedure, stored procedure, or query are invoked for each source-target mapping.
- **ALLPARAMS {REQUIRED | OPTIONAL}**
  - **REQUIRED:** Specifies that all parameters must be present for the stored procedure or query to execute
  - **OPTIONAL:** Enables the stored procedure or query to execute without all parameters being present (the default)
- **PARAMBUFSIZE <num bytes>:** By default, each stored procedure or query is assigned 10,000 bytes of space for input and output. For stored procedures requiring more space, specify <num bytes> with an appropriate amount of buffer space.
- **MAXVARCHARLEN <num bytes>:** This determines the maximum length allocated for any output parameter in the stored procedure or query. The default is 200 bytes.
- **TRACE [ALL|ERROR]:** If TRACE or TRACE ALL is specified, the input and output parameters for each invocation of the stored procedure or query are output to the report file. If TRACE ERROR is specified, parameters are output only after an error occurs in the stored procedure or query.
- **ERROR <action>:** This requires one of the following arguments:
  - **IGNORE:** The database error is ignored and processing continues.
  - **REPORT:** The database error is written to a report.
  - **RAISE:** The database error is handled in the same way as a table replication error.
  - **FINAL:** The database error is handled as a table replication error but does not process any additional queries.
  - **FATAL:** Database processing abends.

SQLEXEC: Syntax as a Stand-Alone Statement

When a SQLEXEC parameter is used at the root level, the syntax is:

```
SQLEXEC
{'exec <sp name> ()' | '<sql query>' | '<database
command>'}
[EVERY <n> {SECONDS | MINUTES | HOURS | DAYS}]
[ONEXIT]
```

Examples:

```
SQLEXEC 'exec prc_job_count ()'
SQLEXEC ' select x from dual '
SQLEXEC 'exec prc_job_count ()' EVERY 30 SECONDS
SQLEXEC 'exec prc_job_count ()' ONEXIT
SQLEXEC 'SET TRIGGERS OFF'
```


- **'exec <sp name> ()':** Specifies the name of a stored procedure to execute. The statement must be enclosed within double quotation marks.
  Example: `SQLEXEC 'exec prc_job_count ()'`
- **'<sql query>':** Specifies the name of a query to execute. Enclose the query within quotation marks. For a multiline query, use quotation marks on each line. For best results, type a space after each begin quotation mark and before each end quotation mark (or at least before each end quotation mark).
  Example: `SQLEXEC ' select x from dual '`
- **'<database command>':** Executes a database command
- **EVERY <n> {SECONDS | MINUTES | HOURS | DAYS}:** Causes a stand-alone stored procedure or query to execute at defined intervals
  Example: `SQLEXEC 'exec prc_job_count ()' EVERY 30 SECONDS`
- **ONEXIT:** Executes the SQL statement when the Extract or Replicat process stops gracefully

SQLEXEC: Error Handling

- There are two types of potential errors that must be considered when implementing SQLEXEC:
  - An error is raised by the database (either the query or stored procedure).
  - The procedure map requires a column that is missing from the source database operation (probably in an update statement).
- When an error is raised by the database:
  - Error handling allows the error to be ignored or reported
  - These error handling options are controlled with the ERROR option in the SQLEXEC clause

SQLEXEC: Using the GETVAL Function to Get Results

The **GETVAL** function supplies a mechanism to:

- Extract stored procedure and query output parameters
- Subsequently map them or use in a **COLMAP** or **FILTER** clause

Syntax:

```
@GETVAL (<name>.<parameter>)
```

Example:

```
MAP schema.tab1, TARGET schema.tab2,
SQLEXEC(SPNAME lookup, PARAMS (param1 = srccol)),
COLMAP(USEDEFAULTS, targcol = @GETVAL (lookup.param1));
```

ORACLE

<name>: The name of the stored procedure or query. When using SQLEXEC to execute the procedure or query, valid values are as follows:

- **Queries:** Use the logical name specified with the ID option of the SQLEXEC clause. ID is a required SQLEXEC argument for queries.
- **Stored procedures:** Use one of the following, depending on how many times the procedure is to be executed within a TABLE or MAP statement:
  - For multiple executions, use the logical name defined by the ID clause of the SQLEXEC statement. ID is required for multiple executions of a procedure.
  - For a single execution, use the actual stored procedure name.

<parameter>: Valid values are one of the following:

- The name of the parameter in the stored procedure or query from which the data will be extracted and passed to the column map
- RETURN_VALUE if values returned by a stored procedure or query are extracted

The parameter value can be extracted depending on the following circumstances:

1. The stored procedure or query executes successfully.
2. The stored procedure or query results are not yet expired.

Rules for determining expiration are defined by the `SQLEXEC EXEC` option. When a value cannot be extracted, the `@GETVAL` function results in a “column missing” condition. Usually this means that the column is not mapped. You can also use the `@COLTEST` function to test the result of the `@GETVAL` function to see if the column is missing, and map an alternative value if desired.

Summary

In this lesson, you should have learned how to:

- Map columns between different schemas
- Select and filter data for replication
- Use built-in data transformation functions
- Use **SQLEXEC** to interact directly with a database

Practice 6: Overview

The practices for this lesson cover the following topics:

- Setting up column mapping between dissimilar source and target tables
- Setting up data selection in both Extract and Replicat streams
- Applying data transformation functions
- Using **`SQLEXEC`** to invoke a stored procedure

7

Custom Behavior Through User Exits

Objectives

After completing this lesson, you should be able to:

- Describe the user exits facility
- Compare and contrast user exits with natively available column lookups and conversion functions
- Analyze the steps required to develop a user exit routine
- Describe the Oracle GoldenGate Application Adapters for JMS and Flat File user exit, which is an example of a custom implementation of the user exit framework

User Exits: Overview

User exits:

- Have custom logic written in C, C++, or Java by the customer
- Are invoked at different points in Extract or Replicat processing (through the **CUSEREXIT** parameter)
- Allow you to extend or customize the functionality of data movement and integration beyond what is supported through mapping, functions, or **SQLEXEC**
- Can perform an unlimited number of functions

At different points during Extract and Replicat processing, routines that you create in C can be invoked to perform an unlimited number of functions.

Oracle GoldenGate for Oracle includes a C and C++ development environment that can be used to develop user exits.

To develop user exists in Java, an additional module (Oracle GoldenGate for Java) is available.

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 GoldenGate. User exits can be a better alternative to the built-in functions because, with a user exit, data is processed once (when extracted) rather than twice (extracted, and then read again to perform the transformation).

User exits cannot be used for tables that are being processing by a Data Pump Extract in pass-through mode.

User Exits: Applications

User exits:

- Perform arithmetic operations or data transformations beyond those provided with GoldenGate built-in functions
- Perform additional table lookups or clean up invalid data
- Respond to events in customized ways, for example, by sending a formatted email message or paging a supervisor based on some field value
- Accumulate totals and gather statistics
- Perform conflict detection or custom handling of errors or discards
- Determine the net difference in a record before and after an update (conflict-resolution technique)

Additional applications of user exits:

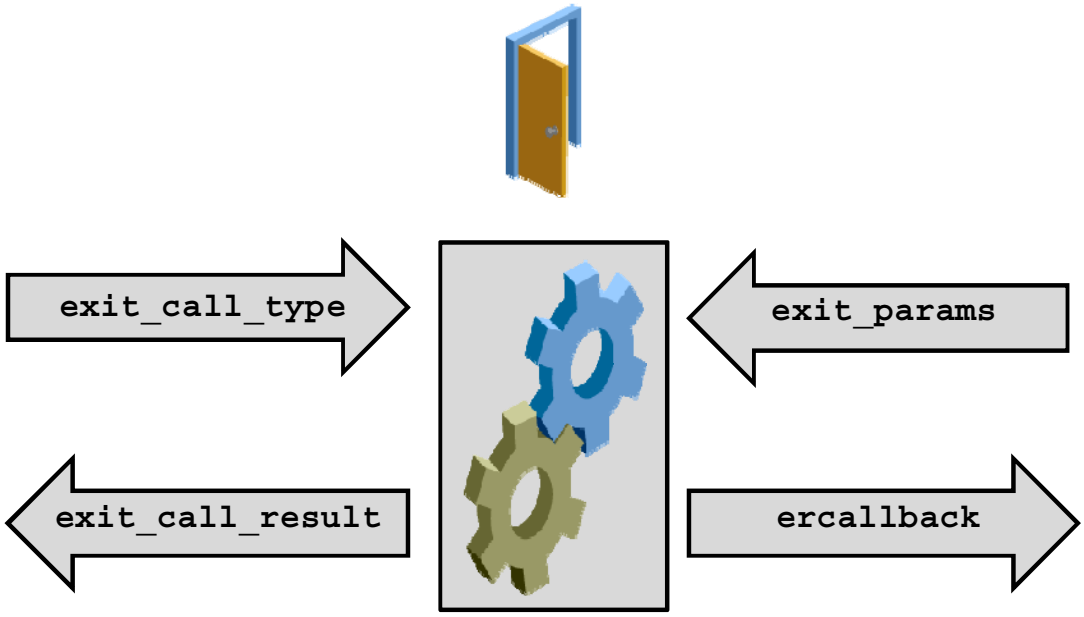
- Implement record archival functions offline
- Accept or reject records for extraction or replication based on complex criteria
- Normalize a database during conversion

User Exits: High-Level Processing Logic

The high-level processing logic:

- Accepts different events and information from Extract or Replicat
- Passes the information to the appropriate paragraph or routine for processing
- Returns a response and information to the caller

User Exits: Parameters



- **EXIT_CALL_TYPE** indicates when, during processing, the Extract or Replicat process calls the user exit: at start processing, stop processing, begin transaction, end transaction, process record, process marker, discard record, fatal error, or call result.
- **EXIT_CALL_RESULT** provides a response to the routine: OK, ignore, stop, abend, or skip record.
- **EXIT_PARAMS** supplies information to the routine: calling program path and name, function parameter, or “more records” indicator.
- **ERCALLBACK** implements a callback routine. Callback routines retrieve record and GoldenGate context information and modify the contents of data records.

User Exits: Parameters

EXIT_CALL_TYPE indicates the processing point of the caller, and determines the type of processing to perform. Extract and Replicat call the shell routine with the following calls:

- **EXIT_CALL_START:** Invoked at the start of processing. The user exit can perform initialization work.
- **EXIT_CALL_STOP:** Invoked before the caller stops or ends abnormally. The user exit can perform completion work.

- **EXIT_CALL_BEGIN_TRANS:** In Extract, invoked just before the output of the first record in a transaction; in Replicat, invoked just before the start of a transaction
- **EXIT_CALL_END_TRANS:** In Extract and Replicat, invoked just after the last record in a transaction is processed
- **EXIT_CALL_CHECKPOINT:** Called just before an Extract or Replicat checkpoint is written
- **EXIT_CALL_PROCESS_RECORD:** In Extract, invoked before a record buffer is output to an Extract file; in Replicat, invoked just before a replicated operation is performed. This call is the basis of most user exit processing.
- **EXIT_CALL_PROCESS_MARKER:** Called during Replicat processing when a marker from a NonStop server is read from the trail, and before writing to the marker history file
- **EXIT_CALL_DISCARD_RECORD:** Called during Replicat processing before a record is written to the discard file
- **EXIT_CALL_DISCARD_ASCII_RECORD:** Called during Extract processing before an ASCII input record is written to the discard file
- **EXIT_CALL_FATAL_ERROR:** Called during Extract or Replicat processing just before GoldenGate terminates after a fatal error
- **EXIT_CALL_RESULT:** Set by the user exit routines to instruct the caller on how to respond when each exit call completes (see the following)

EXIT_CALL_RESULT is set by the user exit routines and instructs the caller on how to respond when each exit call completes. The following results can be specified by the operator's routines:

- **EXIT_OK_VAL:** If the routine does nothing to respond to an event, assumes OK. If the call specified `PROCESS_RECORD` or `DISCARD_RECORD` and `OK_VAL` is returned, the caller processes the record buffer returned by the user exit, and uses the parameters set by the exit.
- **EXIT_IGNORE_VAL:** Rejects records for further processing
- **EXIT_STOP_VAL:** Instructs the caller to stop immediately
- **EXIT_ABEND_VAL:** Instructs the caller to abend immediately
- **EXIT_PROCESSED_REC_VAL:** Instructs Extract or Replicat to skip the record, but updates the statistics that are printed to the export file for that table and for that operation type

EXIT_PARAMS supplies information to the user exit routine, such as the program name and user-defined parameters. You can process a single data record multiple times:

- **PROGRAM_NAME:** Specifies the full path and name of the calling process (for example, `\ggs\extract` or `\ggs\replicat`). Use this parameter when loading a GoldenGate callback routine by using the Windows API or to identify the calling program when user exits are used with both Extract and Replicat processing.
- **FUNCTION_PARAM:** Allows you to pass a parameter that is a literal string to the user exit. Specify the parameter with the `EXITPARAM` option of a `TABLE` or `MAP` statement. `FUNCTION_PARAM` can also be used at exit call startup to pass the parameters that are specified in the `PARAMS` option of the `CUSEREXIT` parameter.

- **MORE_RECS_IND:** Is set on return from an exit; for database records, determines whether Extract or Replicat processes the record again. This allows the user exit to output many records per record processed by Extract, a common function when converting Enscribe to SQL (data normalization). To request the same record again, set MORE_RECS_IND to CHAR_NO_VAL or CHAR_YES_VAL.

ERCALLBACK executes a callback routine. A user callback routine retrieves context information from the Extract or Replicat process and its context values, including the record itself, when the call type is one of the following:

- EXIT_CALL_PROCESS_RECORD
- EXIT_CALL_DISCARD_RECORD
- EXIT_CALL_DISCARD_ASCII_RECORD

Syntax: ERCALLBACK (<function_code>, <buffer>, <result_code>);

- **<function_code>:** The function to be executed by the callback routine. The user callback routine behaves differently based on the function code that is passed to the callback routine. Although some functions can be used for both Extract and Replicat, the validity of the function in one process or the other is dependent on the input parameters that are set for that function during the callback routine.
- **<buffer>:** A void pointer to a buffer that contains a predefined structure associated with the specified function code
- **<result_code>:** The status of the function executed by the callback routine. The result code returned by the callback routine indicates whether or not the callback function was successful.

See the *Oracle GoldenGate Reference Guide* for function codes and result codes.

User Exits: Implementation

- On Windows: Create a DLL in C and export a routine to be called from Extract or Replicat.
- On UNIX: Create a shared object in C and create a routine to be called from Extract or Replicat.
- The routine must accept the following parameters:
  - **EXIT_CALL_TYPE**
  - **EXIT_CALL_RESULT**
  - **EXIT_PARAMS**
- In the source for the DLL or shared object, include the **usrdecs.h** file (in the GoldenGate installation directory).
- Call the **ERCALLBACK** function from the shared object to retrieve record and application context information.

To implement user exits in Windows or UNIX, perform the following:

1. On Windows, create a user exit DLL in C and export a routine to be called from Extract or Replicat.

   On UNIX, create a shared object in C and create a routine to be called from GoldenGate. This routine is the communication point between Extract or Replicat and your routines.

You can define the name of the routine, but it must accept the following user exit parameters:

- `EXIT_CALL_TYPE`
- `EXIT_CALL_RESULT`
- `EXIT_PARAMS`

Example export syntax for the `MyUserExit` routine:

```
__declspec(dllexport) void MyUserExit (
exit_call_type_def exit_call_type,
exit_result_def *exit_call_result,
exit_params_def *exit_params)
```

2. In the source for the DLL or shared object, include the `usrdecs.h` file. This file contains type definitions, return status values, callback function codes, and several other definitions.
3. Include callback routines in the user exit when applicable. Callback routines retrieve record and application context information, and modify the contents of data records.

Extract and Replicat export an `ERCALLBACK` function to be called from the user exit routine. The user exit must explicitly load the callback function at run time by using the appropriate Windows or UNIX API calls.

User Exits: Samples

- Sample user exit files are located in **<GoldenGate installation directory>/UserExitExamples**.
- Each directory contains the **.c** file, as well as makefiles and a **readme.txt** file.

`exitdemo.c` shows how to initialize the user exit, issue callbacks at given exit points, and modify data. The demonstration is not specific to any database type.

`exitdemo_passthru.c` shows how the `PASSTHRU` option of the `CUSEREXIT` parameter can be used in an Extract Data Pump.

`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.

User Exits: Calling

- You can call a user exit from Extract or Replicat by using the **CUSEREXIT** parameter.
- Syntax:

```
CUSEREXIT <DLL or shared object name> <routine name>
[, PASSTHRU]
[, INCLUDEUPDATEBEFORES]
[, PARAMS "<startup string>"]
```

- Examples:

```
CUSEREXIT userexit.dll MyUserExit
CUSEREXIT userexit.dll MyUserExit, &
  INCLUDEUPDATEBEFORES, PASSTHRU, &
  PARAMS "init.properties"
```


- **<DLL or shared object name>:** Is the name of the Windows DLL or UNIX shared object that contains the user exit function
- **<routine name>:** Is the name of the exit routine to be executed
- **PASSTHRU:** Is valid only for an Extract Data Pump. It assumes that no database is required, and that no output trail is allowed. It expects that the user exit will perform all the processing and that Extract will skip the record. Extract will perform all the required data mapping before passing the record to the user exit. Instead of a reply status of `EXIT_OK_VAL`, the reply will be `EXIT_PROCESSED_REC_VAL`. All process statistics are updated as if the records were processed by GoldenGate.
- **INCLUDEUPDATEBEFORES:** Passes the before images of column values to a user exit. When using this parameter, you must explicitly request the before image by setting the `requesting_before_after_ind` flag to `BEFORE_IMAGE_VAL` within a callback function that supports this flag. Otherwise, only the after image is passed to the user exit. By default, GoldenGate works with only after images.

When using INCLUDEUPDATEBEFORES for a user exit that is called from a Data Pump or from Replicat, always use the GETUPDATEBEFORES parameter for the primary Extract process, so that the before image is captured, is written to the trail, and causes a process record event in the user exit. In a case where the primary Extract also has a user exit, GETUPDATEBEFORES causes both the before image and the after image to be sent to the user exit as separate EXIT_CALL_PROCESS_RECORD events.

If the user exit is called from a primary Extract (the one that reads the transaction log), only INCLUDEUPDATEBEFORES is needed for that Extract. GETUPDATEBEFORES is not needed in this case, unless other GoldenGate processes downstream need the before image to be written to the trail. INCLUDEUPDATEBEFORES does not cause before images to be written to the trail.

- **PARAMS "<startup string>":** Passes the specified string at startup; can be used to pass a properties file, startup parameters, or other string. The string must be enclosed within double quotation marks. Data in the string is passed to the user exit in EXIT_CALL_START exit_params_def.function_param. If no quoted string is specified with PARAMS, exit_params_def.function_param is NULL.

Simple User Exit

```
void MyUserExit (exit_call_type_def exit_call_type,
                 exit_result_def    *exit_call_result,
                 exit_params_def    *exit_params)
{
   switch(exit_call_type)
   {
        case EXIT_CALL_START:
            output_msg ("\nUser exit: EXIT_CALL_START.
                         Called from program: %s\n",
                          exit_params->program_name);
            break;
        case EXIT_CALL_STOP:
            output_msg ("\nUser exit:EXIT_CALL_STOP.\n");
   }
   *exit_call_result = EXIT_OK_VAL;
}
```


A user exit routine is called repeatedly by the Extract or Replicat processes. The first thing that a user exit must do is to find out at what stage of the process the user exit was called. This is accomplished by using a cascading switch statement, which tests the call type, and acts accordingly.

One of the most important call types is `EXIT_CALL_PROCESS_RECORD` (not shown in the example in the slide), which is invoked by Extract before a record buffer is output to an Extract file. Within Replicat processing, `EXIT_CALL_PROCESS_RECORD` is invoked just before a replicated operation is performed.

Using Callback Routines

- Callback routines allow user exits to access information stored in the Extract or Replicat process.
- **ERCALLBACK** executes a callback routine. A user callback routine retrieves context information from the Extract or Replicat process and sets context values.
- Use the **GET_ENV_VALUE** function to return information about the Oracle GoldenGate environment. After the callback returns, the **getenv_value_def** structure contains the required information.

Syntax:

```
ERCALLBACK (GET_ENV_VALUE, &env_ptr, &result_code);
Structure definition:
typedef struct
{
    char *information_type; /* Type of getenv wanted PASSED IN BY USER */
    char *env_value_name;   /* label of getenv wanted PASSED IN BY USER */
    char *return_value;     /* return value of getenv */
    long max_return_length; /* Maximum buffer length */
    long actual_length;     /* Actual buffer length */
    short value_truncated;  /* Was value truncated? */
} getenv_value_def;
```

The structure requires two fields to be populated: `information_type` and `env_value_name`. The code stores `GGENVIRONMENT` in the `information_type` field and `"GROUPTYPE"` in the `env_value_field`. After the callback returns, the value fetched by the callback will be in the `return_value` field of the `getenv_value_def` structure.

Oracle GoldenGate Adapters for JMS

Oracle GoldenGate Application Adapters for JMS and Flat File:

- This is a software component that implements a specific user exit (`libggjava_ue.so` or `libggjava_ue.dll`).
- Through the Oracle GoldenGate Java API, the transactional data that is captured by GoldenGate can be delivered to targets other than a relational database, such as Java Message Service (JMS), by writing files to disk or integrating with a custom application's Java API.
- Oracle GoldenGate for Java provides the ability to execute code written in Java from the Oracle GoldenGate Extract process. This Java framework communicates with the user exit through the Java Native Interface (JNI).

The Oracle GoldenGate Application Adapters for JMS and Flat File is a bridge between the C/C++ world and Java. The Oracle GoldenGate core functionality is implemented by using C++. In situations where data captured from databases must be made available to applications written in Java, the Application Adapters for JMS and Flat File can be used to integrate the two environments.

Summary

In this lesson, you should have learned how to:

- Describe the user exits facility
- Evaluate the use of user exits as an alternative or a complement to the column lookup and conversion functions that are available natively in Oracle GoldenGate
- Analyze the steps required to develop a user exit routine
- Describe the Oracle GoldenGate Application Adapters for JMS and Flat File user exit, which is an example of a custom implementation of the user exit framework

Practice 7: Overview

- This practice covers creating a user exit that uses Oracle GoldenGate Application Adapters for JMS and Flat File to capture all DML statements on the replication source, thereby making them available to a JMS queue as XML messages.
- The JMS broker that is used in the practice is Apache ActiveMQ.

8

Configuring Zero Down–Time Migration Replication

Objectives

After completing this lesson, you should be able to:

- Plan and configure a zero down–time migration by using Oracle GoldenGate
- Perform an upgrade involving a cutover to a new database
- Interrupt the upgrade and fall back to the old configuration in minimal time, without data loss

Software Upgrade Issues

- Oracle GoldenGate can be configured to mitigate the challenges involved in upgrading by allowing the new, upgraded database platform to be ready ahead of time, while being synchronized with the old and still running environment, until the application software is also upgraded.
- Conversely, if (during cutover) unforeseen circumstances force a fallback to the old database, switching back is still possible because the old database was kept synchronized with the new environment during the interim period.

Many IT companies are trapped with old releases of software, often paying premium maintenance support for obsolete versions of the database, simply because they cannot tolerate the down time associated with a complex upgrade. Zero down–time migration allows the IT shops that are running Oracle GoldenGate to migrate to new releases, thus mitigating all risks associated with untested software.

When to Use Zero Down–Time Migration

- Application upgrade
- Database upgrade
- Migration to or from a different database
- Migration to or from a different operating system
- Upgrade of operating system
- Routine patch maintenance

Zero down–time migration can be used in different scenarios, whenever the OS, RDBMS, or application is either re-hosted or upgraded. Major changes, such as porting the current application to a different database or a different operating system, typically require the ability to rapidly back out, while at the same time ensuring that no data is lost during the interim period, when the application is already using the upgraded database or OS.

Zero Down–Time Requirements

For a period of time after the migration cutover, the old database must be kept synchronized with the new database until you are certain that a fallback will not occur.

- Data in the old database must be kept current with the new database, and there should no replication lag at the time of cutover or fallback. Before the initial cutover, some lag can be tolerated, but that lag must be eventually eliminated before application cutover.
- After cutover to the new database, the database roles are reversed. The new database becomes the replication source and the old database becomes the target. This enables the old database to quickly become operational if the upgrade is aborted.

ORACLE

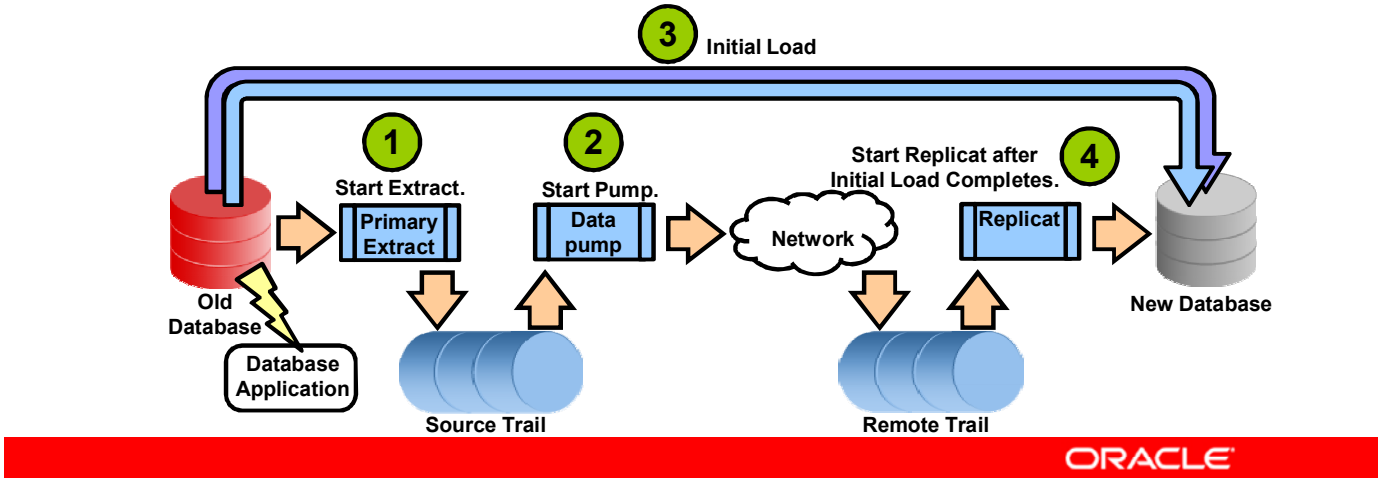
The underlying topology for zero down–time migration is one-way replication, with a few changes for zero down–time migration. One-way replication is the simplest topology, and is often used for reporting or query offloading.

In this lesson, the one-way replication topology is used for zero down–time migration. Data is replicated from a single source database to a single target database in only one direction at a time. Changes to database data are made only at the source database, and then replicated to the target database. The source database is the old database from which you are migrating. After the migration cutover to the new database, the replication is reversed and data is replicated back from the new database to the old database to allow for migration fallback. This backward replication can stay in place for as long as needed. After you are sure that you will stay on the new database and fallback is not needed, you can turn off the backward replication.

Seven-Step Approach to Zero Down–Time Configuration: Phase 1

A zero down–time configuration can be achieved with a two-phase, seven-step procedure:

- **Phase 1 (steps 1–4):** Configure and start the Extract, Pump, and Replicat processes from the old database to the new database.
- **Phase 2 (steps 5–7):** Configure (but do not start) the fallback Extract, Pump, and Replicat processes.

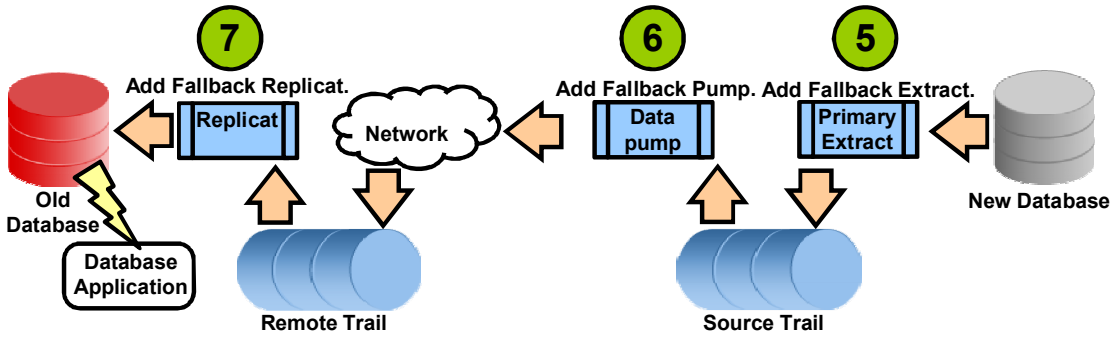


When initial changes have been applied and no Oracle GoldenGate lag is left, the databases are fully synchronized. At this point, the primary and Data Pump Extracts and Replicat can continue to run and keep the old and new databases synchronized in real time with the ongoing changes. The new database is ready and waiting for the application to cutover for the migration.

When you finish synchronizing the old and new databases, you should perform data validation. There may be cases where data is not replicated properly, or perhaps there is an error in the configuration. You can do data validation manually with `SELECT` queries for row counts and some selected data values, or you can use an automated tool such as Oracle GoldenGate Veridata.

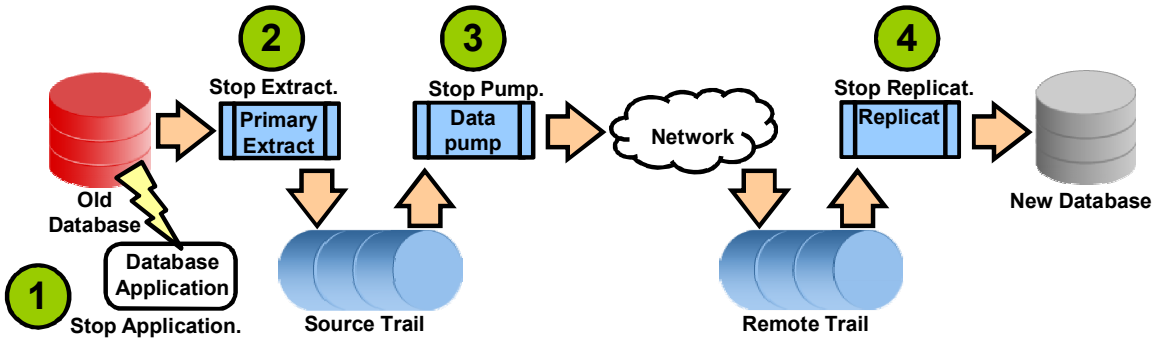
Phase 2: Preparing Fallback

- After you are successful in replicating from the old database to the new database, the next step is to add the fallback Extract.
- For now, you add only the Extract. Later, you start it after the application cutover activities.



Preparing for Cutover

- Stop the application. Make sure that there is no lag before stopping the Extract process.
- Make sure that there is no lag. Then stop the Data Pump.
- Stop the Replicat process.

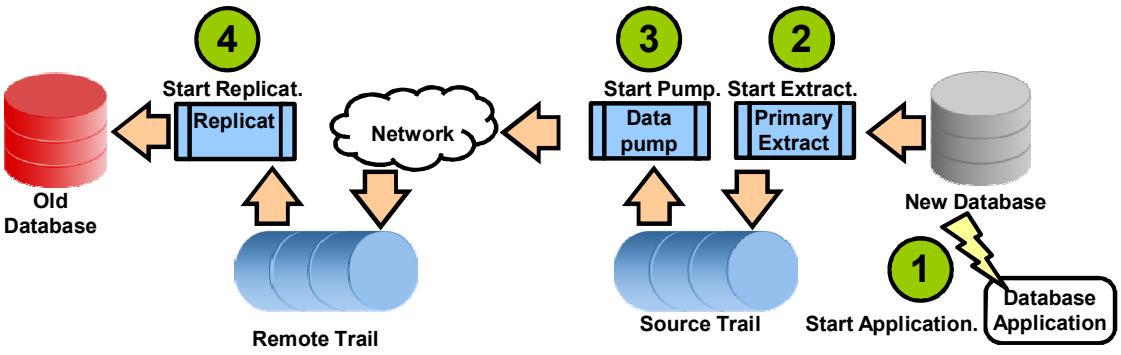


ORACLE

During migration cutover, you switch your database application from the old database to the new database. After the application is cut over to the new database, replication is reversed to the old database to allow for fallback. The cutover preparation implies a graceful termination of the replication environment, ensuring that there are no pending transactions within the replication system before the cutover is performed.

Performing Cutover

- Start the application, which now points to the new database.
- Start the Extract and the Data Pump processes on the new hosting computer.
- Start the Replicat process, which synchronizes the old database with the new, currently used database.



Now the old database is no longer being used by the application and replication from the old database to the new database is stopped. However, you want to maintain synchronization between the new database (now being updated directly by the application) and the old database to support fallback. To achieve this, you start the primary Extract, the Data Pump, and the Replicat processes that were previously configured.

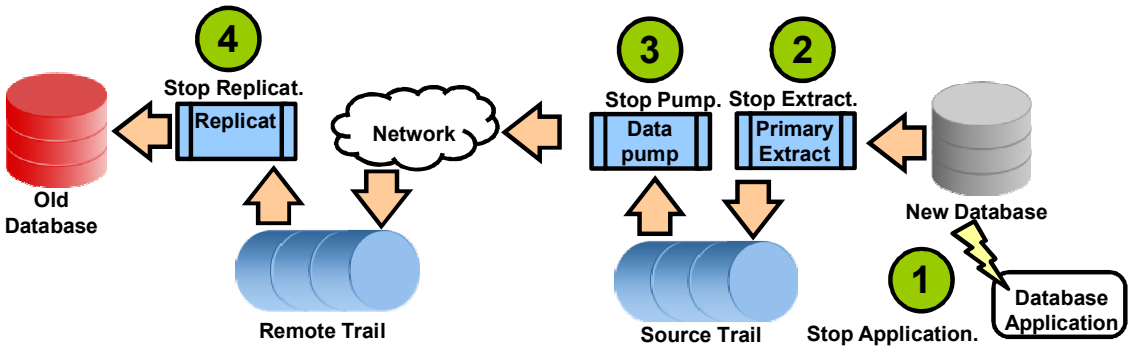
Deciding to Fall Back

- Sometimes, in spite of all tests and hard work, a decision must be made to abandon the upgrade and to re-attempt it later, when issues have been resolved.
- The zero down–time migration option acts like an insurance policy, guaranteeing that you can recover from an abnormal situation with no or minimal damage.
- With careful planning and adherence to best practices, it is possible to fall back to a pre-migration working system in minimal time, without incurring data loss.

The inevitable down time that users experience while the fallback procedure takes place is directly proportional to the lag time between the source and the target databases. Minimizing the lag between the two replication ends is the goal if fallback and the eventual new migration attempt must be carried out in a shortened period of time.

Performing Migration Fallback

- Stop the application. Check that no lag exists before stopping the primary Extract.
- Verify that no lag remains before stopping the Data Pump.
- On the node that is hosting the old database, verify that no lag persists. Then stop the Replicat process.

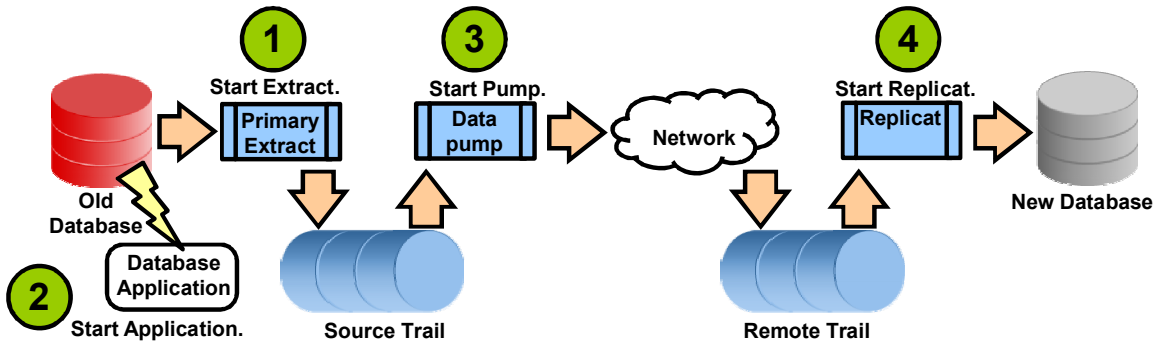


Stop the application that is running in the new database. At this time, be sure to leave the Local Extract running so that it can capture any remaining database transactions. After verifying that there is no lag, stop the primary Extract on the new database, as well as the Data Pump and the Replicat.

Important: Use the GGSCI command `LAG EXTRACT <extract name>` to make sure that you do not have pending transactions before stopping the various Oracle GoldenGate processes.

Performing Migration Fallback

- Start the application that now points to the old database.
- Start the Extract and the Data Pump processes.
- On the remote node that is hosting the replicated new database, start the Replicat process.



After you perform all the tasks to accomplish a migration fallback, you are back in the same situation as when you started. The application is now up and running in the old database, and the new database is kept synchronized with the old database through Oracle GoldenGate replication. You can now investigate and remedy the causes of the forced upgrade rollback before attempting it again.

Summary

In this lesson, you should have learned how to:

- Plan and configure a zero down–time migration by using Oracle GoldenGate
- Perform an upgrade that involves a cutover to a new database
- Interrupt the upgrade and fall back to the old configuration in minimal time, without data loss

Practice 8: Overview

This practice covers the following topics:

- Configuring and performing a zero down–time migration
- Configuring the fallback
- Performing migration cutover
- Simulating an aborted migration to exercise fallback

9

Bidirectional Replication: Two-Node Configuration

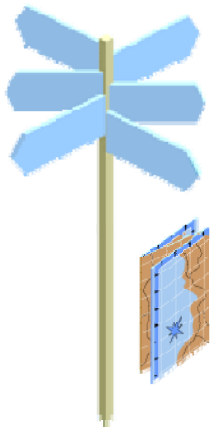
Objectives

After completing this lesson, you should be able to:

- Analyze the advantages and disadvantages of an active-active replication configuration
- Describe the requirements that must be satisfied to implement a bidirectional replication solution
- Configure loop prevention
- Handle table truncates
- Implement basic data conflict resolution routines
- Set up DDL replication

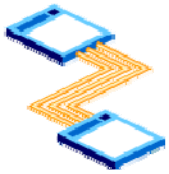
Roadmap

- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



Bidirectional Replication: Advantages

- Load sharing
- Fault tolerance:
  - In case of instance outage, user connections can be rerouted to the surviving instance.
- Business continuity:
  - Critical business functions are available to customers, suppliers, and regulators while the failed instance is restored.
  - Crucial service-level agreements are fulfilled, even in case of unscheduled database instance outage.
- Geographically separated data centers:
  - 24 × 7 cycle, with *follow-the-sun* model
  - Protection from regional disasters
  - Local access for better performance



ORACLE

Transactionally intense applications can benefit from splitting the load across two instances, which provides better performance.

An active-active replication configuration provides the obvious benefit of redundancy and fault tolerance. An application can survive the temporary outage of one of the two instances with minimal disruption:

- Database connection pools can rapidly, and mostly automatically, reconnect to the surviving instance.
- Only in-process transactions on the failed instance must be rolled back. However, users can resubmit the failed transactions on the surviving instance.
- If the surviving instance can withstand the additional traffic without noticeable performance degradation, business operations can continue without serious disruption until the failed instance is restored.

Bidirectional Replication: Disadvantages

- Increased complexity:
  - Additional administrative burden
- Latency in data propagation:
  - Data conflicts
  - Constant maintenance of conflict resolution
- Nontransparent deployment:
  - Need for applications to be replication-aware
  - Mostly incompatible with third-party packaged software

An Oracle GoldenGate active-active configuration is by its very nature asynchronous, which implies a certain latency in the propagation of data between the two instances being replicated.

During the latency period, data conflicts can materialize when two or more transactions that affect the same row (or set of rows) occur in rapid succession.

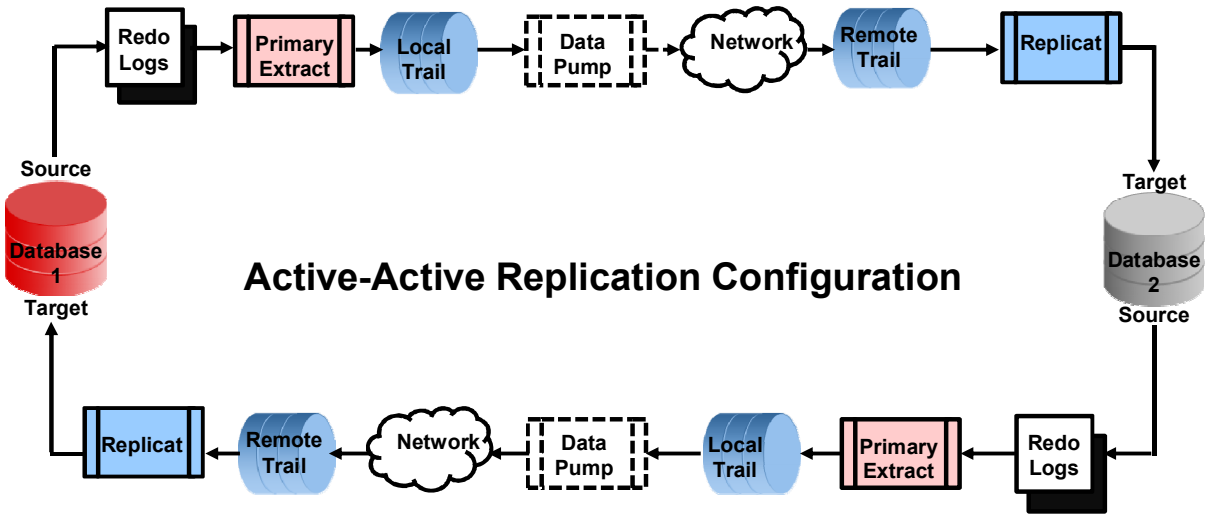
Data conflicts must be detected and resolved, and sometimes the logic to resolve the data conflict is convoluted and difficult to test.

Procedures must be devised to deal with conflicts that are detected but not resolved by the automated routines. These procedures must be run daily.

Active-active replication is not recommended for use with commercially available packaged business applications. This is because they might contain objects and data types that are not supported by Oracle GoldenGate, or they might perform automatic DML operations that you cannot control, but that will be replicated by Oracle GoldenGate and cause conflicts when applied by Replicat.

Data conflict avoidance is a much better solution than data conflict resolution. It requires the applications to be replication-aware.

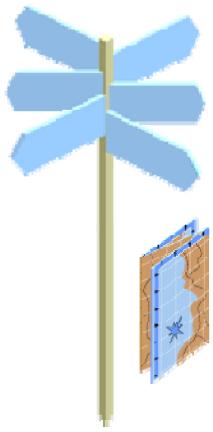
Bidirectional Configuration



The diagram in the slide shows all the basic components of an active-active replication configuration. The lower part of the diagram is the mirror image of the upper part.

Roadmap

- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



Bidirectional Replication Considerations

- Consider expected transaction load.
  - Consistent
  - Moderate volume evenly spread across replicated objects
- Challenges of bidirectional replication:
  - Long-running transactions
  - Tables with large number of columns changing
  - Tables with LOBs

Requirements

Crucial preliminary tasks required for a bidirectional configuration:

- Create a checkpoint table on both databases.
- Establish the trusted source.
- Handle primary keys and sequences.
- Manage triggers and cascade deletes.
- Implement loop detection and prevention.
- Implement data conflict detection and resolution.

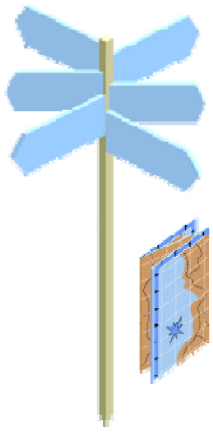
A checkpoint table is used by the Replicat process to store the known positions in the trail from which to start after an expected or unexpected (abrupt) shutdown. A dedicated schema for Oracle GoldenGate should be used to store all objects needed for replication.

One of the two databases must be designated as the trusted source. Usually, this database is the one that originates the replication (that is, the initial load source). Frequent backups of the trusted source should be taken.

As a preliminary task, database sequences should be created to generate alternate numbers on each database to make sure that no number overlap occurs.

Roadmap

- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



Data Replication Looping

The following example sequence demonstrates the problem that occurs without loop detection:

1. A row is updated on System A.
2. The update operation is captured and sent to System B.
3. The row is updated on System B.
4. The update operation is captured and sent to System A.
5. The row is updated on System A.

Without loop detection, this loop continues endlessly.

ORACLE

To avoid looping, Replicat's operations must be prevented from being sent back to the source table by Extract. This is accomplished by configuring Extract to ignore transactions issued by the Replicat user. The example in the slide illustrates why data looping must be prevented.

Data Replication Looping

The following is another scenario that demonstrates the problem that occurs without loop detection:

1. A row is deleted on System A.
2. The deletion is captured and sent to System B.
3. The row is deleted on System B.
4. The deletion is captured and sent to System A.
5. The deletion fails on System A because the row is not found.

Loop Prevention

- To avoid replication loops:
  - The Replicat process must use a dedicated Oracle user
  - All replication for transactions executed by the Replicat user must be suppressed
- To exclude the Replicat user from propagating transactions:
  - All Extract processes on the source database must contain the **`TranLogOptions ExcludeUser <username>`** or **`TranLogOptions ExcludeUserID <user id>`** clause:

```
TranLogOptions ExcludeUser ggs_user
```

Or

```
TranLogOptions ExcludeUserID 86
```

ORACLE

`ExcludeUserID` is available in Oracle Database 10*g* and later releases.

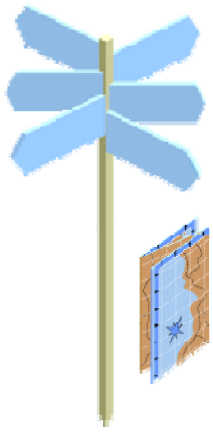
To determine the user ID of the user running Replicat transactions, connect to the Oracle RDBMS using the Replicat user and issue the following statement:

```
select UID from dual;
```

Note: Instead of using `TranLogOptions`, you can create a trace table with the `ADD TRACETABLE` command in GGSCI. However, this method uses more processing overhead than `TranLogOptions`.

Roadmap

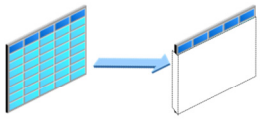
- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



Configuring Table Truncates

Bidirectional replication of truncates is not supported.

- All table truncates must occur on only one database, while data is replicated in both directions.
- Select one of the two databases as the “trusted source” where truncates are allowed.
- Use Oracle Security to prevent truncates on the other database.
- Configure the **GETTRUNCATES** and **IGNORETRUNCATES** parameters in the Extract and Replicat parameter files.



ORACLE

`TRUNCATE` commands must originate from only one database—and from the same database each time. To prevent accidental truncates, configure all database roles so that they cannot execute `TRUNCATE` on the database that is not designated to allow truncates.

On the system where the `TRUNCATE` command is permitted, you should configure the Extract and Replicat parameter files to contain the `GETTRUNCATES` parameter.

On the other system, you should configure the Extract and Replicat parameter files to contain the `IGNORETRUNCATES` parameter.

Modifying Triggers and Cascade Deletes

- Triggers and `ON CASCADE DELETE` constraints generate DML operations that could conflict with replication.
  - Modify triggers to ignore DML operations that are carried out by Replicat.
  - Disable `ON CASCADE DELETE` constraints and use a trigger instead.
- Database-generated sequential values (for example, sequences) must not be replicated in a bidirectional setup.
  - The range of generated values must be different on each database and cannot overlap.

To suppress triggers for certain Oracle database versions, you can use the `DBOPTIONS` parameter with the `SUPPRESSTRIGGERS` option to disable the triggers for the Replicat session.

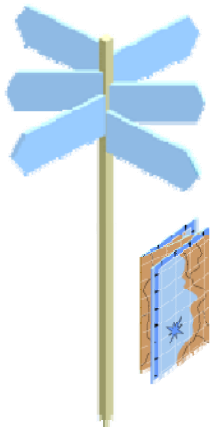
`SUPPRESSTRIGGERS` was supported in Oracle Database 10*g*; its support was dropped in Oracle Database 11*g* R1 but was reintroduced in 11*g* R2.

Instead of an `ON DELETE CASCADE` referential integrity setup, create a `BEFORE` trigger so that the child tables are deleted before the delete operation is performed on the parent table. This reverses the logical order of a cascaded delete, but is necessary so that the operations are replicated in the correct order to prevent “table not found” errors on the target.

In an active-active replication configuration, the sequences that are used to produce primary keys should be created to generate alternate numbers on each database.

Roadmap

- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



Bidirectional Replication and Data Conflicts

Oracle GoldenGate replicates asynchronously.

- Even if a subsecond lag is allowed to occur between the source and the target, this interval is sufficient to produce data conflicts.
- Conflicts can be avoided by using workload-partitioning and data-segmentation techniques.
- The Extract process can capture a before image and an after image of updates, which can be compared to determine if a conflict occurred.
- Several built-in commands are available to enable the Replicat process to detect and resolve data conflicts.
- More complex conflict resolution is available using **`SQLEXEC`** and user exits.

ORACLE

Conflicts can be minimized (but not completely avoided) by configuring a very low latency between the replication source and the target.

Data conflict avoidance must be implemented at the application level (that is, it is not transparent to the application) and implies having certain data written in only one partition that belongs to one of the two databases (and never to the other). Other techniques involve maintaining a clear segmentation at the column level, attributing data ownership to only one application, and always updating on a specific instance.

Oracle GoldenGate provides built-in “Conflict Detection and Resolution” (CDR) functions, which are the building blocks for the creation of simple conflict detection and resolution routines.

CDR Facility

CDR relies on four keywords:

- **`GetUpdateBefores`:** Specified in the Extract; forces the capture of the before image
- **`GetBeforeCols`:** Specified in the Extract; lists the columns for which the before image must be captured
- **`CompareCols`:** Specified in the Replicat; lists the columns included in the `WHERE` clause with their before image
- **`ResolveConflict`:** Specifies how Replicat handles conflicts on operations that are performed on the tables by using the **`MAP`** statement

`The GetUpdateBefores` and `GetBeforeCols` keywords are specified in the Extract parameter files.

`The CompareCols` and `ResolveConflict` keywords appear in the Replicat parameter files.

The CDR facility requires all four keywords to be used across the Extract/Replicat boundaries to carry out its task.

Conflict Types

In bidirectional replication configurations, three types of data conflicts can materialize:

- **Uniqueness conflicts:** These occur when Replicat applies an insert or update operation that violates a uniqueness integrity constraint.
- **Update conflicts:** These occur when Replicat applies an update that conflicts with another update to the same row.
- **Delete conflicts:** These occur when two transactions originate at different databases, and one deletes a row while the other updates or deletes the same row.

ORACLE

An example of a uniqueness conflict is when two transactions that originate from two different databases insert a row into a table with the same primary key value.

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.

In the case of delete conflicts, the row being deleted does not exist so it cannot be either updated or deleted. Replicat cannot find the row because the primary key does not exist.

Automatic Data Conflict Detection and Resolution

Using CDR, the most common data conflicts can be automatically resolved:

- A uniqueness conflict for an `INSERT`
- A "no data found" conflict for the following:
  - 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
  - For an `UPDATE` when the row does not exist
  - 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
  - For a `DELETE` when the row does not exist

`CompareCols` is used as a parameter in the `MAP` statement and determines the DML statement that you are checking for:

```
ON UPDATE
ON DELETE
```

`CompareCols` also determines the columns that should be used for the comparison. The syntax is:

```
MAP <source table>, TARGET <target table>,
COMPARECOLS(
{ON UPDATE | ON DELETE}
{ALL | KEY | KEYINCLUDING (<col>[,...]) | ALLEXCLUDING (<col>[,...])
}
[,...]
)
```

ResolveConflict Arguments

Argument	Description
INSERTROWEXISTS	An inserted row violates a uniqueness constraint on the target.
UPDATEROWEXISTS	An updated row exists on the target, but one or more columns have a before image in the trail that is different from the current value in the database.
UPDATEROWMISSING	An updated row does not exist in the target.
DELETEROWEXISTS	A deleted row exists in the target, but one or more columns have a before image in the trail that is different from the current value in the database.
DELETEROWMISSING	A deleted row does not exist in the target.

ResolveConflict Syntax

```
MAP <source table>, TARGET <target table>,
RESOLVECONFLICT (
{INSERTROWEXISTS | UPDATEROWEXISTS |UPDATEROWMISSING |
DELETEROWEXISTS | DELETEROWMISSING}
( {DEFAULT | <resolution name>},
{USEMAX (<res_col>) |
USEMIN (<res_col>) |
USEDELTA | DISCARD | OVERWRITE | IGNORE})
[, COLS (<col>[,...])]
)
[RESOLVECONFLICT (, ...)]
```

Conflict Resolution Methods

- CDR offers several built-in methods for automatic resolution:
  - **USEMAX**
  - **USEMIN**
  - **USEDELTA**
  - **OVERWRITE**
  - **IGNORE**
  - **DISCARD**
- Some of the preceding methods (**OVERWRITE** and **DISCARD**) should be used with caution because they could lead to data divergence.

The goal of every replication topology is data convergence, which means that eventually all nodes participating in replication will contain the same data. The goal of data conflict resolution is, therefore, to achieve data convergence. Conflict resolution methods such as `OVERWRITE` and `DISCARD` might be easier to implement, but they can introduce discrepancies that are not easily reconcilable, thereby leaving the replication environment in a divergent state.

Basic CDR in Action

	Table ACCT	
Column	**Type**	**Attribute**
name	varchar2(30)	primary key
address	varchar2(100)	
salary	number	
balance	number	
updated_ts	timestamp	

The **ACCT** table, which is identically defined in both environments, is replicated in an active-active replication topology.

CDR rule for the **ACCT** table:

```
MAP west.acct, TARGET east.acct,
  COMPARECOLS(ON UPDATE ALL)
  RESOLVECONFLICT(UPDATEROWEXISTS,
  (DELTA_RESOLUTION_METHOD, USEDELTA, COLS(balance)),
  (MAX_RESOLUTION_METHOD, USEMAX(update_ts),
  COLS(name,address,salary,update_ts)),
  (DEFAULT OVERWRITE));
```

ORACLE

COMPARECOLS forces MAP to use the before image of all columns in the trail record, in the Replicat WHERE clause for updates and deletes.

The DEFAULT clause forces MAP to use all columns as the column group for all conflict types; the resolution applies to all columns.

For an INSERTROWEXISTS conflict, use the USEMAX resolution. If the row exists during an insert, use the *updated_ts* 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 *updated_ts* column as the resolution column.

If the value in the trail is greater, apply the update.

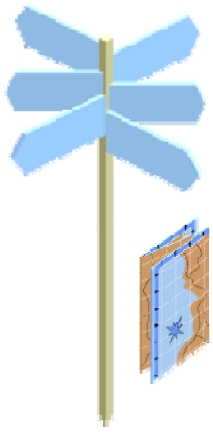
For a DELETEROWEXISTS conflict, use the OVERWRITE resolution. If the row exists during a delete operation, apply the delete.

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 DELETEROWMISSING conflict, use the DISCARD resolution. If the row does not exist during a delete operation, discard the trail record.

Roadmap

- Bidirectional replication: advantages and disadvantages
- Active-active requirements
- Loop detection and prevention
- Table truncates
- Conflict detection and resolution
- DDL replication



DDL Support in Bidirectional Configurations

Oracle GoldenGate supports DDL bidirectional replication between two systems (and two systems only).

- DDL support is configured as in master/slave topologies:
  - In both databases, the DDL objects must be created (**GGS_MARKER**, **GGS_DDL_SEQ**, **GGS_DDL_HIST**, **GGS_DDL_TRIGGER_BEFORE**, and so on).
  - To create the DDL objects, run the administrative scripts **marker_setup.sql**, **ddl_setup.sql**, **role_setup.sql**, **ddl_enable.sql**, and **ddl_pin.sql** on both databases, when connected as SYSDBA.

Best practices mandate the use of a dedicated schema to store the objects that are needed by DDL replication. If the `GGS_DDL` schema is used for this purpose, the execute privileges on `utl_file` should be granted to `GGS_DDL`:

```
GRANT EXECUTE ON utl_file to GGS_DDL;
```

The `GLOBALS` parameter file in GGSCI should indicate that `GGS_DDL` is used to replicate DDL statements:

```
GGSCHEMA GGS_DDL
```

The Oracle GoldenGate install directory contains the SQL scripts that are needed to install the required objects for DDL replication:

1. `marker_setup.sql`
2. `ddl_setup.sql`
3. `role_setup.sql`
4. `ddl_enable.sql`
5. `ddl_pin.sql`

From the Oracle GoldenGate install directory, connect to Oracle as SYSDBA by using `sqlplus`, and run the DDL scripts in the sequence shown in the slide.

Each script will prompt for the DDL schema (`GGS_DDL`).

Each user assigned to the Extract, GGSCI, and Manager processes should be granted the `GGS_GGSUSER_ROLE` role.

DDL Support in Bidirectional Configurations

- To enable DDL replication in both directions:
  - Include the **GETAPPLOPS** option for the **DDLOPTIONS** statement in the Extract parameter files (both systems)
  - Include the **GETREPLICATES** option for the **DDLOPTIONS** statement in the Extract parameter files (both systems)
  - Include the **UPDATEMETADATA** option for the **DDLOPTIONS** statement in the Replicat parameter files (both systems)
- Use **ADDTRANDATA** to add supplemental log groups automatically for objects that are created or modified through DDL statements.

The resultant `DDLOPTIONS` statements should look like the following:

- Extract (trusted source and additional instance):
  `DDLOPTIONS GETREPLICATES, GETAPPLOPS`
- Replicat (trusted source and additional instance):
  `DDLOPTIONS UPDATEMETADATA`
- To include supplemental log groups for created or modified objects:
  `DDLOPTIONS GETREPLICATES, GETAPPLOPS, ADDTRANDATA` (valid only for the Extract process)

DDL Propagation Lag

- After a DDL statement is issued on one database, you should ensure that replication is successful before issuing DML statements (or additional DDL statements) against the modified or created object.
- To avoid metadata inconsistencies, the DDL statement should be propagated by Replicat onto the target system, and also captured by the Extract on the target system, before it is safe to issue further changes on the underlying objects either created or modified via DDL statements.

ORACLE

Again, Oracle GoldenGate replication occurs asynchronously. The Oracle data dictionary should converge on both database systems before DML statements start using the new or modified objects. As a general guideline, you should allow double the time of normal latency between the two systems before using the new or modified objects.

Summary

In this lesson, you should have learned how to:

- Analyze the advantages and disadvantages of an active-active replication configuration
- Describe the requirements that must be satisfied to implement a bidirectional replication solution
- Configure loop prevention
- Handle table truncates
- Implement basic data conflict resolution routines
- Set up DDL replication

Practice 9: Overview

This practice covers the following topics:

- Configuring a fully operational active-active replication topology
- Enabling CDR-based conflict detection and resolution
- Setting up DDL replication and verifying correct propagation of DML and DDL statements

10

Conflict Detection and Resolution: Custom Techniques

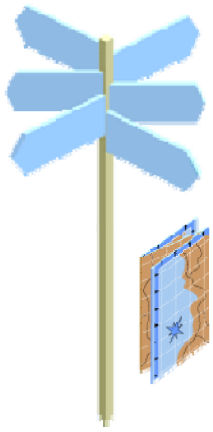
Objectives

After completing this lesson, you should be able to:

- Minimize or prevent data conflicts by using conflict avoidance techniques
- Implement advanced data conflict resolution methods

Roadmap

- Data conflict avoidance techniques
- Advanced conflict resolution methods



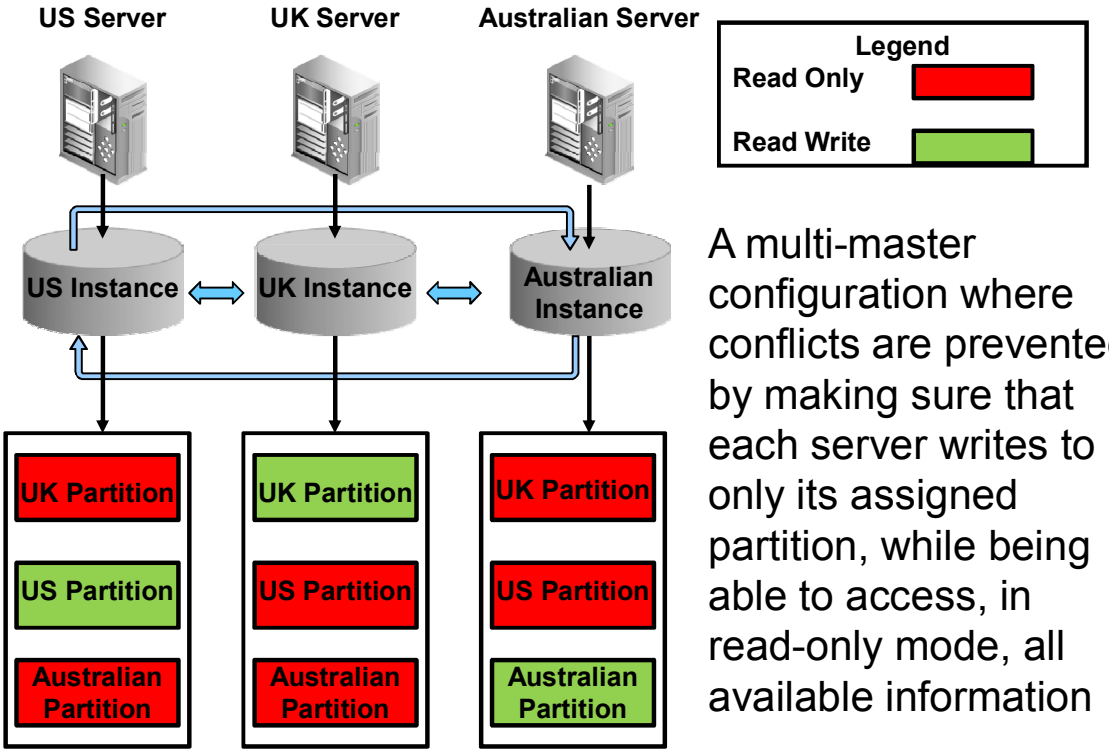
Avoiding, Rather than Resolving, Conflicts

- The need for replication sometimes arises after the application has been deployed. In this case, replication is "retrofitted" into an already working solution, and must be implemented transparently to the application.
- When replication is designed from the ground up, several techniques can be used to make the application replication-aware and to prevent data conflicts from occurring.
- The most common conflict avoidance techniques are:
  - Workload partitioning
  - Data segmentation

Workload partitioning generally applies to a set of tables that belong to an easily identifiable group, which could be geography-based (US operations versus UK operations) or organization-based (HR application domain versus Finance application domain). In this context, partitioning does not imply physical table partitioning. Instead, it implies logical partitioning, where each "partition" pertains to the organizational group or the geographical entity (for example, the UK partition or the HR partition).

Data segmentation generally applies to columns within tables. One application is the "logical owner" of certain columns that belong to a table; a different application is the "logical owner" of a different set of columns. The two applications are allowed to update only their respective set of columns, which prevents them from crossing their segmentation boundaries. This arrangement minimizes, or even completely prevents, data conflicts.

Application Segmentation Through Workload Partitioning



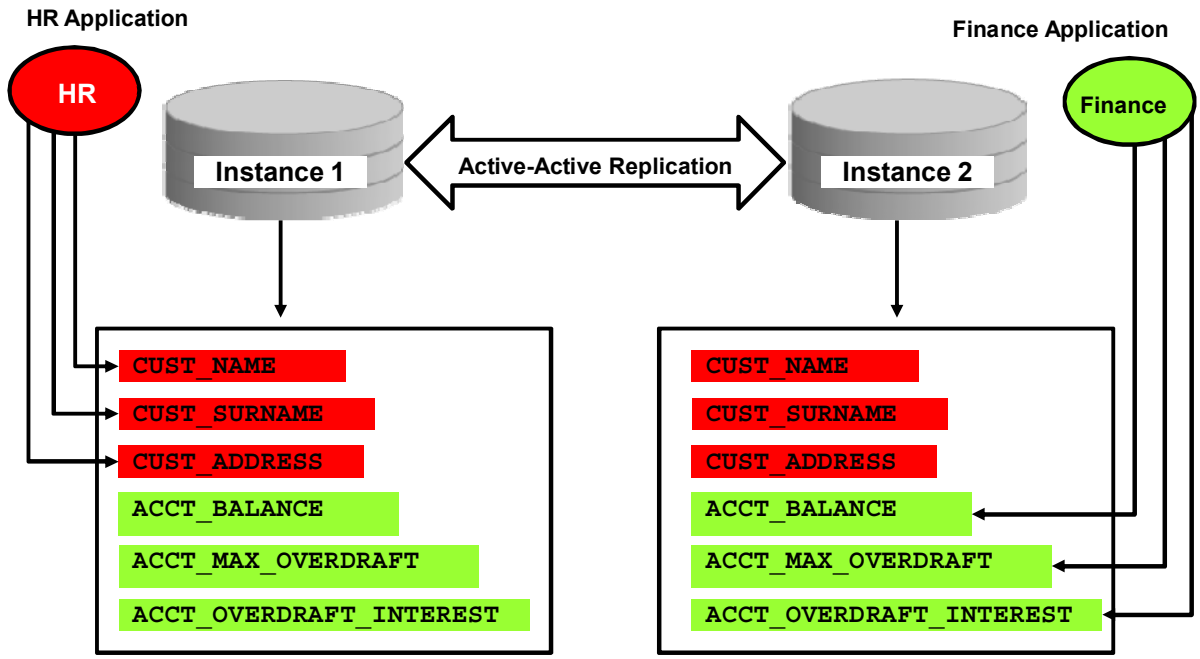
A multi-master configuration where conflicts are prevented by making sure that each server writes to only its assigned partition, while being able to access, in read-only mode, all available information

ORACLE

In this example, a three-way replication configuration is set up in such a way that all conflict types (insert, update, and delete) are avoided by using a strict segmentation based on geography.

The US-based server, which hosts the Oracle US instance, runs an application configured to write information on only the US partition of all shared tables. Accordingly, the UK server runs an application configured to write information on only the UK partition of all the shared tables. The same applies to the Australia-based server and instance. The three Oracle instances are constantly synchronized via Oracle GoldenGate, yet conflicts never occur thanks to the strict segmentation implemented at the database and application level.

Conflict Avoidance Through Data Segmentation

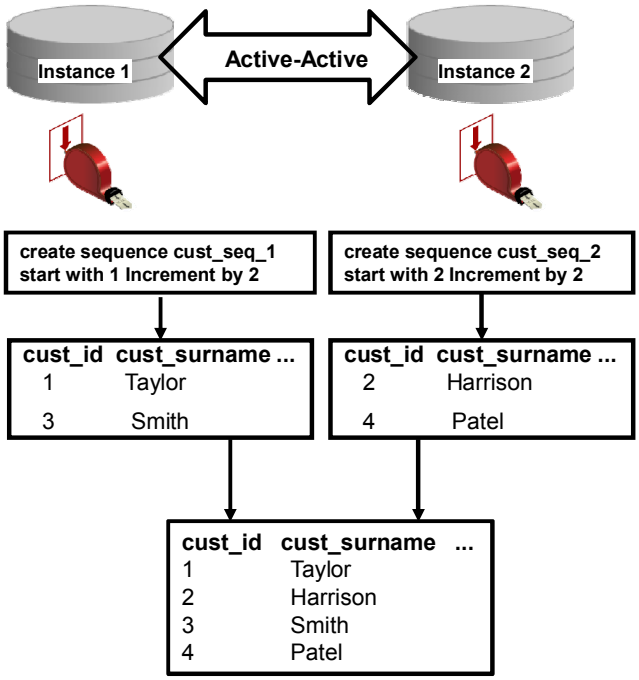


ORACLE

In this example, in an active-active replication configuration, updates to the financial columns of the `CUSTOMER_ACCT` table are allowed only from the financial application, and updates to the HR columns are allowed only from the HR application.

Though the database is kept synchronized through Oracle GoldenGate, data conflicts are prevented because of a strict separation of data ownership across the different applications.

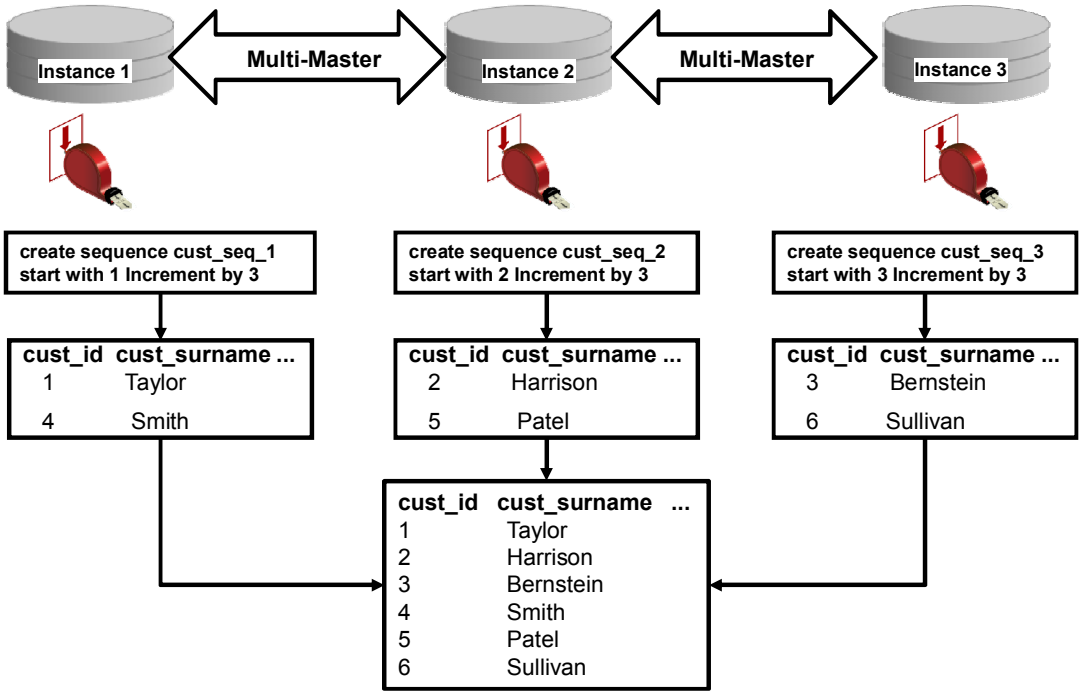
Insert Conflict Avoidance Through Sequence Generation



Many insert conflicts can be avoided just by handling primary key issues.

In this example, primary key generation is alternated across the two instances participating in an active-active configuration.

Multi-Master Primary Key Generation



ORACLE

Alternating the generation of the primary keys avoids most `INSERT` data conflicts.

For a three-server environment, server one starts at 1 and increments by 3 (1, 4, 7, 10, 13, and so on), server two starts at 2 and increments by 3 (2, 5, 8, 11, 14, and so on), and server three starts at 3 and increments by 3 (3, 6, 9, 12, 15, and so on).

Handling Delete Data Conflicts

- It is sometimes appropriate to log data conflicts without even trying to resolve them. If the business meaning of a row deletion is termination, it does not really matter what instance was responsible for the physical deletion of the row from the database.
- In this case, the delete data conflict can safely be ignored. It is a best practice, however, to detect the occurrence of the conflict and either keep a log of it or store the after image data in an exception table.

Quiz

Primary key generation using alternate sequence numbers is an effective method to prevent all types of data conflicts.

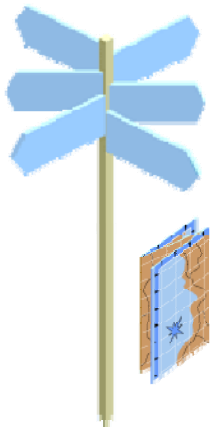
a. True
b. False

Answer: b

Primary key generation mitigates only `INSERT` data conflicts.

Roadmap

- Data conflict avoidance techniques
- Advanced conflict resolution methods



Custom Conflict Resolution via SQLEXEC

- The Oracle GoldenGate Automatic Conflict Detection and Resolution (CDR) mechanism offers a convenient method to handle simple data conflicts.
- There are instances, however, when CDR is not adequate to handle the complexity behind data conflict generation, and custom logic must be injected into the conflict resolution routines.
  - Oracle GoldenGate allows custom code to be run and evaluated to resolve data conflicts by using the **SQLEXEC** facility, which is available as a parameter of both **TABLE** (for Extract) and **MAP** (for Replicat) statements.

SQLEXEC was covered in lesson 6 titled “Data Mapping, Data Selection/Filtering, and Data Transformation.”

Rather than revisiting the syntax of this powerful command, this lesson shows SQLEXEC in action. SQLEXEC is applied to a classic data conflict scenario in this lesson and quantitative resolution methods are used to resolve inventory data conflicts.

Classic Data Conflict Scenario That Affects Inventory Columns

- Customers can buy T-shirts by placing orders online or via the phone. The call center is situated in Nashville, Tennessee, and phone orders are stored in a local database, which is synchronized active-active with the second database in New York, which serves the online clientele.
- There are currently 20 white T-shirts of size XXL in stock.
  - In rapid succession, two customers buy three white XXL T-shirts each, one customer placing an online order and the other customer ordering over the phone.

Simultaneous Transactions

```
New York Database (serving online orders)
update inventory set quantity = quantity - 3 and
item ='T-SHIRT' and color = 'WHITE' and size = 'XXL'
Nashville Database (serving phone orders)
update inventory set quantity = quantity - 3 and
item ='T-SHIRT' and color = 'WHITE'and size = 'XXL'
```

- Because both transactions occur in close proximity, the original inventory of 20 has not been updated to reflect changes.
- As a result, both transactions believe they are deducting the number of T-shirts ordered from an inventory of 20.

Occurrence of Data Conflict

- Consider the Nashville order. It is going to be applied to the New York database, but it fails. Why?
  - It fails because the before image of the quantity is expected to be 20 but, instead, it is 17.
  - The order from New York is also going to fail because it expects a quantity of 20 but, instead, it finds 17.
- The resolution needs to look at the actual change in inventory at the source system and apply that to the target system, rather than using the actual numbers.

Origin of the Discrepancy

The Nashville database is updated by using the following query:

```
update inventory set quantity = 17 where item ='T-SHIRT' and color
='WHITE' and size = 'XXL' and quantity = 20;
```

The New York database is updated by using the following query:

```
update inventory set quantity = 17 where item ='T-SHIRT' and color
='WHITE' and size = 'XXL' and quantity = 20;
```

Quantitative Resolution

- The formula for a quantitative resolution of the inventory update conflict scenario is:

```
update inventory set quantity = quantity - (before image
- after image) where item=<x> and size=<y> and color= <z>
```

- Applied to the Nashville database from New York:

```
update inventory set quantity = 17 - (20 - 17) where
item = 'T-SHIRT' and color = 'WHITE' and size = 'XXL';
```

- Applied to the New York database from Nashville:

```
update inventory set quantity = 17 - (20 - 17) where item
= 'T-SHIRT' and color = 'WHITE' and size = 'XXL';
```

ORACLE

Data convergence is achieved. Both databases reflect the correct change in the inventory: 14 white T-shirts are in stock after the two orders are fulfilled.

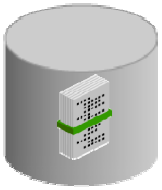
Implementing a Quantitative Resolution by Using SQLEXEC

The key to a quantitative resolution of update conflicts is to have access to the before image from the trail file.

- This is accomplished by specifying the **GETUPDATEBEFORES** and **GETBEFORECOLS** parameters for the Extract process:

```
GETUPDATEBEFORES
TABLE NASHVILLE.INVENTORY, GETBEFORECOLS (ON UPDATE ALL);
```

The processing logic occurs in the Replicat process, via a **SQLEXEC** statement included in the MAP clause for the **INVENTORY** table.



In the NASHVILLE database, the parameter file for the Extract that is processing the INVENTORY information is defined as:

```
Extract nexinv
ExtTrail ./dirdat/in
UserID gguser, Password Welcome1
GetUpdateBefores
TranLogOptions ExcludeUser gguser
Table NASHVILLE.INVENTORY GetBeforeCols(On Update All);
```

When an UPDATE statement is issued on the INVENTORY table, the before image information is written in the trail file for all columns in the table.

SQLEXEC in Action

```
GETUPDATES
Map NASHVILLE.INVENTORY, Target NYC.INVENTORY
SQLEXEC (id qty, query &
 "SELECT quantity FROM NYC.INVENTORY WHERE &
item=:v_item AND size=:v_size and color=:v_color ", &
    params (v_item = item, v_size = size, v_color = &
color)), colmap (item=item,  color = color, size = size,&
quantity = @if(qty.quantity <> BEFORE.quantity, &
@compute(qty.quantity-(BEFORE.quantity - quantity)), &
quantity));
```

- The query selects the quantity column when an update statement is processed.
- The **@if** and **@compute** functions are used to check if the quantity value is changed by the update, and to use the before image value to implement the quantitative resolution formula.

The Replicat parameter file contains the following statements:

```
REPLICAT RNYC
ASSUMETARGETDEFS
DISCARDFILE ./DIRRPT/RNYC.DSC, PURGE
USERID GGUSER, PASSWORD WELCOME1
ALLOWDUPTARGETMAP
MAP NASHVILLE.*, TARGET NYC.*;
GETUPDATES
IGNOREINSERTS
IGNOREDELETES
Map NASHVILLE.INVENTORY, Target NYC.INVENTORY
```

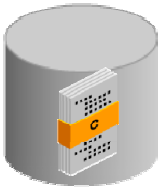
```
SQLEXEC (id qty, query
 "SELECT quantity FROM NASHVILLE.INVENTORY WHERE item=:v_item
 AND &
 size=:v_size and  color=:v_color ",  params (v_item = item,
 v_size =   size, v_color = &
 color)), colmap (item=item,     color = color, size = size,
 quantity =      @if(qty.quantity <> &
 BEFORE.quantity,    @compute(qty.quantity-(BEFORE.quantity -
 quantity)), quantity));
```

Using User Exits for More Complex Logic

SQLEXEC and the Oracle GoldenGate–provided functions such as **@if**, **@compute**, and **@case** are the building blocks that enable more sophisticated conflict resolution routines than CDR.

For even more sophisticated needs, for instance, when lookups and validations involve more than one table or when the computation involves complex rules, user exits can be used.

- The **CUSEREXIT** call accepts an **INCLUDEUPDATEBEFORES** parameter, which forces the before images of column values to be passed to the user exit.



ORACLE

When using `INCLUDEUPDATEBEFORES` for a user exit that is called from a Data Pump or from Replicat, always use the `GETUPDATEBEFORES` parameter for the primary Extract process. When you do this, the before image is captured and written to the trail, and it causes a `process_record` event in the user exit.

By default, Oracle GoldenGate works with only after images. When using the `INCLUDEUPDATEBEFORES` parameter for a `CUSEREXIT` call, you must explicitly request the before image by setting the `requesting_before_after_ind` flag to `BEFORE_IMAGE_VAL` within a callback function that supports this flag.

Summary

In this lesson, you should have learned how to:

- Minimize or prevent data conflicts by using conflict avoidance techniques
- Implement advanced data conflict resolution methods

Practice 10: Overview

This practice covers the following topics:

- Configuring a replication topology that avoids data conflict by using workload partitioning and primary key generation
- Configuring a replication topology that resolves data conflict by using quantitative resolution methods

11

Multi-Master Replication Topology: Three-Node Configuration

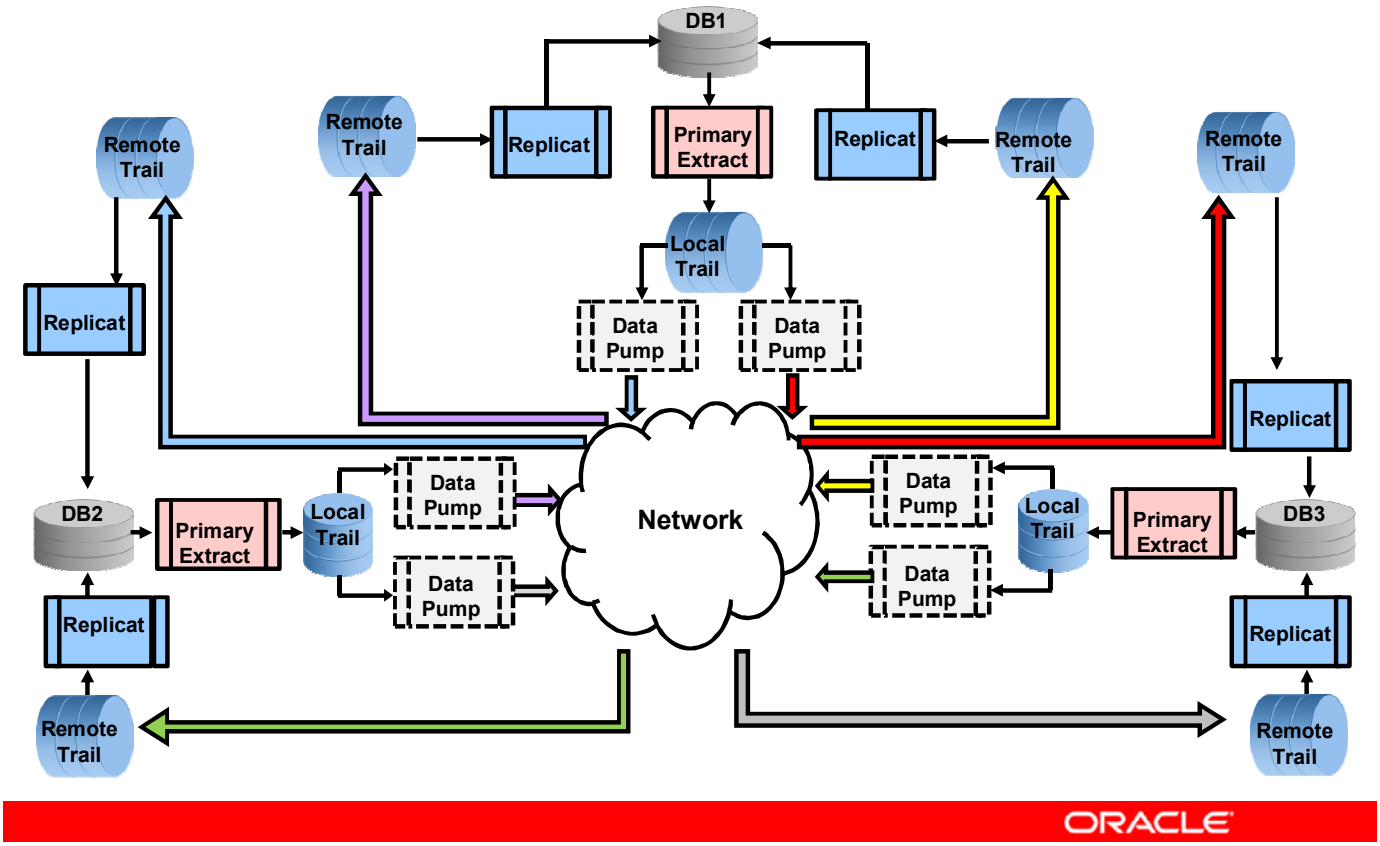
Objectives

After completing this lesson, you should be able to:

- Analyze the increased complexity that is inherent in a multi-master replication configuration
- Justify the need to implement data conflict avoidance techniques to keep data conflict resolution to a minimum level
- Implement data conflict resolution routines in a multi-master replication environment

ORACLE

Multi-Master Configuration Diagram



In a multi-master configuration, each database propagates its changes to all other databases in the replication chain.

- The database DB1 uses a local trail and two Data Pumps (red and blue arrows) to propagate the changes to DB2 (blue arrow) and DB3 (red arrow).
- The database DB2 uses a local trail and two Data Pumps (purple and gray arrows) to propagate the changes to DB1 (purple arrow) and DB3 (gray arrow).
- The database DB3 uses a local trail and two Data Pumps (yellow and green arrows) to propagate the changes to DB1 (yellow arrow) and DB2 (green arrow).

Increased Administrative Complexity

- A multi-master replication configuration implies a proliferation of replication objects.
  - The three-node configuration shown in the diagram in the previous slide has a minimum of:
    - Three primary Extract groups and three local trail files
    - Six Data Pump Extract groups
    - Six Replicat groups and six remote trail files
- Real-life replication implementations usually have many more objects.
  - The tables being replicated should be grouped in multiple Extract/Replicat pairs, according to criteria pertaining to performance and low- or high-activity ratios.

In a multi-master replication environment, each node must propagate all changes to all other nodes. Each Extract writes the database activity into a local trail file. Multiple Data Pumps, one per node in the replication chain, are responsible for transmitting the transactions to all nodes. At the remote nodes, each Replicat applies the changes to the target database.

In real-life scenarios, multiple Extract/Replicat pairs are needed to implement replication efficiently. Several criteria are used to determine how to group tables together in one Extract/Replicat pair:

- High transaction volumes versus low transaction volumes
- Logical table grouping according to business needs
- Physical table grouping with tables linked via referential integrity
- Conflict resolution complexity

Extract Group Configuration

- Consider using several Extract groups:
  - Use one Extract group to handle all tables with low activity rates.
  - Group tables with high activity rates and assign each group an Extract process.
  - Group tables implementing referential integrity (R.I.) with each other in the same Extract/Replicat pair.
- Each Extract/Replicat pair requires a significant amount of resources. You should strive to balance performance and memory consumption.
  - Avoid a proliferation of Extract groups; at the same time, avoid a monolithic Extract group to handle all tables.

ORACLE

Each Classic Extract or Replicat process requires more than 25 MB to 50 MB of system shared memory. An Integrated Extract requires even more memory (by default, 1 GB of memory per Extract), and this memory is allocated from the Oracle System Global Area (SGA).

You should strive to limit the number of Extract/Replicat pairs to a minimum, while satisfying the basic rules of grouping tables according to their activity rate, R.I. links, and similarity of business purpose.

Extract Group Configuration

In a multi-master configuration, consider grouping the tables that require special handling of data conflict and resolution in different Extract groups.

- Avoid extra-long lists of CDR routines crammed together in the same parameter file.
- Keep the size of the parameter file for each Extract/Replicat pair group to a manageable size.
- Use comments to document each data conflict and resolution routine.

ORACLE

Replicat parameter files can rapidly become unwieldy when many tables are listed one after the other, together with their data conflict and resolution routines.

Consider using the same name for the column that carries update time stamp information for all tables involved in time stamp–based resolution routines. You can use the asterisk sign to identify multiple tables in the `MAP` statement, and if the time stamp column is named the same across all tables identified by the asterisk, one CDR routine can be reused for all tables.

Example: Suppose there are several tables named with a `FIN` prefix to indicate that they carry financial data. Each table has a column named "`modified_ts`" that is populated every time the row is updated.

The following statement would define a time stamp–based conflict resolution routine for all financial tables at once:

```
MAP source.FIN*, TARGET target.FIN*,
 COMPARECOLS (ON UPDATE ALL),
 RESOLVECONFLICT (UPDATEROWEXISTS, (DEFAULT, USEMAX (modified_ts)),
 RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX (modified_ts));
```

Configuring a Multi-Master Extract

- A multi-master Extract parameter file must include several parameters:
  - **EXTRACT <name>**
  - **EXTTRAIL <local trail path>**
  - **TRANLOGOPTIONS EXCLUDEUSER | EXCLUDEUSERID**
  - **GETUPDATEBEFORES**
  - **GETBEFORECOLS (ON <operation> {ALL | KEY | KEYINCLUDING (<column list>) | ALLEXCLUDING (<column list>)})**
  - **TABLE <owner>.*** [, **FETCHCOLS <columns>** | **FETCHCOLSEXCEPT <columns>**];
- It cannot include **DDLOPTIONS**, because DDL replication is not supported in active-active configurations with more than two nodes.

In an active-active multi-master configuration, each Extract parameter list must include:

- Data looping prevention
- Before image inclusion, which is essential for data conflict and resolution handling

Additional parameters include:

- `ENCRYPTTRAIL <encryption options>` to encrypt the data stream during network transport
- `TABLEEXCLUDE <owner.table>` to exclude specific tables from replication

Configuring a Multi-Master Replicat

A multi-master Replicat parameter file must include several parameters:

- **REPLICAT <name>**
- **ASSUMETARGETDEFS**
- **GETUPDATEBEFORES**
- **MAP <owner>.*, TARGET <owner>.*,COMPARECOLS (ON <operation> {ALL | KEY |KEYINCLUDING (<column list>) | ALLEXCLUDING (<column list>)}), RESOLVECONFLICT (<conflict type> (<resolution name>, <resolution type> COLS(<column>[,...]));**

- **MAP <owner>.*, TARGET <owner>.<exceptions>, EXCEPTIONSONLY;**

ORACLE

In an active-active multi-master configuration, each Replicat parameter list must include:

- Before image inclusion
- Data conflict and resolution routines
- Mapping of exceptions to exceptions table: Although it is not mandatory, it is advisable to log all exceptions to ensure proper troubleshooting.

Although `ASSUMETARGETDEFS` is not strictly mandatory, in a complex environment such as a multi-master replication configuration, it is advisable to replicate identical schemas and objects.

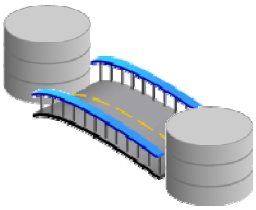
Additional parameters include:

- `DECRYPTTRAIL <encryption options>` to decrypt the data stream during network transport
- `MAPEXCLUDE <owner.table>` to exclude specific tables from delivery

Multi-Master Data Pumps

For each local trail file, the number of Data Pumps should be equivalent to the number of replicated nodes minus one.

- In a three-node configuration, one local trail has two Data Pump Extract processes, which write to the corresponding remote trail file on the other two nodes.
- Data Pumps should be declared as **PASSTHRU**.
  - In active-active configurations, schemas across replicated instances contain identical objects. **PASSTHRU** generally improves the performance of the pump.



ORACLE

To use the PASSTHRU mode, the names of the source and target objects must be identical. No column mapping, filtering, SQLEXEC functions, transformation, or other functions that require data manipulation can be specified in the parameter file. Although not required, complex replication configurations should avoid replicating objects that are not identical, because this requires the use of DEFGEN and the SOURCEDEFS parameter, which preclude the use of PASSTHRU.

Data Conflict Avoidance

Given the inherent complexity of a multi-master replication configuration, it is essential to minimize data conflicts.

- Whenever possible, restrict the ability to insert and update data to one node only.
  - Use workload partitioning and data segmentation.
- Manage primary keys through alternate sequencing to avoid **INSERT** conflicts.
- Prioritize **UPDATE** potential conflicts from simpler resolution to more complex resolution, and implement simpler resolution by using CDR.
  - Use custom resolution techniques based on **SQLEXEC** to handle more difficult resolutions.
  - Resort to user exits for the most complex conflicts.

ORACLE

It is possible to minimize data conflict for `INSERT` operations by using the alternate sequencing technique for primary key generation.

Sometimes, `DELETE` operations that generate data conflicts can be safely disregarded, if a row delete means termination of a business entity, such as an account, and it does not matter which node initiated the termination.

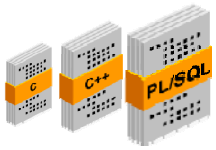
`UPDATE` operations are the most likely cause for data conflicts. It is essential in active-active replication configurations to keep `UPDATEs` to a minimum. There are two main ways to accomplish this goal:

- Use workload partitioning to confine updates to tables or partitions to a single node in the multi-master environment.
- At an application level, limit the columns to be updated.

Avoid using client tools that automatically generate SQL statements behind the scenes; they usually reverse-engineer the table structure from the data dictionary, and create CRUD forms using all columns belonging to a table. In the process, they generate `UPDATE` statements for each column, even when the form modifies only one or two fields.

Data Conflict Management

- Minimize the number of columns that can be updated.
- Consider adding a time stamp column to each table, which stores the time of the last update to the row.
  - Use CDR with **`USEMAX` or `USEMIN <timestamp column>`** to implement time stamp–based resolution.
- Consider using trusted-source resolution methods when it makes business sense (simplest possible conflict resolution).
- For inventory-like columns, implement the quantitative resolution formula using **`SQLEXEC`**.
- Code a user exit for the most complex resolution logic.



ORACLE

From a conceptual standpoint, the trusted-source resolution method is the simplest. In case of conflict, data stored in the trusted source always wins.

Time stamp–based resolution provides added flexibility. The `USEMAX` and `USEMIN` clauses allow for the resolution of conflicts based on the time when the row was updated. You can decide that for a specific column, the newer value is to be accepted for data convergence, whereas a different column could follow the opposite rule (the older value is the one to be chosen).

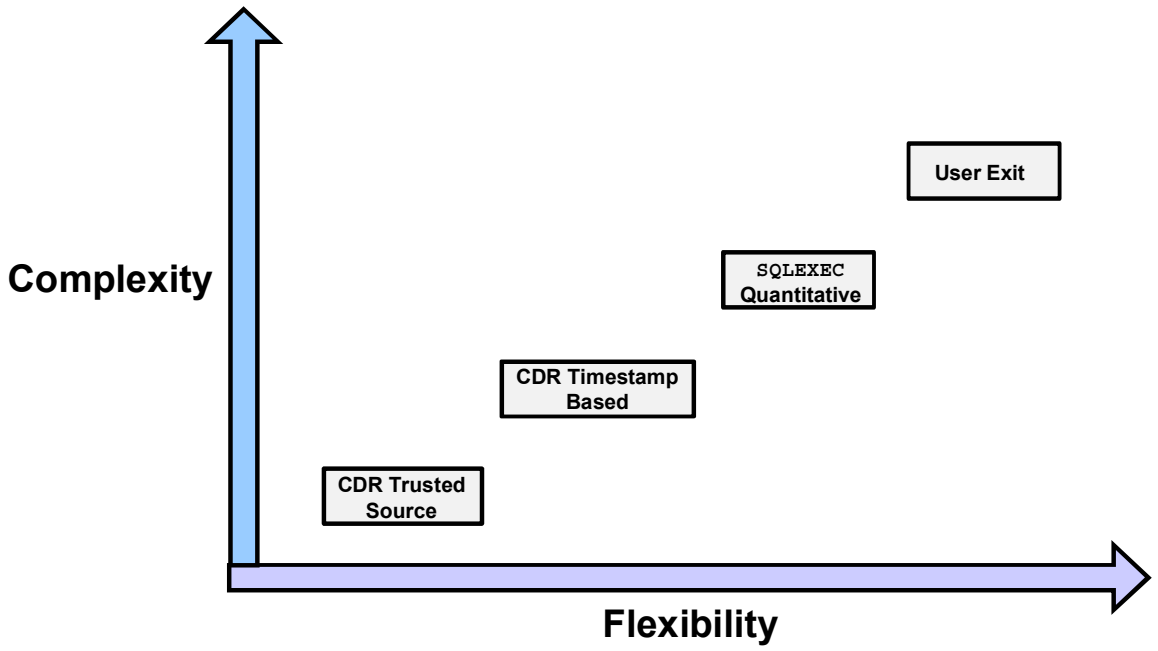
Watermark type of columns (that is, columns that store quantities, balances, and—in general—values that can increase and decrease at each update) should be handled using the quantitative formula:

update <quantity_column> set quantity = quantity – (before image – after image)

`SQLEXEC` is also extremely useful in other resolution scenarios, for instance, when a decision must be made based on complex lookups that involve fetching values from tables that are not included in the transaction processed by Replicat.

Very complex resolution scenarios, where the decision on what value to retain for data convergence is based on several factors that require elaborate computation, might require user exits. The inherent complexity of a user exit and the administrative burden associated with its management are good reasons to limit the number of user exits to a minimum.

Conflict Resolution: Complexity Versus Flexibility



Summary

In this lesson, you should have learned how to:

- Analyze the increased complexity that is inherent in a multi-master replication configuration
- Justify the need to implement data conflict avoidance techniques to keep data conflict resolution to a minimum level
- Implement data conflict resolution routines in a multi-master replication environment

Practice 11: Overview

This practice covers the following topics:

- Creating a three-way active-active replication environment
- Implementing data segmentation to avoid conflicts
- Configuring data conflict detection and resolution:
  - CDR, trusted-source method
  - CDR, time stamp–based method
- Generating DML activity to verify data convergence and check data conflict resolution statistics

12

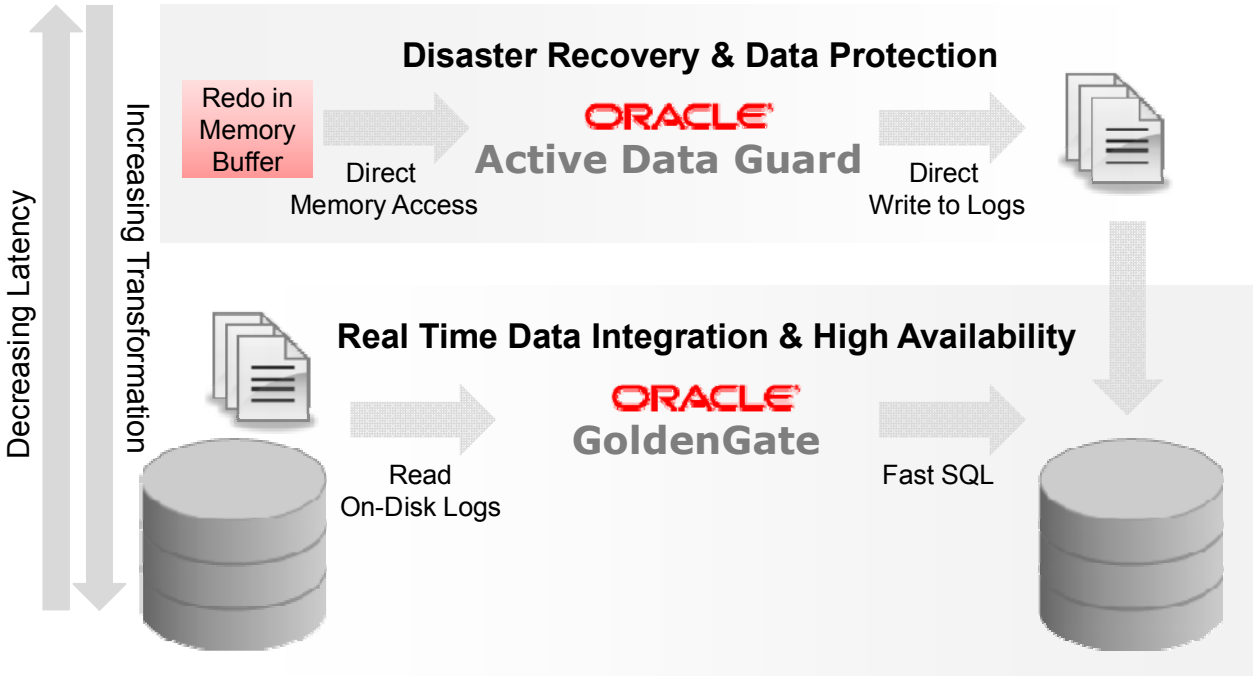
Active Data Guard and Oracle GoldenGate: How to Achieve Maximum Availability

Objectives

After completing this lesson, you should be able to:

- Identify scenarios and use cases that require Oracle GoldenGate to operate with Active Data Guard
- Configure data extraction and delivery to automatically restart after a Data Guard role transition
- Ensure consistency and synchronization between Extract and Replicat after a role transition

Active Data Guard and Oracle GoldenGate



Basic Oracle Data Guard Concepts

- **Switchover:** Planned role transition from a primary database to one of its standby databases

```
DGMGRL > SWITCHOVER TO DALLAS
```

  - Requires connectivity to both primary and standby database
  - Ensures that all redo files have been shipped
  - Ensures orderly transition (both databases agree on everything)
- **Failover:** It occurs when the primary database (all instances of a RAC primary database) fails and one of the standby databases is transitioned to take over the primary role.

```
DGMGRL > FAILOVER TO DALLAS
```

  - No connectivity with primary
  - New primary becomes the source of truth

ORACLE

Data Guard command-line interface (DGMGRL) is used in the examples to manually switch over (or fail over) the primary database onto the secondary database.

When the Fast Start Fail Over (FSFO) facility is enabled, fail over to standby is automatically initiated if certain triggering events happen:

- The datafile has gone offline because of an I/O error.
- The control file is deemed to be corrupt.
- The Log Writer (LGWR) process gets an I/O error and cannot write to any log file.
- The ARCHIVER cannot write because of I/O error.
- Dictionary corruption is detected.

Data Guard FSFO

- Allows the Data Guard broker to automatically fail over to a previously chosen, synchronized standby database in the event of loss of the primary database
- Quickly and reliably fails over the target standby database to the primary database role, without requiring any manual steps to invoke the failover

The Oracle Data Guard broker is a distributed management framework that automates and centralizes the creation, maintenance, and monitoring of Data Guard configurations.

FSFO Modes of Operation

- Zero Data Loss Mode
  - Redo transport is set to **`SYNC`** with **`MaxAvailability`** protection mode.
- User-specified Data Loss Mode
  - Redo transport is set to **`ASYNC`** with **`MaxPerformance`** protection mode.
  - The user can specify the maximum amount of data loss (**`FastStartFailoverLagLimit`**).
- Reinstatement of the Failed Primary Database
  - Following the failover, Data Guard broker will automatically try to reinstate the failed primary as a new standby database.

FSFO Issue: 1

- Active Data Guard used for offloading read-intensive applications, where a multi-node RAC primary database fails over to a single instance database
  - Post Role Transition: Thread counts may be different; need for capability to handle such mismatches transparently
- Resetlogs change on failover
  - Failover always results in creation of a new database incarnation; multiple fast start failovers can happen in a short period of time; need for handling resetlogs operation transparently
- Zero Data Loss Guarantee
  - Redo state is fuzzy until ACK is received; need for a way to avoid redo fuzziness during Extract

ORACLE

In a Zero Data Loss Guarantee configuration, commits are not acknowledged to the user until an ACK is received from standby.

In this case, the redo state is fuzzy until the ACK is received, as the scenarios below can occur:

- Commit in both Online Redo Logs (ORLs) and Standby Redo logs (SRLs)
- Commit in ORL but not in SRL
- Commit in SRL but not in ORL
- Commit in neither ORL nor SRL

FSFO Issue: 2

- In an **ASYNC FSFO** configuration, the user can define an acceptable window of data loss. Transitions of this nature expect some amount of data loss, so replication must account for this.
  - Oracle GoldenGate must remain consistent with the failover target database.
- After a RAC node fails, Oracle GoldenGate must resume.
  - GoldenGate processes should not require manual intervention to restart on node failures if other nodes are available.
- Similarly, after role transitions, the GoldenGate processes must be cleaned up at the old primary and restored on the new primary.
  - GoldenGate must seamlessly follow the primary database.

FSFO Issue: 3

- Primary and standby databases do not share storage.
  - Oracle GoldenGate files must be available post role transition:
    - Checkpoint file
    - Bounded Recovery file
    - Trail file
    - Parameter file
- GoldenGate must be kept consistent post role transition.
  - True for both no data loss and data loss transitions
- After a Data Guard role transition of a Target database:
  - Data Pump may need to send trail data to the new primary; Replicat processes wait until new trail file data is received; need for a mechanism to ensure proper routing of trail after transition events

ORACLE

Role Transition Complexity: Oracle GoldenGate to the Rescue

- Clearly, there is some technical and procedural complexity around role transition events.
  - Understand what Zero Data Loss means.
- The database is the source of truth.
  - Only the RDBMS can give you zero data loss.
  - The software that is managing the role transition scenarios must inevitably be closely integrated with the RDBMS software.
- Oracle GoldenGate can be used to support maximum availability; it complements the Active Data Guard features and takes care of "edge" scenarios.

Oracle Bundled Agent (XAG)

- Is Oracle Clusterware additional software, specifically designed to manage GoldenGate resources
  - XAG allows you to register a GoldenGate instance with Cluster Ready Services (CRS) to improve High Availability.
- Uses the **`AGCTL`** facility for registering/starting and stopping resources
- Solves the key process-related issues of detecting and migrating an Oracle GoldenGate instance, i.e. full failover support for GoldenGate

XAG: Increased Availability

- XAG provides increased availability in the face of failures:
  - Loss of instance (RAC node failure)
  - Loss of primary database (Data Guard Failover integration)
- XAG **agctl** in action:

```
$ agctl add goldengate GG_Source --gg_home /u03/ogg \
    --instance_type source \
    --nodes dbm01db05,dbm01db06 \
    --vip_name gg_vip_source \
    --filesystems dbfs_mount --databases ggs \
    --oracle_home /u01/app/oracle/product/12.1/db_1 \
    --monitor_extracts ext_1a,dpump_1a
$ agctl start goldengate GG_Source
```

- After the Oracle GoldenGate processes have been added to a bundled resource, they should only be started and stopped using **AGCTL**.

ORACLE

In the example in the slide, the agctl utility is used to define an Oracle GoldenGate–related resource (GG_Source)—all required parameters are provided:

- GoldenGate home (--gg_home)
- Instance type (--instance_type, source in the example)
- Nodes (--nodes, dbm01db05 and dbm01db06 in the example)
- Virtual Ip Name (--vip_name, gg_vip_source in the example)
- File systems (--filesystems, in the example, dbfs_mount, which is the CRS-defined resource for the DBFS mount point)
- Oracle Home (--oracle home)
- Oracle GoldenGate data extraction groups (--monitor_extracts, in the example, ext_1a and dpump_1a)

XAG: Key to Maximum Availability

- XAG integrates with Oracle Clusterware and promotes Oracle GoldenGate to be a first-class citizen in the cluster infrastructure.
- XAG makes it possible to automate all operations that are required to ensure transparent recovery of data extraction and delivery after a role transition (Active Data Guard) or a loss of instance (Oracle RAC configuration).

Summary

In this lesson, you should have learned how to:

- Identify scenarios and use cases that require Oracle GoldenGate to operate with Active Data Guard
- Use the Oracle Bundled Agent (XAG) to automate Oracle GoldenGate–related tasks
- Ensure that replication resumes with no fuzziness or data loss after a role transition

Practice 12: Overview

This practice covers the following topics:

- Installing the XAG software on both RAC nodes
- Configuring a virtual IP for Oracle GoldenGate
- Configuring a bundled OGG resource, including Extract and Data Pump
- Simulating a RAC node failure to exercise XAG failover