

Bi-Directional Topology for Business Continuity

CHAPTER

5

Overview

The objective of this chapter is to use Oracle GoldenGate 12c two-node configuration for building high availability infrastructure to support business applications maximum availability by minimizing disruption due to system failure. Oracle customers have been implementing Oracle real application clusters (RAC) for high availability and Oracle Data Guard (DG) for disaster recovery. The two solutions co-exist for best continuity. RAC and DG require an advanced level of skills to implement and manage together. RAC and DG are the major components of Oracle Maximum Availability Architecture (MAA) for sites running mission critical systems and requiring a maximum level of scalability, data availability, and data protection. For applications that may compromise RAC embedded scalability and DG fast start failover, Oracle GoldenGate achieves the requirements for business continuity with a sustainable budget, operational cost, and reasonable infrastructure complexity.

The flexibility of Oracle GoldenGate advanced configuration enables the software to provide business continuity similar to RAC and DG combined using active-passive and active-active bi-directional topology architecture. However, this specific implementation is not out-of-the-box and requires a considerable level of Oracle GoldenGate and database experience in respect to the database design and applications coding.

This chapter implements Oracle GoldenGate for business continuity using similarly designed architecture for deploying:

- Active-Passive Model for Disaster Recovery (DR)
- Active-Active Model for High Availability (HA)

Unlike Oracle Real Application Clusters (RAC) in terms of scalability, Oracle GoldenGate does not promote the built-in scalability provided by RAC, which is the core function of RAC to support linearly increasing transactions volume. When scalability is an important element of the application, sufficient evaluation is necessary before deploying Oracle GoldenGate for high availability. Also, with Oracle Real Application Clusters (RAC), increasing the number of nodes delivers higher scalability.

However, Oracle GoldenGate encourages the use of two-node architecture only for high-availability. This is recommended to avoid complex data conflict detection and resolutions in an active-active bi-directional topology.

For an enterprise, Oracle GoldenGate does not replace RAC or DG, rather it is implemented to complement and integrate with the Oracle clusterware and Oracle physical standby database. Refer to Chapter 12 to learn more about integrating Oracle GoldenGate with RAC and DG.

For sites running the Oracle Database Standard Edition, Oracle GoldenGate is an attractive option for implementing high availability (HA) and disaster recovery (DR). The option is cost-effective and highly suitable for in-house developed applications, but needs to be developed from the group-up so that the applications are replication-aware. In deploying Oracle GoldenGate on bi-directional active-active topology, the challenge is to handle data conflicts which are discussed using the following:

- Data and Applications Segmentation
- Oracle GoldenGate Built-In Conflict Detection and Resolution (CDR)
- Custom Conflict Detection and Resolution using PL/SQL Sub-Programs Development
- Oracle Virtual Private Database (VPD)

GoldenGate Business Continuity Architecture

Oracle GoldenGate bi-directional architecture achieves better performance by splitting the load across two instances. This sustains fault tolerance when one of the two instances fail by moving all processing to the surviving system. However, the top advantage of using Oracle GoldenGate bi-directional architecture is the geographically separated data centers that enable cost-effective business continuity.

Figure 5-1 depicts bi-directional topology components needed for business continuity. The data flow is from the primary system (the source) to the live standby system (the target) and vice versa. The primary system database S1E2 is on Oracle Database 11g and the live standby system database T1E2 is on Oracle Database 11g. The Oracle GoldenGate building blocks of extract groups, delivery groups, and related components for data movement are outlined below.

The Primary System (The Source) Components

- Extract groups, the primary extract ES1E203, the local trail ./dirdat/sc and the secondary extract PS1E203.

- The delivery group, the dynamic collector server, the remote trail `./dirdat/tc` and the delivery process RS1E203.

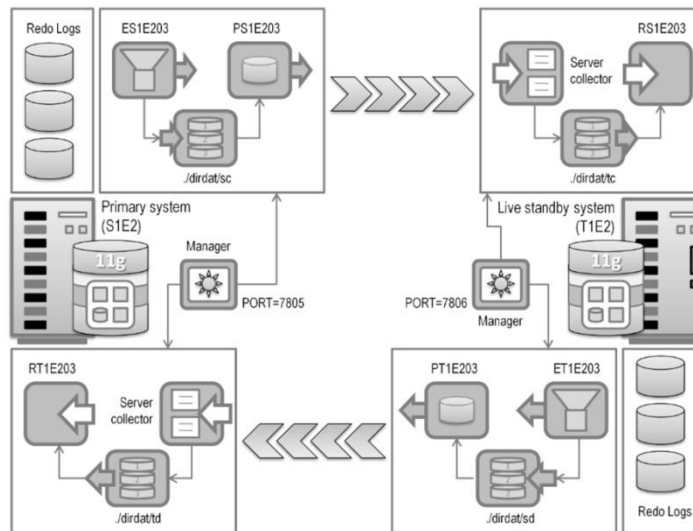


Figure 5-1: Bi-Directional Topology Building Blocks

The Live Standby System (The Target) Components

- Extract groups, the primary extract ET1E203, the local trail `./dirdat/sd` and the secondary extract PT1E203.
- The delivery group, the dynamic collector server, the remote trail `./dirdat/td` and the delivery process RT1E203.

Database Configuration

The primary system database and live standby database are operating on ARCHIVELOG mode. The transaction logs (online redo log files) structure of both databases are identical. Each has 3 groups used by the primary extract for performing change data capture (CDC). Online redo log files are archived and stored in the fast recovery area (FRA) for a recovery window of (7) days in case of being required by the primary extract or the recovery manager (RMAN) recover operation. The below RUN command backup the archived log files while maintaining a recovery window of (7) days of archived files in the FRA. Deleting files that are not required by the recovery window of (7) days.

```
RMAN> RUN {
2> BACKUP ARCHIVELOG ALL;
```

```

3> DELETE COPY OF ARCHIVELOG ALL COMPLETED BEFORE 'SYSDATE-7';
4> }

```

At minimum, set the size of the recovery window equal to or greater than the value of the database parameter `CONTROL_FILE_RECORD_KEEP_TIME`. The default is (7) days, unless a recovery catalog is configured. Also, ensuring FRA storage capacity allows retaining (7) days of archived log files by setting the database parameter `DB_RECOVERY_FILE_DEST_SIZE` as needed.

Extending Oracle GoldenGate for High Availability

When using Oracle GoldenGate for business continuity, the task of migrating users from the primary system to standby system and vice versa requires the database administrator (DBA) intervention and users reconnect to the surviving instance. However, for an active-active environment, the application needs the load balancing technique for connecting sessions to the database with minimal workload. Figure 5-2 shows the load balancer workflow that dynamically routes the application to the database with minimal workload.

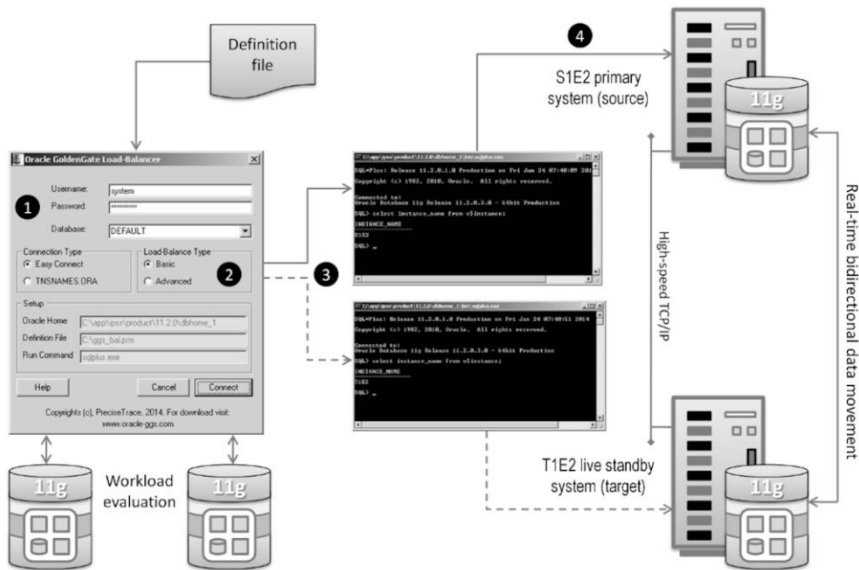


Figure 5-2: High Availability Load Balancer

The load balancer negotiates the primary and standby database to determine the preferred instance based on current workload. Then, it uses the selected connection type to establish the connection to the primary or standby systems – the database

with lesser workload. The load balancer uses a definition file that defines the primary and standby systems. Each line is composed of the host name or IP address, the listener port, the database service name, and the run command. The definition file `c:\ggs-bal.prm` is mandatory and the database credential must be identical for the primary system and live standby system database. The following lines define the primary and live standby systems configuration.

```
ggs-source      1521    sle2.precisetrace.com    sqlplus.exe
ggs-target      1521    tle2.precisetrace.com    sqlplus.exe
```

The above load balancer maintains the workload between the primary and the live-standby systems by performing the following:

1. Connecting to the primary and standby systems for evaluating the current database workload.
2. Comparing the two workloads to determine the preferred database instance – the one with lesser workload.
3. Using the selected connection type to compose the database connect string.
4. Routing and spawning the application to the preferred system.

When using Oracle GoldenGate for high-availability (active-active), the number of Oracle GoldenGate database instances should be limited to a maximum of (3) database instances to avoid complex conflicts resolution and configuration. To download the load balancer utility, visit <http://www.oracle-ggs.com>.

Disaster Recovery

When using Oracle GoldenGate to implement a disaster recovery (DR) site, it is developed as *active-passive*, bi-directional topology architecture. The source database is designated as the primary system and the target database is designated as the live-standby system. The live-standby system database remains on read-write mode. Applications users are connected to either the primary or the live-standby system, but not both. The live-standby system remains ready for the failover and switchover situations. Routing connection to one-and-only-one system is mostly handled from within the application servers as depicted on Figure 5-3 where sessions are established from either the primary or the live-standby system database.

The tremendous advanced features, configurable components flexibility, communication robustness and extendibility of Oracle GoldenGate have technically established a distinctive architecture for designing an enterprise heterogeneous platform suitable for protection and integration the enterprise data. Having a live-standby database continuously synchronized in real-time maximizes the usability of the database for other applications. Another distinctive advantage of Oracle GoldenGate is heterogeneity.

The capability of the live-standby system to run on a dissimilar platform from the primary system attracts decision makers and technical team members to build a sophisticated, one-of-a-kind disaster recovery site. Mostly designed to run on a mid-range Intel-based system powered by Oracle Linux. It delivers high-quality service and low-cost per transaction with best of class stability and reliability.

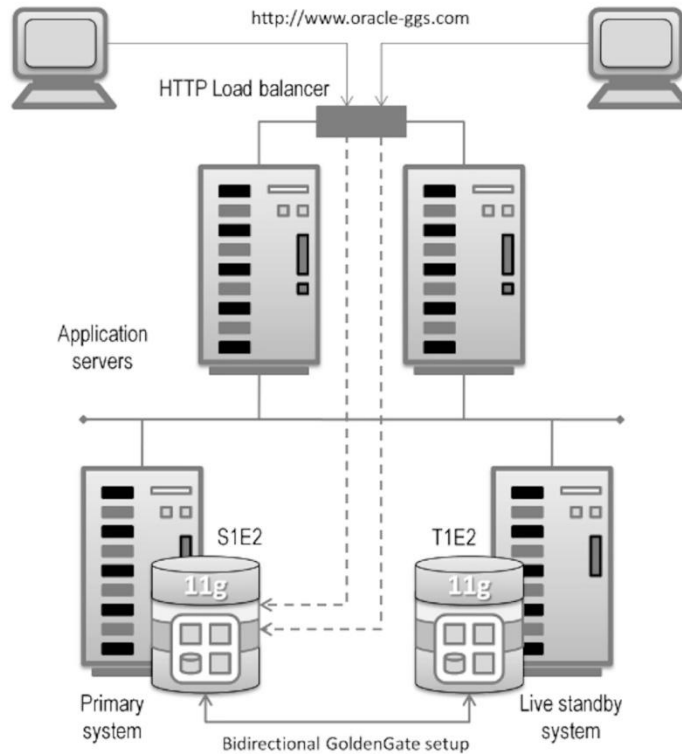


Figure 5-3: Disaster Recovery Connections

Oracle GoldenGate Value Proposition

A value proposition of Oracle GoldenGate is the certification with Oracle Database Standard Edition (SE). Small to mid-range server sites running Oracle Database SE can develop their disaster recovery solution using Oracle GoldenGate only.

The demonstration of this chapter was developed using Oracle Database 11g SE running on Oracle Linux. This value proposition should not be underestimated!

```
SQL> SELECT * FROM v$version;
```

```
BANNER
```

```
-----  
Oracle Database 11g Release 11.2.0.3.0 - 64bit Production
```

```
PL/SQL Release 11.2.0.3.0 - Production
CORE      11.2.0.3.0      Production
TNS for Linux: Version 11.2.0.3.0 - Production
NLSRTL Version 11.2.0.3.0 - Production
```

```
SQL>
```

Disaster Recovery Considerations

The concept of disaster recovery (DR) imposes the standby database to operate on read-only mode at all times. However, because Oracle GoldenGate requires that the database operates on read-write, it is only the synchronized tables that are restricted from DML operation outside of the Oracle GoldenGate apply session (Replicat). Also, even though the bi-directional setup is already in place, it is active from only one direction at any time. Other guidelines to consider when using Oracle GoldenGate for disaster recovery are listed below:

Live-Standby System

At all times, the data flow is active from only one direction. The live-standby database (target) remains on read-write mode but fundamentally inactive – used only by the Replicat sessions. Under normal situations, OLTP users are connected to the primary system database (source) only. Utilizing the live-standby system for other applications should avoid interaction with mapped tables used by Oracle GoldenGate delivery processes (Replicats).

Operating Mode and Failover

The live-standby system database should be operated on ARCHIVELOG mode. This prepares the live-standby system for a planned switchover and unplanned failover. Oracle GoldenGate switchover and failover is an interactive operation and handled manually. Using tools such as Oracle GoldenGate Veridata and the Oracle GoldenGate Monitor are highly recommended to proactively avoid data divergence and minimizing latency.

Query Offloading

OLAP users are normally directed to the live-standby system database (target). It is the ideal platform for running long query reports, but the database should not be modified by creating additional objects such as indexes and materialized views. Otherwise, these created objects must be dropped as part of a failover/switchover operation.

Unsupported Data Types

The unsupported data types by Oracle GoldenGate is minimum. The most common applications data types not supported by Oracle GoldenGate is BFILE, which handled using the PL/SQL supplied API DBMS_LOB and DBMS_FILE_TRANSFER. Refer to Chapter 11 to learn more about using these packages with Oracle GoldenGate. Evaluating the extent of unsupported data types is necessary to qualify using Oracle GoldenGate for disaster recovery solution. Refer to Chapter 1 to learn more about unsupported data types.

Disaster Recovery Implementation

A disaster recovery implementation using Oracle GoldenGate consists of the tasks listed in Table 5-1. Testing switchover and failover for moving users from the primary system to the live-standby system and then moving users back to the primary system is the last step and executed for planned & unplanned scenarios.

Task
1. Install software, configure manager process, and start Oracle GoldenGate instance.
2. Enable database supplemental logging and archive log mode.
3. Create Oracle GoldenGate database user and grant privileges.
4. Login to the database from GGSCI and Add table-level logging.
5. Prepare GLOBALS parameters and the checkpoint table.
6. Perform initial load from primary (source) to live standby (target) systems.
7. Configure, create, and deploy Oracle GoldenGate groups for bi-directional topology.
8. Enable data capture and delivery from primary to live standby system.
9. Perform moving users for planned & unplanned switchover and failover.

Table 5-1: Bi-Directional Configuration for High Availability and Disaster Recovery Solutions

The destination for each of the above tasks is shown in Figure 5-4. Since this is a bi-directional topology architecture, the tasks are performed on the primary system/source database and on the live-standby system/target database.

The live-standby system should have adequate resources to support the post-failover/ switchover operation timeframe. Having matched resources is recommended. Even though Oracle GoldenGate supports a heterogeneous database environment, it is absolutely not recommended when implementing high availability and disaster recovery solutions using Oracle GoldenGate. The primary and live-standby platform should have identical architecture. The planned switchover and the unplanned failover is a manual process. Automation is developed using Linux shell script and Oracle GoldenGate obey files. The use of Oracle GoldenGate Veridata for data consistency and the Oracle GoldenGate Monitor are highly recommended for such an environment.

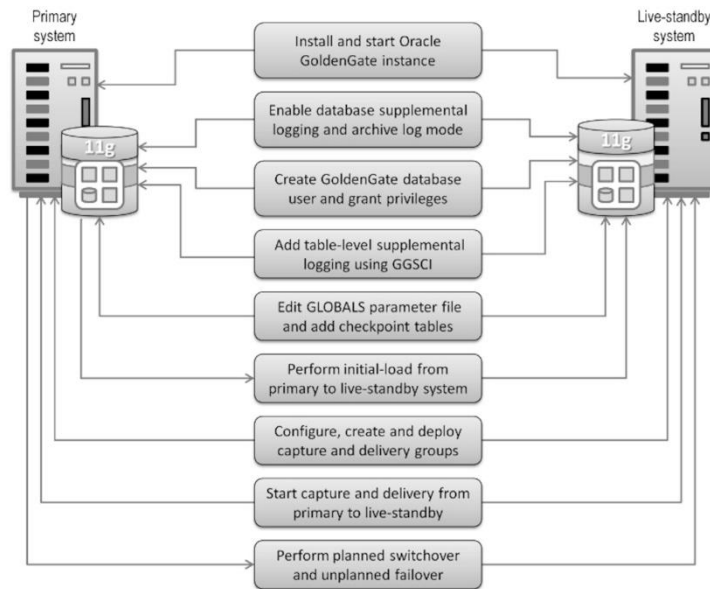


Figure 5-4: Bi-Directional Topology Architecture Tasks vs. Destinations

Software Installation

Successful installation of Oracle GoldenGate on the primary and live-standby systems must be successfully completed to proceed further. Consider the platform architecture. The chapter uses Oracle GoldenGate for Oracle Database 11g 64-bit. The same bundle of Oracle GoldenGate 12c provides support for Oracle database enterprise and standard editions. Refer to Chapter 2 to learn about installing Oracle GoldenGate.

Supplemental Logging

Enable supplemental logging on the primary and live-standby databases. Also, ensure the databases are operating on ARCHIVELOG mode.

```
SQL> CONN / AS SYSDBA
Connected.
SQL> SELECT log_mode FROM v$database;

LOG_MODE
-----
ARCHIVELOG

SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

```
Database altered.
```

```
SQL>
```

Refer to Chapter 2 to learn more about database supplemental logging settings. Even though the data flow is in one direction at any time, ensuring the database supplemental logging and ARCHIVELOG mode are enabled on the primary and Live-standby systems minimizes the meantime to migrate users and reverse the processing direction.

Database Users

Create Oracle GoldenGate database user on the primary and live-standby databases. The use of dedicated Oracle GoldenGate is required for active-active bi-directional topology. It is used for an endless loop detection purpose as explained in this chapter.

```
SQL> CREATE USER ggs_admin identified by oracle;
```

```
User created.
```

```
SQL> GRANT DBA TO ggs_admin;
```

```
Grant succeeded.
```

```
SQL> CONN ggs_admin/oracle
```

```
Connected.
```

```
SQL>
```

Refer to Chapter 2 to learn more about the precise database privileges granted to Oracle GoldenGate users on the primary (source) and live-standby (target) database. The above command grants DBA which is considered an over grant and not recommended or even possible for some sites.

Start Oracle GoldenGate Instances

The primary system and live-standby system are running on Oracle Database 11g Standard Edition 64-bit. The primary system Oracle GoldenGate instance manager process is listening on port 7805 and the live-standby Oracle GoldenGate manager process is listening on port 7806. Using the GoldenGate Software Command Interface (GGSCI), configure and start the manager process on the primary and live-standby systems.

```
GGSCI (apps-srv1) 6> START MGR
```

```
Manager started.
```

```
GGSCI (apps-srv1) 7> INFO MGR
```

```
Manager is running (IP port apps-srv1.7805, Process ID 13632).
```

```
GGSCI (apps-srv1) 8>
```

Repeat the above GGSCI commands for the live-standby system.

```
GGSCI (apps-srv2) 2> START MGR  
Manager started.
```

```
GGSCI (apps-srv2) 3> INFO MGR
```

```
Manager is running (IP port apps-srv2.7806, Process ID 30133).
```

```
GGSCI (apps-srv2) 4>
```

Refer to Chapter 2 to learn more about the manager process common parameters settings for managing trails and unattended restart.

Checkpoint Table

The database checkpoint table is the default for the Replicat process checkpoint. Prepare the GLOBALS parameter file, then use GGSCI to create the checkpoint table. The GLOBALS parameter file contains the following one line only. Where GGS-ADMIN is the Oracle GoldenGate database schema.

```
CHECKPOINTTABLE ggs-admin.chkpttab
```

```
GGSCI (apps-srv1) 1> DBLOGIN USERID ggs_admin, PASSWORD oracle  
Successfully logged into database.
```

```
GGSCI (apps-srv1) 2> ADD CHECKPOINTTABLE
```

```
No checkpoint table specified. Using GLOBALS specification  
(ggs_admin.chkpttab)...
```

```
Successfully created checkpoint table ggs_admin.chkpttab.
```

```
GGSCI (apps-srv1) 3>
```

Create the checkpoint table on primary system and live-standby systems. Remember to exit and restart GGSCI to activate parameters added to GLOBALS parameter file.

Performing Initial-Data Load

This is a major task and must be delivered 100% successfully. The live-standby system database must be established prior to beginning the configuration tasks. Even though this demonstration is from Oracle-to-Oracle database (homogenous), file to Replicat Oracle GoldenGate initial-data load method is used to establish the live-standby (target)

database. Since we consider the initial load from an inactive database, the initial load is performed using the following steps:

- Configure and start the initial load Extract (ELOAD03)
- Configure the initial load Replicat (RLOAD03)
- Start the initial load Replicat (RLOAD03)

Initial Load Extract

Before starting the initial load extract, some analysis must be conducted to determine any unsupported column data types. Since this configuration is for developing a disaster recovery (DR), all tables must be included in the capture groups. For a very large list of tables, use the following PL/SQL code or the Extract and Replicat wild-card syntax format with exclude option to prepare the ELOAD03.PRM file.

```
SQL> CREATE DIRECTORY ggs_dir as '/d01/oracle/ggs/dirprm';
```

Directory created.

```
SQL> GRANT ALL ON DIRECTORY ggs_dir to public;
```

Grant succeeded.

```
SQL> DECLARE
2     filehandler UTL_FILE.FILE_TYPE;
3     CURSOR c1 is SELECT table_name
4                   FROM   dba_tables
5                   WHERE  owner='OSM$REPAPI';
6     vtable_name VARCHAR2(30);
7 BEGIN
8     fileHandler := UTL_FILE.FOPEN('GGS_DIR', 'eload03.prm', 'W');
9     UTL_FILE.PUTF(fileHandler, 'SOURCEISTABLE\n');
10    UTL_FILE.PUTF(fileHandler, 'USERID ggs_admin, PASSWORD oracle\n');
11    UTL_FILE.PUTF(fileHandler, 'RMTHOST apps-srv2, MGRPORT 7806\n');
12    UTL_FILE.PUTF(fileHandler, 'RMTFILE ./dirdat/initload01.dat, PURGE\n');
13    OPEN c1;
14    LOOP
15        FETCH c1 into vtable_name;
16        EXIT WHEN c1%NOTFOUND;
17        UTL_FILE.PUTF(fileHandler, 'TABLE osm$repapi.'||vtable_name||';\n');
18    END LOOP;
19    CLOSE c1;
20    UTL_FILE.FCLOSE(fileHandler);
21 EXCEPTION
22     WHEN utl_file.invalid_path THEN
23         raise_application_error(-20000, 'ERROR: Invalid GoldenGate path.');
```

PL/SQL procedure successfully completed.

```
SQL>
```

```
GGSCI (apps-srv1) 1> EDIT PARAMS eload03
```

```
SOURCEISTABLE
USERID ggs_admin, PASSWORD oracle
RMTHOST apps-srv2, MGRPORT 7806
RMTFILE ./dirdat/initload03.dat, PURGE
TABLE osm$repapi.POLICY_TYPES;
TABLE osm$repapi.DISCOUNT_TYPES;
TABLE osm$repapi.CUSTOMERS;
TABLE osm$repapi.POLICIES;
```

```
GGSCI (apps-srv1) 1> SHELL ./extract pf ./dirprm/eload03.prm rf
./dirrpt/eload03.rpt
```

The GGSCI SHELL command allows you to invoke programs and operating system shell commands from within Oracle GoldenGate. The parameter SOURCEISTABLE directly reads records from the database tables, and must be the first parameter in the configuration file.

Initial Load Replicat

The initial load data file has been created on the live-standby system. Verify the existence of ./dirdat/initload03.dat on the live standby. Before starting the Replicat program, create a special run Replicat process (RLOAD03) as shown below.

```
GGSCI (apps-srv2) 1> ADD REPLICAT rload03, SPECIALRUN
REPLICAT added.
```

```
GGSCI (apps-srv2) 2>
```

```
GGSCI (apps-srv2) 2> EDIT PARAMS rload03
```

```
REPLICAT rload03
SPECIALRUN
ASSUMETARGETDEFS
HANDLECOLLISIONS
USERID ggs_admin, PASSWORD oracle
EXTFILE ./dirdat/initload03.dat
DISCARDFILE ./dirrpt/rload03.dsc, PURGE
MAP osm$repapi.*, TARGET osm$repapi.*;
END RUNTIME
```

```
GGSCI (apps-srv2) 1> SHELL ./replicat pf ./dirprm/rload03.prm rf
./dirrpt/rload03.rpt
```

Now, the live-standby database is established and identical to the primary database. Before proceeding further, verify ascertain the data using Oracle GoldenGate Veridata.

Also, because the initial-load is done from an inactive database, review Oracle GoldenGate report generated from the initial-load Replicat process.

```
GGSCI (apps-srv2) 4> VIEW REPORT ./dirrpt/rload03.rpt
```

```
Reading ./dirdat/initloa3.dat, current RBA 37987353, 200007 records
```

```
Report at 2014-01-04 17:27:04 (activity since 2014-01-04 17:26:08)
```

```
From Table OSM$REPAPI.POLICY_TYPES to OSM$REPAPI.POLICY_TYPES:
```

#	inserts:	4
#	updates:	0
#	deletes:	0
#	discards:	0

```
From Table OSM$REPAPI.DISCOUNT_TYPES to OSM$REPAPI.DISCOUNT_TYPES:
```

#	inserts:	3
#	updates:	0
#	deletes:	0
#	discards:	0

```
From Table OSM$REPAPI.CUSTOMERS to OSM$REPAPI.CUSTOMERS:
```

#	inserts:	100000
#	updates:	0
#	deletes:	0
#	discards:	0

```
From Table OSM$REPAPI.POLICIES to OSM$REPAPI.POLICIES:
```

#	inserts:	100000
#	updates:	0
#	deletes:	0
#	discards:	0

```
Last log location read:
```

```
FILE:      ./dirdat/initload03.dat
RBA:       37987353
TIMESTAMP: 2014-01-04 14:24:58.801283
EOF:       NO
READERR:   400
```

As per the report details, the initial load has been completed successfully. Refer to Chapter 3 for more details regarding initial data load techniques and options.

Bi-Directional Topology Configuration

The basic task of configuring Oracle GoldenGate to support a disaster recovery site is to have a working bi-directional data flow, from the primary system to the live-standby system and vice versa. The purpose of the basic configuration is to test the capture, route, and delivery groups on a bi-directional environment using automated database transactions developed in PL/SQL and run from the command line. Figure 5-5 depicts Oracle GoldenGate groups that are configured next.

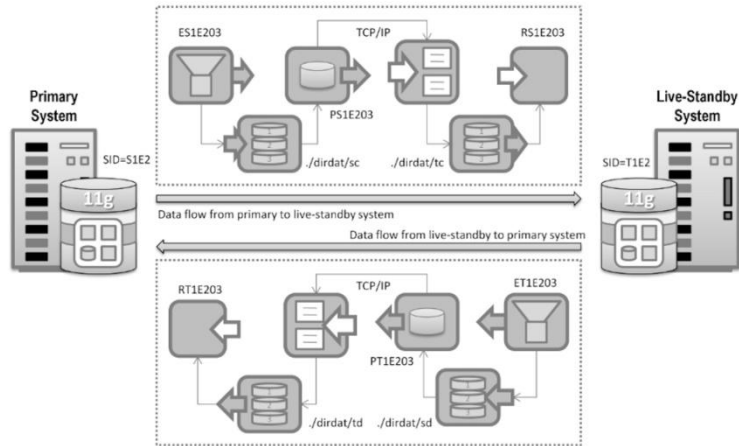


Figure 5-5: Bi-Directional Capture and Delivery Groups

After completing the bi-directional configuration, run and verify the transactional testing from both sides. The following procedure TRANS_LOAD is run as an anonymous PL/SQL block. To download the TRANS_LOAD source code, visit <http://www.oracle-ggs.com>.

```
SQL> BEGIN
2   FOR i IN 100901..101000 LOOP
3     trans_load(1, i);
4   END LOOP;
5 END;
6 /
```

PL/SQL procedure successfully completed.

SQL>

Since the procedure TRANS_LOAD does not handle primary key collision, ensure the lower and upper range of the loop statement is correctly bounded to avoid primary key violation error.

Intermediate Systems

The use of an intermediate system between the primary and live-standby systems is an option. This configuration is only recommended when the current active system needs to ship trail files to a third system, not really part of this high availability bi-directional topology. Other reasons for using an intermediate system is to offload the current active system (primary system) from routing trail records to the remote system, or network security prevents either system from sending trail records, and the

need of a dedicated system for sending trail to the third system regardless of which system (primary or live-standby) is active. The disadvantage is increasing the latency, which increases the likelihood of data conflicts. Refer to Chapter 12 for details regarding intermediate systems.

Primary to Standby Data Flow

The configuration of the primary (source) to the live-standby (target) systems data flow is composed of deploying the primary & secondary extract groups and a delivery group. Figure 5-6 shows the components configured to move data from the primary to the live-standby system.

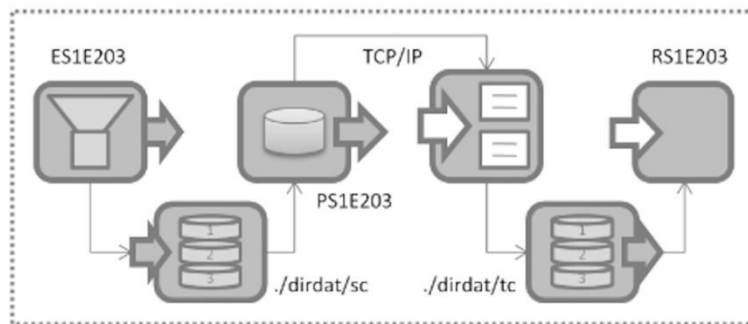


Figure 5-6: Primary to Standby Data Flow

Primary Extract Group

The primary group is composed of the parameter file ES1E203.PRM, the primary extract process ES1E203, the local trail ./dirdat/sc, and the checkpoint file ./dirchk/ES1E203.CPE. The primary extract ES1E203 is configured to capture from the transaction logs and writes to the local trail. The next two commands create the primary extract and its local trail. The local trail files is sized 128MB.

```
GGSCI (apps-srv1) 2> EDIT PARAMS es1e203
```

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME="/d01/app/oracleSE")
SETENV (NLS_LANG=AMERICAN AMERICA.AR8ISO8859P6)
USERID ggs_admin, password oracle
EXTTRAIL ./dirdat/sc
TABLE osm$repapi.CUSTOMERS;
TABLE osm$repapi.POLICIES;
```

```
GGSCI (apps-srv1) 3> ADD EXTRACT es1e203, TRANLOG, BEGIN now, THREADS 1
```

```
EXTRACT added.
```

```
GGSCI (apps-srv1) 5> ADD EXTTRAIL ./dirdat/sc, EXTRACT es1e203, MEGABYTES 128  
EXTTRAIL added.
```

```
GGSCI (apps-srv1) 6>
```

Secondary Extract Group

The secondary group is composed of the parameter file PS1E203.PRM, the data pump extract process PS1E203, the remote trail ./dirdat/tc, and the checkpoint file ./dirchk/PS1E203.CPE. The data pump extract PS1E203 sends data over TCP/IP to the live-standby system. The next commands create the secondary group (data pump) and its remote trail. The remote trail files is sized 128MB.

```
GGSCI (apps-srv1) 6> EDIT PARAMS ps1e203
```

```
GGSCI (apps-srv1) 6> EDIT PARAMS ps1e203 EXTRACT ps1e203  
USERID ggs_admin, PASSWORD oracle  
PASSTHRU  
RMTHOST apps-srv2, MGRPORT 7806  
RMTTRAIL ./dirdat/tc  
TABLE osm$repapi.CUSTOMERS;  
TABLE osm$repapi.POLICIES;
```

```
GGSCI (apps-srv1) 7> ADD EXTRACT ps1e203, EXTTRAILSOURCE ./dirdat/sc  
EXTRACT added.
```

```
GGSCI (apps-srv1) 11> ADD RMTTRAIL ./dirdat/tc, EXTRACT ps1e203, MEGABYTES 128  
RMTTRAIL added.
```

```
GGSCI (apps-srv1) 12>
```

Delivery Group

The delivery group is composed of the parameter file RS1E203.PRM, the Replicat process PS1E203, the remote trail ./dirdat/tc, and the checkpoint file ./dirchk/ES1E203.CPR. The delivery process uses the live-standby database SQL Native Call to apply transactions to the database.

```
GGSCI (apps-srv2) 2> ADD REPLICAT rs1e203, EXTTRAIL ./dirdat/tc  
REPLICAT added.
```

```
GGSCI (apps-srv2) 3>
```

Standby to Primary Data Flow

To support switchover and failover operations for planned and unplanned modes, the data flow from the standby to the primary systems must be in place. Figure 5-7 shows the data flow from the live-standby to the primary system.

The live-standby to primary data flow configuration remains inactive. It is activated only when users are moved from the primary system to the live-standby system. The application servers, where the applications are running, are configured to connect to one and only one specific system – either the primary system or the live-standby system, but not both.

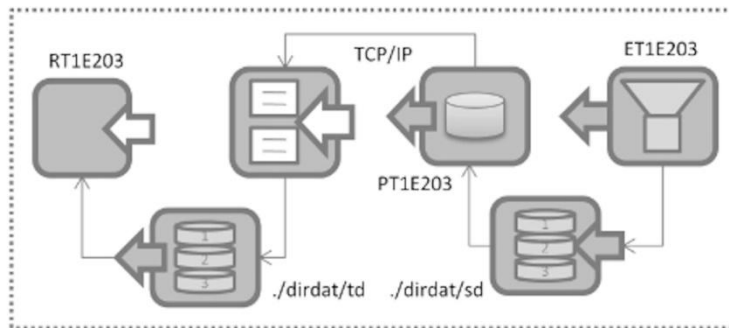


Figure 5-7: Standby to Primary Data Flow

Primary Extract Group

The primary group is composed of the parameter file ET1E203.PRM, the primary extract process ET1E203, the local trail `./dirdat/sd` and the checkpoint file `./dirchk/ET1E203.CPE`. The next commands create the primary extract and its local trail. The local trail files sized 128MB.

```
GGSCI (apps-srv2) 1> EDIT PARAMS et1e203
```

```
EXTRACT et1e203
SETENV (ORACLE_SID="T1E2")
SETENV (ORACLE_HOME="/d01/app/oracleSE/product/11.2.0/dbhome_1")
SETENV (NLS_LANG="AMERICAN AMERICA.AL32UTF8")
USERID ggs_admin, PASSWORD oracle
EXTTRAIL ./dirdat/sd
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
```

```
GGSCI (apps-srv2) 5> ADD EXTRACT et1e203, TRANLOG, BEGIN now, THREADS 1
EXTRACT added.
```

```
GGSCI (apps-srv2) 6> ADD EXTTRAIL ./dirdat/sd, EXTRACT etle203, MEGABYTES 128
EXTTRAIL added.

GGSCI (apps-srv2) 7>
```

Secondary Extract Group

The secondary group is composed of the parameter file PT1E203.PRM, the data pump extract process PT1E203, the remote trail ./dirdat/td and the checkpoint file ./dirchk/PT1E203.CPE. The next commands create the secondary group (data pump) and its remote trail. The remote trail files sized 128MB.

```
GGSCI (apps-srv2) 7> EDIT PARAMS ptle203
```

```
EXTRACT ptle203
USERID ggs_admin, PASSWORD oracle
PASSTHRU
RMTHOST apps-srv1, MGRPORT 7805
RMTRAIL ./dirdat/td
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
```

```
GGSCI (apps-srv2) 9> ADD EXTRACT ptle203, EXTTRAILSOURCE ./dirdat/sd
EXTRACT added.
```

```
GGSCI (apps-srv2) 11> ADD RMTRAIL ./dirdat/td, EXTRACT ptle203, MEGABYTES 128
RMTRAIL added.
```

```
GGSCI (apps-srv2) 12>
```

Delivery Group

The delivery group is composed of the parameter file RT1E203.PRM, the Replicat process PT1E203, the local trail ./dirdat/sd and the checkpoint file ./dirchk/ET1E203.CPR for reporting. The Replicat process ES1E203 uses a checkpoint table, which is the default location for checkpoint. The next command creates the delivery group (Replicat).

```
GGSCI (apps-srv1) 3> EDIT PARAMS rtle203
```

```
REPLICAT rtle203
SETENV (ORACLE_SID=S1E2)
SETENV (NSL_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rtle203.dsc, PURGE
MAP osm$repapi.customers, TARGET osm$repapi.customers;
```

```
MAP osm$repapi.policies, TARGET osm$repapi.policies;
```

```
GGSCI (apps-srv1) 4> ADD REPLICAT rtle203, EXTTRAIL ./dirdat/td  
REPLICAT added.
```

```
GGSCI (apps-srv1) 5>
```

This concludes Oracle GoldenGate bi-directional configuration. Under normal business operation, the data flow is always from the primary system (source) to the live-standby system (target). Users are temporary migrated to the live-standby system either due to switchover or failover operation.

Switchover and Failover

Moving users from the primary to the standby site is performed as a planned (switchover) or unplanned (failover) situation. A switchover is for well-planned specific system maintenance tasks such as primary site database upgrade, standby operating system upgrade, etc. The following sections describe the process workflow for a primary site protected by one live standby site, which may need to perform either a switchover or failover.

A switchover consists of the following tasks:

- Switchover for planned primary to standby site
- Moving users back to the primary site

A failover consists of the following tasks:

- Failover for unplanned primary to standby site
- Re-establish the primary site
- Moving users back to the primary site

During the switchover and failover operations, the primary system is referred to by (P) and the live standby system is referred to by (S). This distinction avoids executing commands on the wrong Oracle GoldenGate instance or database.

Planned Primary to Live-Standby Switchover Workflow

Database and system administrators plan and perform system maintenance tasks. For systems operating 24/7, temporarily moving users from the primary system to the live-standby system enables business continuity by sustaining the minimum downtime while ensuring data integrity. Figure 5-8 depicts the steps to move users from the primary to the live-standby system.

The two common maintenance task benefits from a planned switchover are:

- Database Upgrades
- Operating System Upgrades

Due to less resource availability on the live-standby system, the switchover is temporary. Users are moved back to the primary system after successfully performing the maintenance tasks.

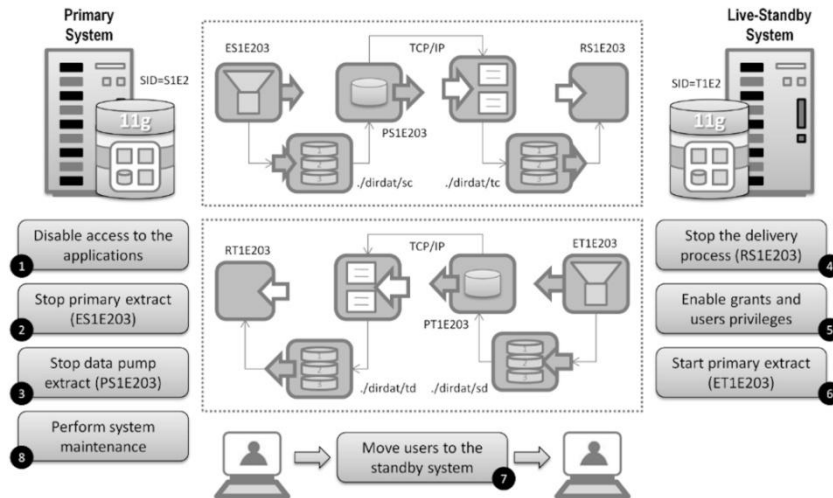


Figure 5-8: Planned Switchover from Primary to Standby System

Tasks performed on the primary system are referred to as (P), whereas tasks performed on the live-standby system referred to as (S).

Disable Access to the Applications (P)

For web-based applications accessing the database, shutdown of the application servers disables access to the database. To ascertain that no transactions were applied to the database during the switchover, after shutdown of the application servers, place the primary database on restricted sessions as shown below.

```
SQL> CONN / AS SYSDBA
Connected.
SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;

System altered.

SQL> GRANT RESTRICTED SESSION to ggs_admin;
```

```
Grant succeeded.

SQL> SELECT logins
       2 FROM v$instance;

LOGINS
-----
RESTRICTED
```

The primary and secondary extracts are still running to complete capture and routing of the transaction after the shutdown of the application servers. After finishing Step 8 of Figure 5-8, revoke the restricted session privilege from the Oracle GoldenGate database user and disable the database restricted session.

```
SQL> REVOKE RESTRICTED SESSION FROM ggs_admin;

Revoke succeeded.

SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;

System altered.

SQL>
```

Stop the Primary Extract (P)

Before stopping the primary extract (ES1E203), issue the LAG EXTRACT command until it returns the message “At EOF, no more records to process.” This indicates the primary extract has processed the transactions logs and is ready to issue the STOP EXTRACT command.

```
GGSCI (apps-srv1) 2> LAG EXTRACT es1e203

Sending GETLAG request to EXTRACT ES1E203 ...
Last record lag 1 seconds.
At EOF, no more records to process.

GGSCI (apps-srv1) 3> STOP EXTRACT es1e203

Sending STOP request to EXTRACT ES1E203 ...
Request processed.

GGSCI (apps-srv1) 4>
```

Stop the Data Pump Extract (P)

Run the LAG EXTRACT command until it returns “At EOF, no more records to process.” This indicates the data pump has sent all transactions to the live-standby system. Now, issue the STOP EXTRACT command.

```

GGSCI (apps-srv1) 5> LAG EXTRACT ps1e203

Sending GETLAG request to EXTRACT PS1E203 ...
Last record lag 4 seconds.
At EOF, no more records to process.

GGSCI (apps-srv1) 6> STOP EXTRACT ps1e203

Sending STOP request to EXTRACT PS1E203 ...
Request processed.

GGSCI (apps-srv1) 7>

```

Stop the Delivery Process (S)

Run the command LAG REPLICAT until it returns the message “At EOF, no more records to process.” This indicates that the delivery process has applied all of the transaction to the live-standby database. Now, issue the STOP REPLICAT command.

```

GGSCI (apps-srv2) 2> LAG REPLICAT rs1e203

Sending GETLAG request to REPLICAT RS1E203 ...
Last record lag 7 seconds.
At EOF, no more records to process.

GGSCI (apps-srv2) 3> STOP REPLICAT rs1e203

Sending STOP request to REPLICAT RS1E203 ...
Request processed.

GGSCI (apps-srv2) 4>

```

Enable Grants and Constraints (S)

This step is site specific. To minimize the downtime, the database DDL commands to grant and enable constraints that should be developed prior to initiating the switchover.

Start the Primary Extract (S)

Prior to moving users to the standby system, run the ALTER EXTRACT command to change the BEGIN timestamp. Then, start the primary extract only. Do not start the data pump extract. Ensure the storage level for the local trail is preserved to handle the primary system maintenance timeframe.

```

GGSCI (apps-srv2) 6> ALTER EXTRACT et1e203, BEGIN now
EXTRACT altered.

GGSCI (apps-srv2) 7> START EXTRACT et1e203

Sending START request to MANAGER ...

```

```
EXTRACT ET1E203 starting
```

```
GGSCI (apps-srv2) 8>
```

Since the manager process is configured to handle local trail space management when the trail files are shipped, they are automatically purged and keep disk space level optimized.

Switch User to the Live-Standby System

Run the necessary commands (scripts) to switchover the application servers to connect to the standby system. This enables the user to connect and perform transactions on the standby system.

Perform System Maintenance (P)

The time incurred to complete the system maintenance tasks impacts the storage level on the live-standby system. Performing the system maintenance on a separate test system minimizes the downtime of the primary system and reveals site specific issues.

The downtime for performing a planned switchover from the primary system to the live-standby system depends on Steps 5 & 7. These are the steps to run the database DDL commands to grant privileges, enable constraints, and start the application servers respectively. To successfully complete Steps 1 –7, it will take an estimated 10-30 minutes, when the commands are well-tested in the form script files.

Planned Standby to Primary Switchover Workflow

After completing the maintenance tasks on the primary system, a planned switchover from the live-standby to the primary system is performed. Figure 5-9 depicts the workflow steps to move users back to the primary system. Ascertain the current active database instance for a connected session using the following SQL statement:

```
SQL> CONN / AS SYSDBA
Connected.
SQL> COLUMN INSTANCE FORMAT A10
SQL> COLUMN DATABASE FORMAT A10
SQL> SELECT SYS_CONTEXT('USERENV', 'INSTANCE_NAME') INSTANCE,
           2 SYS_CONTEXT('USERENV', 'DB_NAME') DATABASE
           3 FROM dual;
```

INSTANCE	DATABASE
S1E2	S1E2

```
SQL>
```

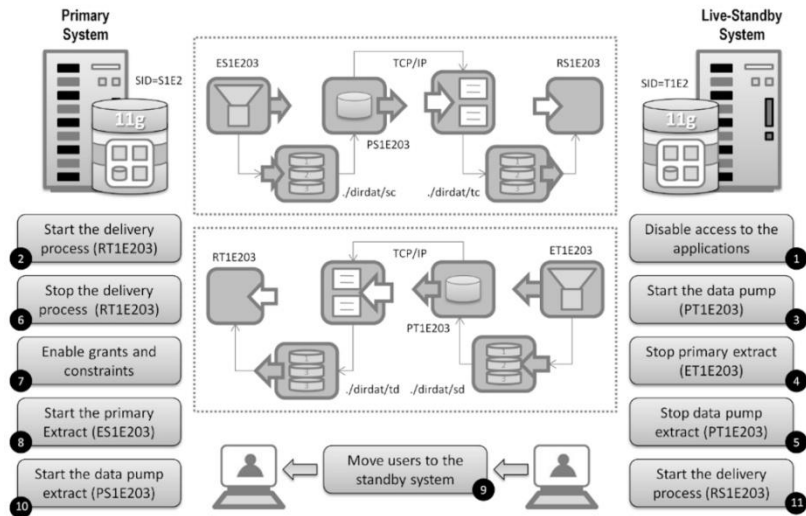


Figure 5-9: Planned Switchover from Live Standby to Primary System

Disable Application Access (S)

Start disabling users access to the applications by a shutdown of the application servers. The primary extract on the standby system remains running to capture the end of the transactions logs.

Start the Delivery Process (P)

To apply the transaction that occurred during the maintenance timeframe, start the delivery process (RT1E203).

```
GGSCI (apps-srv1) 9> START REPLICAT rtle203

Sending START request to MANAGER ...
REPLICAT RT1E203 starting

GGSCI (apps-srv1) 10>
```

Start the Data Pump (S)

Since the data pump process stopped for the duration of the maintenance on the primary system, it needs to be started to send the transactions to the primary system.

```
GGSCI (apps-srv2) 9> START EXTRACT pt1e203
```

```
Sending START request to MANAGER ...  
EXTRACT PT1E203 starting
```

Stop the Primary Extract (S)

While the applications are not running, keep issuing the command LAG EXTRACT until the message “At EOF, no more records to process” returns. This indicates all transaction logs are captured.

```
GGSCI (apps-srv2) 10> LAG EXTRACT et1e203  
  
Sending GETLAG request to EXTRACT ET1E203 ...  
No records yet processed.  
At EOF, no more records to process.  
  
GGSCI (apps-srv2) 11> STOP EXTRACT et1e203  
  
Sending STOP request to EXTRACT ET1E203 ...  
Request processed.  
  
GGSCI (apps-srv2) 12>
```

Stop the Data Pump Extract (S)

Repeatedly issue the command LAG EXTRACT until it returns the message “At EOF, no more records to process.” This indicates all transaction have been sent to the primary system. Then, issue the command STOP EXTRACT.

```
GGSCI (apps-srv2) 12> LAG EXTRACT pt1e203  
  
Sending GETLAG request to EXTRACT PT1E203 ...  
No records yet processed.  
At EOF, no more records to process.  
  
GGSCI (apps-srv2) 13> STOP EXTRACT pt1e203  
  
Sending STOP request to EXTRACT PT1E203 ...  
Request processed.  
  
GGSCI (apps-srv2) 14>
```

Stop the Delivery Process (P)

The next step is to make sure the data processed during the maintenance is applied to the primary database. Keep issuing the command LAG REPLICAT until it returns “At EOF, no more records to process.” This indicates all transactions have been applied to the primary database.

```
GGSCI (apps-srv1) 10> LAG REPLICAT rt1e203

Sending GETLAG request to REPLICAT RT1E203 ...
No records yet processed.
At EOF, no more records to process.

GGSCI (apps-srv1) 11> STOP REPLICAT rt1e203

Sending STOP request to REPLICAT RT1E203 ...
Request processed.

GGSCI (apps-srv1) 12>
```

Enable Grants and Constraints (P)

The primary system is now ready for switchover. Before doing so, run the database DDL command to grant privileges and enable constraints.

Start the Primary Extract (P)

Start the primary extract to capture from the transaction logs.

```
GGSCI (apps-srv1) 12> START EXTRACT es1e203

Sending START request to MANAGER ...
EXTRACT ES1E203 starting
```

Switch User Activity

Starting the application servers allows users to connect to the primary system. Now users are connected back to the primary system. Trails created need to be shipped to the live standby system.

Start the Data Pump (P)

Start the data pump to begin sending data to the live-standby system. In the case of data pump process failure due to connection rejection by the target system, verify the hostname or IP address is reachable by performing PING from within GGSCI command-line and the manager listening port on the live-standby system is correct.

```
GGSCI (apps-srv1) 13> START EXTRACT ps1e203

Sending START request to MANAGER ...
EXTRACT PS1E203 starting

GGSCI (apps-srv1) 14>
```

Start the Delivery Process (S)

The delivery process (RS1E203) is started to keep applying transactions to the live standby system.

```
GGSCI (apps-srv2) 14> START REPLICAT rs1e203

Sending START request to MANAGER ...
REPLICAT RS1E203 starting

GGSCI (apps-srv2) 15>
```

On the primary system, verify the status of the primary and secondary extracts. This is where users are currently executing transactions. On the live standby system, verify the status of the delivery process. Using tools such Oracle GoldenGate Veridata to compare the primary and live-standby databases allows database administrators to discover and fix missing rows, proactively. Another effective administration tool is the Oracle GoldenGate 11g Monitor, which is used for monitor health checks and diagnostics.

Unplanned Primary to Standby Failover Workflow

Situations such as a primary system disk failure are unplanned situations where users are moved to the live standby system until the primary system is recovered. Figure 5-10 depicts the steps to move users from the primary to the live-standby system. The unplanned failover from the primary to live-standby system presumes the live-standby Oracle GoldenGate configuration is in-place, and the database scripts to enable constraints are available.

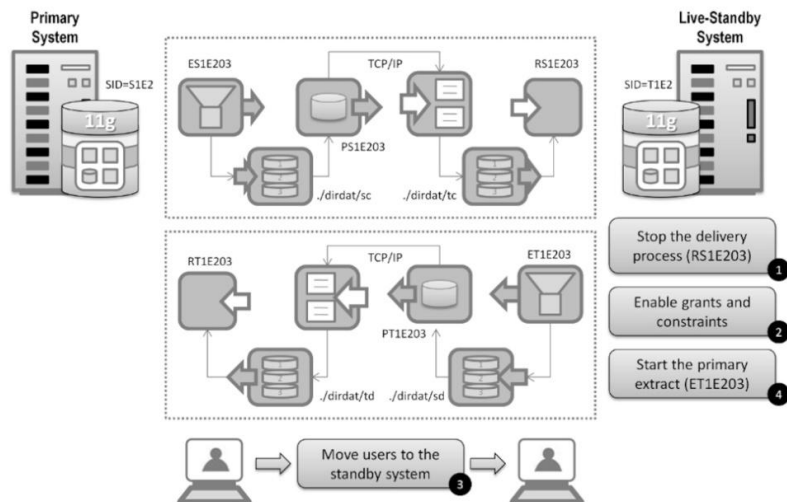


Figure 5-10: Primary to Standby Failover

Stop the Delivery Process (S)

Ensure transactions have been applied to the live standby system by continuing to issue the command LAG REPLICAT, until it returns “At EOF, no more records to process.” This indicates all transactions have been applied to the live standby database.

```
GGSCI (apps-srv2) 1> LAG REPLICAT rs1e203

Sending GETLAG request to REPLICAT RS1E203 ...
Last record lag 8 seconds.
At EOF, no more records to process.

GGSCI (apps-srv2) 2> STOP REPLICAT rs1e203

Sending STOP request to REPLICAT RS1E203 ...
Request processed.

GGSCI (apps-srv2) 3>
```

Enable Grants and Constraints (S)

Since grants and constraints remain disabled during normal operation, before moving users, run the database DDL commands to grant database privileges and enable constraints.

Start the Primary Extract (S)

By starting the primary extract only, it continuously captures the transactions to the local trail, so that it is sent by the data pump when moving users back to the primary system.

```
GGSCI (apps-srv2) 4> START EXTRACT et1e203

Sending START request to MANAGER ...
EXTRACT ET1E203 starting

GGSCI (apps-srv2) 5>
```

Switch Users to Primary System

Redirect the application servers to the standby system. Since the primary system is not available until it is fixed, ensure the disk space is monitored to preserve the size of the local trail because the data pump is not available.

Unplanned Standby to Primary Failover Workflow

In the event of an unplanned failover, users are quickly moved to the standby system where business continues to operate. The downtime associated with an unplanned failover is minimal and mostly proportional to the unapplied amount of the remote trail files currently being applied to the live-standby system.

The next step is to determine the cause of the primary system failure and re-instantiate it from the live-standby system in preparation to move users back to the primary system. Figure 5-11 depicts the workflow for an unplanned live-standby to primary system failover.

The amount of downtime is greatly minimized by performing an initial load from the active live-standby system, then applying the transactions that occurred during the initial load timeframe to the primary system database. When Step 4 is completed, users are unable to execute transactions until Step 12 is completed (which enables the primary system database grants and constraints). Step 16 includes starting the application servers and moving users back to the primary system. The amount of downtime varies and depends on the size of the live-standby system local trail.

Task performed on the primary system referred to by (P), whereas task performed on the live-standby system referred to by (S).

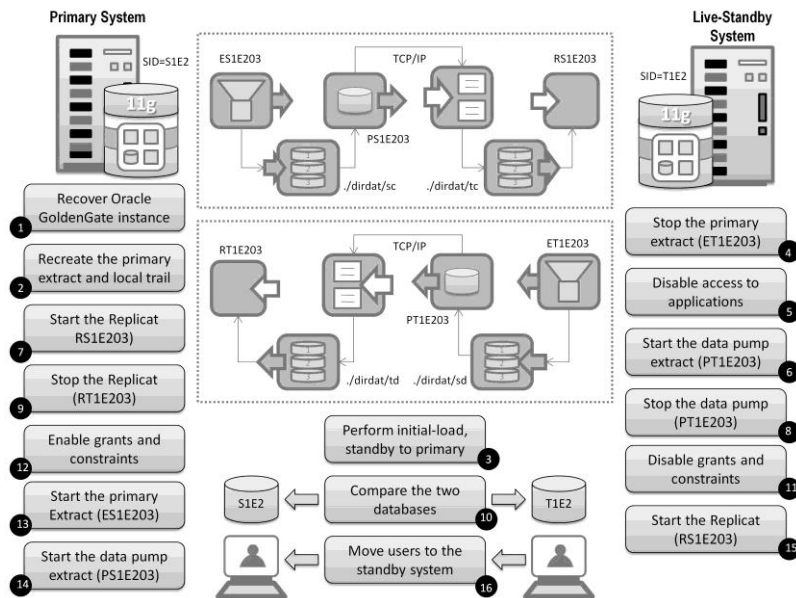


Figure 5-11: Unplanned Standby to Primary System Failover

Recover Oracle GoldenGate Instance (P)

Recovering Oracle GoldenGate instance may require installing the software, preferably to the same location to avoid modifying the instance parameter files. Then, restore the parameter files from the backup and start the manager process, using the same TCP/IP port to avoid modifying the manager and data pump parameter files. Verify the manager is running on the same port.

```
GGSCI (apps-srv1) 19> START mgr
Manager started.

GGSCI (apps-srv1) 20> INFO mgr

Manager is running (IP port apps-srv1.7805, Process ID 11252).

GGSCI (apps-srv1) 21>
```

Recreate the Primary Extract (P)

Regardless of the type of system failure, deleting and creating the primary extract is necessary. This avoids any type of reference to the old primary group.

```
GGSCI (apps-srv1) 13> DELETE EXTTRAIL ./dirdat/sc
Deleting extract trail ./dirdat/sc for extract ES1E203

GGSCI (apps-srv1) 14> DELETE EXTRACT es1e203
Deleted EXTRACT ES1E203.

GGSCI (apps-srv1) 15>
```

```
GGSCI (apps-srv1) 15> ADD EXTRACT es1e203, TRANLOG, BEGIN now, THREADS 1
EXTRACT added.

GGSCI (apps-srv1) 16> ADD EXTTRAIL ./dirdat/sc, EXTRACT es1e203, MEGABYTES 128
EXTTRAIL added.

GGSCI (apps-srv1) 17>
```

Perform Initial Load

There are several techniques to establish the primary database from the live-standby database system. Refer to Chapter 3 for details regarding the available techniques. Oracle GoldenGate direct load technique is illustrated below.

Before performing initial load to establish the primary system database using the direct load method, the database table structure must be prepared. Several techniques are available including the Oracle Database Data Pump Utility.

For direct initial load method, start by preparing the initial load delivery process – as it will be referenced by the initial load extract. The objective is to perform initial load from an active live-standby system database to minimize the downtime.

Create the initial load Replicat using the SPECIALRUN option.

```
GGSCI (apps-srv1) 21> EDIT PARAMS rload03
```

```
REPLICAT rload03
SPECIALRUN
ASSUMETARGETDEFS
HANDLECOLLISIONS
USERID ggs_admin, PASSWORD oracle
EXTFILE ./dirdat/initload03.dat
DISCARDFILE ./dirrpt/rload03.dsc, PURGE
MAP osm$repapi.*, TARGET osm$repapi.*;
END RUNTIME
```

```
GGSCI (apps-srv1) 22> ADD REPLICAT rload03, SPECIALRUN
REPLICAT added.
```

```
GGSCI (apps-srv1) 23>
```

Configure and create the initial load extract, ELOAD03. This extract reads from the active standby database and performs direct inserts using an initial load delivery process.

```
GGSCI (apps-srv2) 6> EDIT PARAMS eload03
```

```
EXTRACT eload03
USERID ggs admin, PASSWORD oracle
RMTHOST apps-srv1, MGRPORT 7805
RMTTASK REPLICAT, GROUP rload03
TABLE osm$repapi.policy_types;
TABLE osm$repapi.discount types;
TABLE osm$repapi.policies;
TABLE osm$repapi.customers;
```

```
GGSCI (apps-srv2) 7> ADD EXTRACT eload03, SOURCEISTABLE
EXTRACT added.
```

```
GGSCI (apps-srv2) 8>
```

The primary system database is established by starting the initial load extract of the live standby system.

```
GGSCI (apps-srv2) 16> START EXTRACT eload03
```

```
Sending START request to MANAGER ...
EXTRACT ELOAD03 starting
```

```
GGSCI (apps-srv2) 17>
```

The following shows part of the report to verify the successful completion of the initial load.

```
GGSCI (apps-srv1) 5> VIEW REPORT rload03
```

```
*****
*                               ** Run Time Statistics **                               *
*****
```

Report at 2014-01-24 17:28:40 (activity since 2014-01-24 17:27:48)

```
From Table OSM$REPAPI.POLICY_TYPES to OSM$REPAPI.POLICY_TYPES:
#               inserts:           4
#               updates:           0
#               deletes:           0
#               discards:          0
From Table OSM$REPAPI.DISCOUNT_TYPES to OSM$REPAPI.DISCOUNT_TYPES:
#               inserts:           3
#               updates:           0
#               deletes:           0
#               discards:          0
From Table OSM$REPAPI.POLICIES to OSM$REPAPI.POLICIES:
#               inserts:        102100
#               updates:           0
#               deletes:           0
#               discards:          0
From Table OSM$REPAPI.CUSTOMERS to OSM$REPAPI.CUSTOMERS:
#               inserts:        102100
#               updates:           0
#               deletes:           0
#               discards:          0
```

Disable Application Access (S)

Stop the applications. This is “application server dependent.”

Stop the Primary Extract (S)

Stop the primary extract in preparation to applying the changes that occurred during the initial load timeframe. Issue the LAG EXTRACT before stopping the extract, this ensures all transactions have been processed by the extract.

```
GGSCI (apps-srv2) 19> LAG EXTRACT etle203
```

```
Sending GETLAG request to EXTRACT ET1E203 ...
Last record lag 2 seconds.
At EOF, no more records to process.
```

```
GGSCI (apps-srv2) 20> STOP EXTRACT et1e203

Sending STOP request to EXTRACT ET1E203 ...
Request processed.

GGSCI (apps-srv2) 21>
```

Start the Data Pump (S)

The data pump is started to send data to the primary system. These are the changes that occurred during the initial load.

```
GGSCI (apps-srv2) 21> START EXTRACT pt1e203

Sending START request to MANAGER ...
EXTRACT PT1E203 starting

GGSCI (apps-srv2) 22>
```

Start the Delivery Process (P)

Apply the changes to the primary database. Ensure handling collision to avoid a unique constraint error.

```
GGSCI (apps-srv1) 7> START REPLICAT rt1e203

Sending START request to MANAGER ...
REPLICAT RT1E203 starting

GGSCI (apps-srv1) 8>
```

Stop the Data Pump (S)

Stop the data pump on the standby system. Keep issuing the command LAG EXTRACT until it returns “At EOF, no more records to process.” This indicates all transactions have been processed.

```
GGSCI (apps-srv2) 23> LAG EXTRACT pt1e203

Sending GETLAG request to EXTRACT PT1E203 ...
Last record lag 621,259 seconds.
At EOF, no more records to process.

GGSCI (apps-srv2) 24> STOP EXTRACT pt1e203

Sending STOP request to EXTRACT PT1E203 ...
Request processed.
```

```
GGSCI (apps-srv2) 25>
```

Stop the Delivery Process (P)

Stop the delivery process on the primary system. Keep issuing the command LAG REPLICAT until it returns “At EOF, no more records to process.” This indicates all transactions have been applied to the primary database.

```
GGSCI (apps-srv1) 9> LAG REPLICAT rt1e203

Sending GETLAG request to REPLICAT RT1E203 ...
Last record lag 621,594 seconds.
At EOF, no more records to process.

GGSCI (apps-srv1) 10> STOP REPLICAT rt1e203

Sending STOP request to REPLICAT RT1E203 ...
Request processed.

GGSCI (apps-srv1) 11>
```

Compare Primary and Standby Databases

For data consistency, prior to proceeding, compare the primary and standby databases. Use tools such as Oracle GoldenGate Veridata to discover the missing rows or SQL*Plus to compare table rows count. For now, compare the two databases using rows count for tables CUSTOMERS and POLICIES using SQL*Plus for the tables.

```
SQL> conn osm$repapi/oracle
Connected.
SQL> select count(*) from customers;

COUNT(*)
-----
102100

SQL>
```

Execute the statement for the databases S1E2 and T1E2. Refer to Chapter 6 to learn more about Oracle GoldenGate Veridata.

Disable Grants and Constraints (S)

The receiving system must have grants and constraints disabled. This avoids referential integrity violations.

Enable Grants and Constraints (P)

Enable the database grants and constraints on the primary system.

Start the Primary Extract (P)

Start the primary extract. Since the primary was recreated, it will start transaction logs capture from the creation timestamp.

```
GGSCI (apps-srv1) 13> START EXTRACT es1e203

Sending START request to MANAGER ...
EXTRACT ES1E203 starting

GGSCI (apps-srv1) 14>
```

Start the Data Pump (P)

Start the data pump. This starts sending data to the standby system.

```
GGSCI (apps-srv1) 15> START EXTRACT ps1e203

Sending START request to MANAGER ...
EXTRACT PS1E203 starting

GGSCI (apps-srv1) 16>
```

Start the Delivery Process (S)

Start the delivery process to apply transactions sent to the standby system.

```
GGSCI (apps-srv2) 25> START REPLICAT rs1e203

Sending START request to MANAGER ...
REPLICAT RS1E203 starting

GGSCI (apps-srv2) 26>
```

Move Users to the Primary System

Now, confirm the Replicat has applied all the transactions to the standby system database, start the application servers to move users back to the primary system. This step is application server/site specific. For example, if it's PL/SQL portal application such as Oracle APEX, it's changing the database connect string only from the live-standby system to the primary system database.

High Availability

Oracle High Availability has been always associated with Oracle Real Application Clusters (RAC), where within the same data center, multiple tightly coupled servers (nodes) access the same database. Another advanced architecture of RAC is Extended RAC, where nodes are loosely coupled and spread across two or more data centers. Figure 5-12 compares the high availability of Oracle GoldenGate and Extended RAC.

The upper part of Figure 5-12 is Oracle GoldenGate configured using an active-active bi-directional topology model. Over the TCP/IP network, data flows from the site A (node-1) to site B (node-2) and vice versa. The self-evident advantage of this solution is the unlimited distance between node-1 and node-2. The drawback is the chance of scalability and data conflict handling. However, because the data is delivered in canonical format for a well-developed application, node-1 and node-2 may use different platforms. For example node-1 is Oracle Database 11g on Oracle Solaris while node-2 is Oracle Database 11g on Oracle Linux. Avoid heterogeneous database architecture for a high availability solution, unless the applications developed to support all database types transparently.

The lower part of Figure 5-12 shows Oracle Extended RAC. The databases are mirrored using dense wavelength division multiplexing (DWDM) over a dark fiber link. This enables always in-sync copies of the database. The self-evident advantage is out-of-box scalability and high availability. The drawback is the distance limitation between site A and site B for up to 100 kilometers and cost-factor.

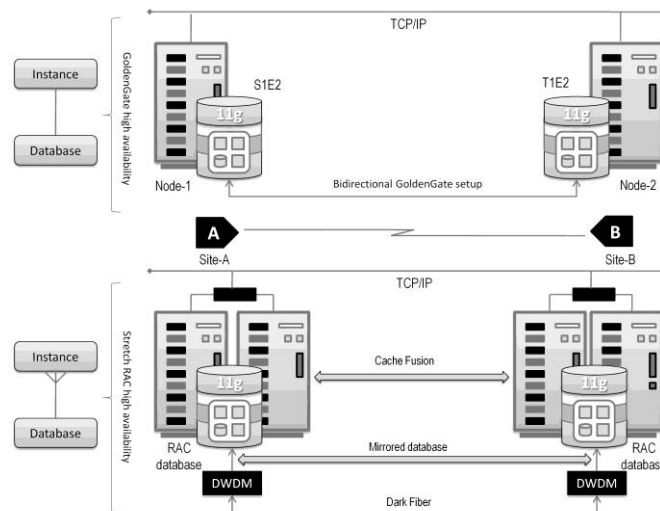


Figure 5-12: Comparison of 2-Node Oracle GoldenGate and Extended RAC

When using Oracle GoldenGate to implement a high availability platform, it is configured as an active-active bi-directional topology model. The source system (node-1) and target system (node-2) are available for user connections. Users may explicitly connect to either system based on predetermined attributes such as location, range of TCP/IP address, built-in application context, etc.

Implicit connections will be based on database and/or system workload. Users requests are routed to the system with minimal workload. This provides connection time workload evaluation to distribute database sessions across both systems. Figure 5-12 shows an implicit form of handling users connections. The database load balancer evaluates the preferred database, which is the database with lesser workload. Then, send the request to the application server which has its own HTTP load balancer. For more details regarding a high availability load balancer, refer to Figure 5-2.

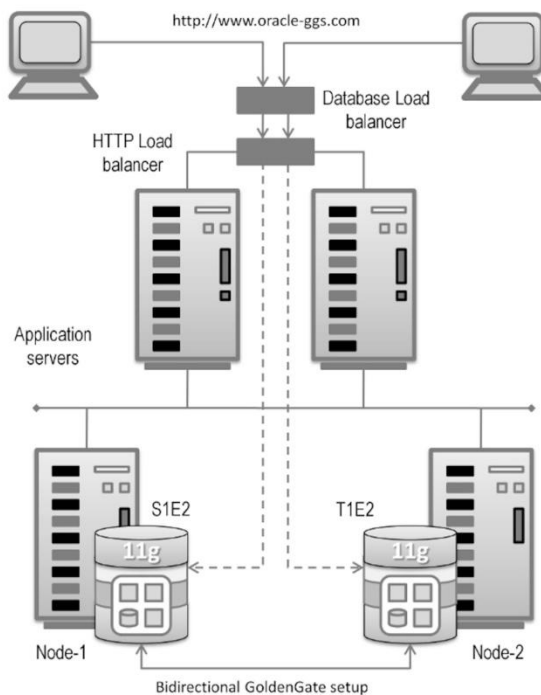


Figure 5-13: Load Balancing on Bi-Directional Topology

Applications running on Windows such as Oracle for .NET employ Oracle SQLNET to connect to the database. Whereas Java desktop applications employs JDBC drivers to connect to the database. For this type of applications, HTTP Servers are eliminated, the client applications establish stateful types of database connections.

Bi-Directional Replication Considerations

Since data manipulation language (DML) statements may occur at 'almost' the same time from either direction, the below must be considered and configured on a bi-directional active-active Oracle GoldenGate configuration.

Endless Loop Detection

In an active-active bi-directional topology, because the Extract and the Replicat are operating on the same table, the Replicat must be prevented from sending the same applied record back to the source to avoid an endless loop situation.

There are two techniques for loop detection to avoid re-extracting and re-replicating the same record:

- Replicat-Extract trace table
- Replicat-Extract parameter

Replicat-Extract Trace Table

When the Replicat applies transactions to the target database, it writes to a trace table which the Extract on the same system uses for loop detection by eliminating the Replicat transactions. When the Replicat starts, if the parameter TRACETABLE is configured then it uses the specified trace table. If the parameter TRACETABLE is not configured, it tries the default trace table name GGS_TABLE owned by the Oracle GoldenGate user of the DBLOGIN parameter. The following commands illustrate a non-default trace table GGS_T1E2_TRC is created and configured as follows:

- Create the trace table GGS_T1E2_TRC. This requires connecting to the database from within Oracle GoldenGate and issues the two GGSCI commands below:

```
GGSCI (apps-srv2) 1> DBLOGIN USERID ggs_admin, PASSWORD oracle
Successfully logged into database.
```

```
GGSCI (apps-srv2) 2> ADD TRACETABLE ggs_admin.ggs_t1e2_trc
```

```
Successfully created trace table GGS_ADMIN.GGS_T1E2_TRC.
```

```
GGSCI (apps-srv2) 3>
```

- Add the parameter TRACETABLE to the Extract and the Replicat parameter file. The GGS_ADMIN is the Oracle GoldenGate database schema.

```
TRACETABLE ggs_admin.GGS_T1E2_TRC
```

Replicat Extract Parameter

The second method uses the Replicat TRANLOGOPTIONS parameter only. This method eliminates the processing overhead associated with using the database trace table explained above. The option EXCLUDEUSER or EXCLUDEUSERID of the parameter TRANLOGOPTIONS explicitly avoid the re-capture of the transactions applied by specified user. This requires using a dedicated Oracle GoldenGate user so that transactions applied by this user are eliminated – avoiding re-sending the transaction back to the originated system. The TRANLOGOPTIONS is shown next.

TRANLOGOPTIONS EXCLUDEUSER ggs_admin

Query DBA_USERS to replace the username with the user ID.

```
SQL> SELECT username, user_id
       2 FROM dba_users
       3 WHERE username='GGS_ADMIN';
```

USERNAME	USER_ID
GGS_ADMIN	1101

SQL>

Data Conflicts

This issue is related to data integrity that must be addressed comprehensively in an active-active bi-directional configuration. There are several types of conflict, avoidance, and detection techniques. Because applications are mostly not replication-ware, resolving data conflict is required and should be governed by the business rules. Oracle GoldenGate best practices recommend resolving data conflicts in the following order:

- Apply avoidance techniques of data segmentation and application partitioning
- Adapt Oracle GoldenGate automatic conflict detection and resolution (CDR)
- Develop custom conflict detection and resolution routines

Details regarding data conflicts and detection techniques are described in this chapter.

Truncate Table

Since the TRUNCATE TABLE command is not supported in an active-active bi-directional configuration, at any point in time, Extracts and Replicats must be configured for one-and-only-one dedicated system to issue the TRUNCATE TABLE database command. This is achieved by adding the parameter GETTRUNCATES

on the allowed system and the parameter IGNORETRUNCATES on the disallowed system. To ensure the TRUNCATE TABLE command is issued from the allowed database only, revoke the TRUNCATE TABLE privilege from the application roles.

Database Sequences

When using database sequence numbers for tables primary and unique keys values on an active-active bi-directional configuration, avoid primary and unique key uniqueness conflicts. There are several techniques for implementing database sequences on an active-active bi-directional configuration. Common techniques for using database sequences are:

- Node-1 database takes odd numbers, while node-2 database takes even numbers
- Node-1 database uses a different range of numbers than node-2 database
- Implement database sequence values based on built-in application contexts

Details regarding database sequence implementations are described in this chapter.

Real-Time Disconnected Transactions

When the applications retrieve and manipulate data synchronously from either of the two databases, database links are created which enable the application to fetch and to manipulate data from either databases. There is no data conflict. This is known as distributed database applications and adhere to database dependency, locks, and resource management as illustrated below.

Create two database links from S1E2 to T1E2 and from T1E2 to S1E2. Before creating the database link, ensure the database connect string is able to establish remote connection.

Create the database link from S1E2 to T1E2.

```
SQL> CREATE PUBLIC DATABASE LINK t1e2.precisetrace.com
  2  CONNECT TO osm$repapi IDENTIFIED BY oracle
  3  USING 'T1E2';
```

Database link created.

SQL>

Create the database link from T1E2 to S1E2.

```
SQL> CREATE PUBLIC DATABASE LINK s1e2.precisetrace.com
  2  CONNECT TO osm$repapi IDENTIFIED BY oracle
  3  USING 'S1E2';
```

Database link created.

SQL>

The following update statement executed remotely on S1E2 over the database link. Do not issue COMMIT statement to continue locking resources.

```
SQL> host hostname
apps-srv2
```

```
SQL> CONN osm$repapi/oracle
Connected.
SQL> UPDATE customers@s1e2 SET
  2     cust_email='falswaimil@gmail.com'
  3 WHERE cust_no=100100;
```

1 row updated.

SQL>

The following update statement waits for the above remote transaction to release resources, issuing COMMIT or ROLLBACK, then execute the update.

```
SQL> host hostname
apps-srv1
```

```
SQL> CONN osm$repapi/oracle
Connected.
SQL> UPDATE customers SET
  2     cust_email='falswaimil@yahoo.com'
  3 WHERE cust_no=100100;
```

SQL>

This illustrates how synchronous transactions used by distributed applications do not trigger data conflict. They adhere to the target database resource management, locks, and queue handling.

However, for near real-time ‘disconnected’ transactions that are running asynchronously, the need to consider and handle data conflict caused by executing the applications is very obvious. Conflicts are either avoided using data and applications segmentation or handled using Oracle GoldenGate built-in and custom conflict detection and resolution. The section ‘conflicts resolution’ describes the built-in and advance custom techniques for handling data conflict.

Applications and Database Design for High Availability

The key to avoid data conflict in an asynchronous environment is to partition data and application dynamically, which is referred to as segmentation. This guarantees

an application's group interacts and manipulates a logically partitioned subset of the database that is isolated from others application's groups. This is the fundamental concept of virtual private database (VPD). Figure 5-14 shows two application's groups in an active-active bi-directional topology model that are designed to be virtually isolated.

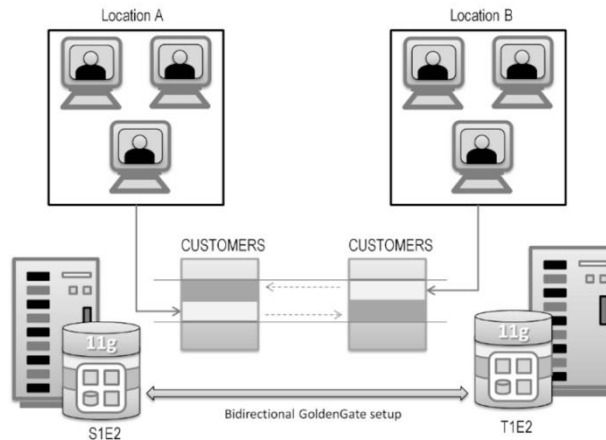


Figure 5-14: Conflict Avoidance Using Data and Applications Segmentation

The above data conflict avoidance technique is either implemented at the applications level or the database. Best practices recommend using the Oracle Database Enterprise Edition Virtual Private Database (VPD) feature. VPD uses policy functions for dynamically modifying SQL Statements to enforce data segmentation through data access control based on application contexts. Virtual Private Database implementation is explained in more detail in the section ‘Virtual Private Database.’

An valuable feature for implementing segmentation by location is performance. Users are located in close proximity to access data. For example, a call center with two locations, where each location handles sales for specific products list avoids data conflict and has local data access which eliminates the overhead associated with network failure. Users continue to enter transactions locally. When the network becomes available, Oracle GoldenGate starts routing data to a remote system.

Bi-Directional Extract and Delivery Groups

The following Oracle GoldenGate configuration is built on the components used by the disaster recovery described above. Since Oracle GoldenGate high availability is deployed as an active-active bi-directional topology architecture, it is recommended to divide it into two phases:

Phase I: Working Bi-Directional Topology Architecture – An end-to-end optimized and well-tested bi-directional replication capable of handling the transactions volume for near real-time data movement. Data conflict must be addressed as it's an unavoidable.

Phase II: Comprehensive Conflicts Handling – Implementing avoidance and detection techniques for handling data conflicts. Since Oracle GoldenGate operates asynchronously, near real-time data movement minimizes conflict occurrences.

Figure 5-15 outlines the groups and trails. The upper part is for node-1 to node-2 data flow and the lower part is the opposite configuration, from node-2 to node-1.

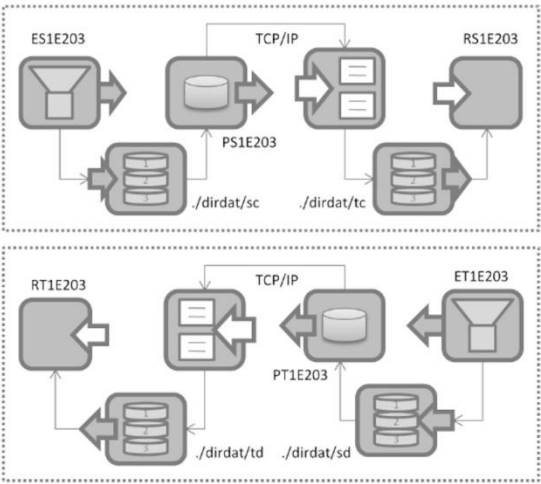


Figure 5-15: Bi-Directional High Availability Topology

Node-1 Parameter Files, Extracts, and Replicats Processes

Node-1 Oracle GoldenGate instance operates the processes listed in Table 5-2. Since an end-to-end data integrity among referenced tables must be enforced, designing multiple capture-and-delivery groups requires understanding the data model, our illustration is based on a single process architecture.

Component	Description
ES1E203	Primary capture process from the primary system transactions logs
./dirdat/sc	Local trail
PS1E203	Secondary extract data pump process send data to node-2 system
./dirdat/tc	Remote trail
RT1E203	Replicat process for node-2 database

Table 5-2: Node-1 Components

The parameter files must be modified to support bi-directional replication for conflict handling. Re-start the processes to activate the newly added parameters.

Node-1 primary extract process ES1E203 is shown below. Similarly modify Node-2 primary extract.

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME="/d01/app/oracleSE")
SETENV (NLS_LANG=AMERICAN_AMERICA.AR8ISO8859P6)
USERID ggs_admin, password oracle
EXTTRAIL ./dirdat/sc
TABLE osm$repapi.CUSTOMERS;
TABLE osm$repapi.POLICIES
```

Node-2 delivery process RS1E203 is shown below. Similarly modify node-1 delivery process.

```
REPLICAT rt1e203
SETENV (ORACLE_SID=S1E2)
SETENV (NLS_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rt1e203.dsc, PURGE
MAP osm$repapi.customers, TARGET osm$repapi.customers;
MAP osm$repapi.policies, TARGET osm$repapi.policies;
```

Verify the status of the processes on node-1 Oracle GoldenGate instance. The status of RUNNING indicates the Extract, the data pump, and the Replicat processes are configured correctly, at least.

The current status of the primary extract, ES1E203.

```
GGSCI (apps-srv1) 4> INFO EXTRACT es1e203

EXTRACT      ES1E203      Last Started 2014-01-28 22:23      Status RUNNING
Checkpoint Lag      03:42:54 (updated 00:00:05 ago)
Process ID          6378
Log Read Checkpoint Oracle Redo Logs
                    2014-01-28 18:41:00 Thread 1, Seqno 30, RBA 39536
                    SCN 0.1997389 (1997389)
```

The local trail details, ./dirdat/sc.

```
GGSCI (apps-srv1) 5> INFO EXTTRAIL ./dirdat/sc

Extract Trail: ./dirdat/sc
Extract: ES1E203
Seqno: 10
RBA: 1601
```

File Size: 128M

The current status of the secondary (data pump) extract, PES1E203.

```
GGSCI (apps-srv1) 8> INFO EXTRACT ps1e203
```

```
EXTRACT      PS1E203      Last Started 2014-01-28 22:25      Status RUNNING
Checkpoint Lag      100:45:36 (updated 00:00:01 ago)
Process ID          6410
Log Read Checkpoint      File ./dirdat/sc000009
                        2014-01-24 17:40:22.000000      RBA 44546492
```

The current status delivery process from standby to primary system, RT1E203.

```
GGSCI (apps-srv1) 10> INFO REPLICAT rtle203
```

```
REPLICAT     RT1E203     Last Started 2014-01-28 22:27      Status RUNNING
Checkpoint Lag      00:00:00 (updated 00:00:08 ago)
Process ID          6428
Log Read Checkpoint      File ./dirdat/td000008
                        First Record      RBA 45282
```

```
GGSCI (apps-srv1) 11>
```

Node-2 Parameter Files, Extracts, and Replicats Processes

Node-2 Oracle GoldenGate instance operates the processes listed in Table 5-3. When Oracle GoldenGate is configured for high availability, node-1 and node-2 systems should have similar resources in terms of CPU speed and RAM size. This allows similar workload handling across the two nodes.

Component	Description
ET1E203	Primary Extract process from node-2 transactions logs
./dirdat/sd	Local trail
PT1E203	Secondary extract data pump process from node-2 to node-1
./dirdat/td	Remote trail
RS1E203	Replicat process on node-1

Table 5-3: Node-2 Components

Verify the status of the processes on node-2 Oracle GoldenGate instance.

```
GGSCI (apps-srv2) 4> INFO EXTRACT et1e203
```

```
EXTRACT      ET1E203      Last Started 2014-01-28 22:31      Status RUNNING
Checkpoint Lag      100:58:00 (updated 00:00:07 ago)
Process ID          17047
Log Read Checkpoint      Oracle Redo Logs
                        2014-01-24 17:33:26      Thread 1, Seqno 25, RBA 42078224
                        SCN 0.1889870 (1889870)
```

```

GGSCI (apps-srv2) 6> INFO EXTRACT pt1e203

EXTRACT      PT1E203      Last Started 2014-01-28 22:32      Status RUNNING
Checkpoint Lag      273:31:53 (updated 00:00:04 ago)
Process ID          17083
Log Read Checkpoint File ./dirdat/sd000005
                    2014-01-17 13:00:31.000000      RBA 45235

GGSCI (apps-srv2) 8> INFO REPLICAT rs1e203

REPLICAT      RS1E203      Last Started 2014-01-28 22:32      Status RUNNING
Checkpoint Lag      00:00:29 (updated 00:00:00 ago)
Process ID          17097
Log Read Checkpoint File ./dirdat/tc000016
                    2014-01-28 22:32:32.880946      RBA 1686454

GGSCI (apps-srv2) 9>

```

Oracle Sequence for Artificial Primary Key Values

Oracle database sequence is a memory-based object. Hence, it is an efficient technique for generating primary key values. For a non-replicated database, sequences are suitable for generating primary key values without major issues. However, when it is used in a database synchronized by Oracle GoldenGate, it should be created to avoid primary key conflict for SQL insert statements. The following illustrates techniques for using Oracle sequences in databases synchronized in real-time by Oracle GoldenGate.

Odd-Even Numbers Distribution

The node-1 database fetches the odd number values only, while node-2 database fetches the even number values. Figure 5-16 depicts the sequence design created below:

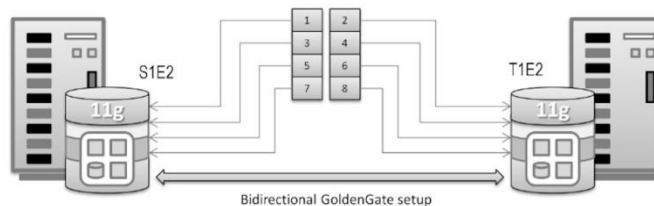


Figure 5-16: Odd-Even Numbers Sequencing

Node-1 database sequence is created to start with 1 and fetches odd numbers.

```

SQL> CREATE SEQUENCE cust_no_seq
      2 START WITH 1
      3 INCREMENT BY 2;

```

Sequence created.

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL  
-----  
          1
```

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL  
-----  
          3
```

```
SQL>
```

Node-2 database sequence is created to start with 2 and fetches even numbers.

```
SQL> CREATE SEQUENCE cust_no_seq  
      2 START WITH 2  
      3 INCREMENT BY 2;
```

Sequence created.

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL  
-----  
          2
```

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL  
-----  
          4
```

```
SQL>
```

Range Numbers Distribution

Node-1 database fetches from one range while node-2 database fetches from a second range. Figure 5-17 depicts the sequence design created below:

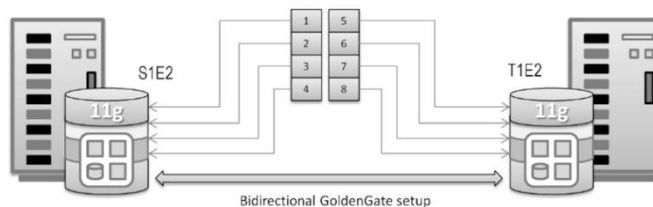


Figure 5-17: Range Numbers Sequencing

Node-1 database begins from 1 to n-1, where n is node-2 and begins sequence number.

```
SQL> CREATE SEQUENCE cust_no_seq  
2 START WITH 1;
```

Sequence created.

Node-2 begins at n, where n=5.

```
SQL> CREATE SEQUENCE cust_no_seq  
2 START WITH 5;
```

Sequence created.

SQL>

Application Context of Numbers Distribution

The following illustrates the use of sequence to generate unique values based on the application context. Figure 5-18 depicts the application context value generator for a two-node architecture.

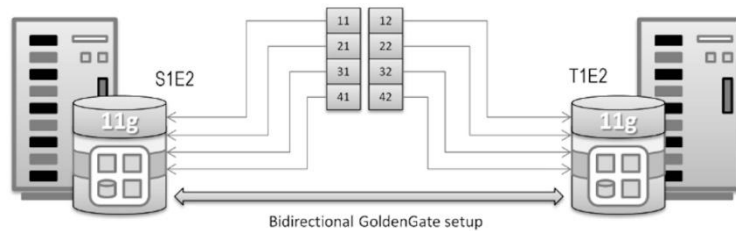


Figure 5-18: Numbers Sequencing based on Application Context

```
SQL> CREATE SEQUENCE cust_no_seq  
2 START WITH 1;
```

Sequence created.

SQL>

Use the database function SYS_CONTEXT to concatenate the sequence number with the instance number (INSTANCE) for the session. This technique is similar to a sequence created with INCREMENT BY 10.

```
SQL> CREATE SEQUENCE cust_no_seq;
```

Sequence created.

```
SQL> SELECT TO_NUMBER(cust_no_seq.nextval||SYS_CONTEXT('USERENV', 'INSTANCE'))  
2 FROM dual;
```

```
TO_NUMBER(CUST_NO_SEQ.NEXTVAL||SYS_CONTEXT('USERENV','INSTANCE'))
```

11

SQL>

Instead of using the generated number by the sequence alone for the primary key values, when combined with SYS_CONTEXT('USERENV', 'DB_NAME'), it generates a unique alphanumeric value of a character data type primary key.

Multi-Master Numbers Distribution

When Oracle GoldenGate is configured to support 3 systems for multi-master data replication, the first system sequence number begins at 1, the second system sequence number begins at 2, and the third system sequence number begins at 3. All sequences increment by 3 as shown in Figure 5-19.

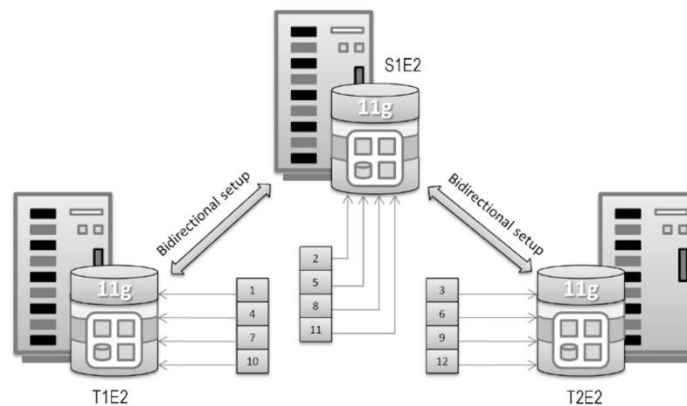


Figure 5-19: Multi-Master Numbers Sequencing

On node-1 (T1E2), create the sequence to begin at 1.

```
SQL> CREATE SEQUENCE cust_no_seq
2 START WITH 1
3 INCREMENT BY 3;
```

Sequence created.

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
NEXTVAL
-----
1
```

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
NEXTVAL
-----
```

SQL>

On node-2 (S1E2), create the sequence to begin at 2.

```
SQL> CREATE SEQUENCE cust_no_seq
      2 START WITH 2
      3 INCREMENT BY 3;
```

Sequence created.

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL
-----
           2
```

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL
-----
           5
```

SQL>

On node-3 (T2E2), create the sequence to begin at 3.

```
SQL> CREATE SEQUENCE cust_no_seq
      2 START WITH 3
      3 INCREMENT BY 3;
```

Sequence created.

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL
-----
           3
```

```
SQL> SELECT cust_no_seq.nextval FROM dual;
```

```
      NEXTVAL
-----
           6
```

SQL>

Database sequences created to avoid SQL insert conflicts are not to be included on the TABLE and MAP statements. To avoid encountering gaps due to database restart, use the NOCACHE attribute when creating sequences. Since the sequence START WITH and INCREMENT BY attributes are different on both systems, ascertain that the Oracle GoldenGate DDL replication is disabled when creating sequences and manually create them on both systems.

Data Conflicts

Without explicit conflict detection, applications will continue processing transactions of inconsistent data. Logical data conflict and inconsistencies should be handled using either Oracle GoldenGate, a built-in, or custom developed conflict detection and resolution techniques. In addition, virtual private database implementation minimizes data conflict as explained in this chapter. The demonstrated conflict resolution techniques are developed for an online application system for a two-node configuration architecture, synchronized using Oracle GoldenGate. The primary database (node-1) is located at Boston, MA and the target database (node-2) is located at Albany, NY as shown in Figure 5-20.

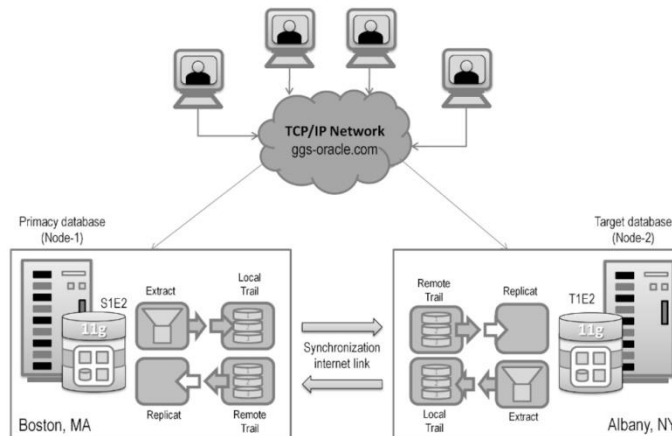


Figure 5-20: Bi-Directional Replication for Online Applications

Let us start by explaining the usages and applications of SQLEXEC parameter. This parameter is used in conjunction with custom data conflict handling technique developed in PL/SQL.

SQLEXEC Parameter

The use of the SQLEXC parameter extends Oracle GoldenGate capabilities, it enables invoking PL/SQL procedures and executes queries from within Oracle GoldenGate Extract or Replicat to support complex data conflict resolutions. The vast majority of data conflicts are handled using queries only, while procedures are mostly applied to more complex data conflicts and transformations.

Data manipulation and data conflict resolution are developed by utilizing Oracle GoldenGate built-in functions. The use of the functions @IF, @STREQ, @COMPUTE, @GETVAL and @GETENV is common. The following illustration

of using SQLEXEC is developed to support a de-normalized POLICIES table which includes a PT_DESC column.

```
SQL> DESC policies
```

Name	Null?	Type
POL_NO	NOT NULL	NUMBER
POL_FROM	NOT NULL	DATE
POL_TO	NOT NULL	DATE
POL_VALUE	NOT NULL	NUMBER(8,2)
POL_SUB_TOTAL	NOT NULL	NUMBER(8,2)
PT_CODE	NOT NULL	VARCHAR2(8)
DT_CODE		VARCHAR2(8)
POL_TOTAL	NOT NULL	NUMBER(8,2)
CUST_NO	NOT NULL	NUMBER
PT_DESC		VARCHAR2(128)

```
SQL>
```

The PT_DESC (Policy Type Description) column exists on the target database only. Hence, a lookup operation within the Replicat is necessary to complete the transaction successfully.

SQLEXEC with Queries

The components of SQLEXEC with a query specifies the query ID, which is used by the built-in function @GETVAL to assign the value to the target column. The PARAMS clause lists the pair of parameter name and value used in the query. The last clause is the COLMAP to perform column mapping which references the query columns list using the query ID.

The Replicat MAP statement uses SQLEXEC to fetch the value for the column PT_DESC using PT_CODE as a parameter.

```
MAP osm$repapi.policies, TARGET osm$repapi.policies, &
  SQLEXEC (ID lookup_pt, &
    QUERY " SELECT pt_desc
            FROM   osm$repapi.policy_types
            WHERE  pt_code = :pt_code_param ", &
    PARAMS (pt_code_param = pt_code)), &
  COLMAP (USEDEFAULTS, pt_desc = @GETVAL(lookup_pt.pt_desc));
```

The Replicat (RS1E203) parameter file using SQLEXEC with query clause.

```
REPLICAT rs1e203
SETENV (ORACLE_SID=T1E2)
SETENV (NSL_LANG=AMERICAN AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rs1e203.dsc, PURGE
MAP osm$repapi.customers, TARGET osm$repapi.customers;
```

```

MAP osm$repapi.policies, TARGET osm$repapi.policies, &
  SQLEXEC (ID lookup_pt, &
    QUERY " SELECT pt_desc
            FROM osm$repapi.policy_types
            WHERE pt code = :pt code param ", &
    PARAMS (pt_code_param = pt_code)), &
  COLMAP (USEDEFAULTS, pt_desc = @GETVAL(lookup_pt.pt_desc));

```

SQLEXEC with Stored Procedures

SQLEXEC invokes stored procedure to perform advanced data transformations and conflicts resolutions. The simplest form of using SQLEXEC with stored procedure is performing a table look operation. The below PL/SQL database procedure performs a table look up operation instead of using query.

```

SQL> CREATE OR REPLACE PROCEDURE lookup_pt(pt_code_param IN VARCHAR2,
2                                     pt_desc_param OUT VARCHAR2) IS
3 BEGIN
4     SELECT pt_desc into pt_desc_param
5     FROM osm$repapi.policy_types
6     WHERE pt_code = pt_code_param;
7 END;
8 /

```

Procedure created.

```
SQL> GRANT EXECUTE on lookup_pt to ggs_admin;
```

Grant succeeded.

```
SQL>
```

The Replicat parameter file is modified to reference the stored procedure using the SQLEXEC parameter.

```

REPLICAT rs1e203
SETENV (ORACLE_SID=T1E2)
SETENV (NSL_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rs1e203.dsc, PURGE
MAP osm$repapi.customers, TARGET osm$repapi.customers;
MAP osm$repapi.policies, TARGET osm$repapi.policies, &
  SQLEXEC (SPNAME osm$repapi.lookup_pt, &
    ID lookup_pt, &
    PARAMS (pt code param = pt code)), &
  COLMAP (USEDEFAULTS, &
    pt_desc = @GETVAL(lookup_pt.pt_desc_param));

```

The above SQLEXEC invokes a stored procedure by specifying the following clauses:

SPNAME is the name of the stored procedure. Ascertain granting Oracle GoldenGate database user to EXECUTE privilege on the called procedure.

ID is the logical name for the procedure and is needed by the @GETVAL function to retrieve the look up value.

PARAMS is the list of input parameters and their initial values.

COLMAP performs column mapping which references the output parameters using the logical name.

SQLEXEC as Stand-Alone Parameter

The third form of SQLEXEC is a stand-alone. This is a one-time call-out to execute query, DDL command, or stored procedure. This type of SQLEXEC is valid for Extract and Replicat parameter files.

The next parameter files ES1E203.PRM use SQLEXEC parameters. The first parameter executes a query to avoid IDLE_TIME enforced by the user profile.

The second parameter executes the DDL command TRUNCATE. The third parameter executes a stored procedure. Notice the procedure must not include input or output formal parameters.

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME=/d01/app/oracleSE/product/11.2.0/dbhome_1)
SETENV (NLS_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs admin, PASSWORD oracle
EXTTRAIL ./dirdat/sc
TRANLOGOPTIONS EXCLUDEUSER ggs_admin
SQLEXEC "SELECT * FROM DUAL" EVERY 55 MINUTES
SQLEXEC "TRUNCATE TABLE osm$repapi.logging" EVERY 1 DAYS
SQLEXEC "CALL send_notify()" EVERY 10 MINUTES
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
```

Custom Data Conflicts Handling

Consider an online application for managing a single inventory site. The inventory database is replicated across two cities – Boston, MA and Albany, NY as depicted in Figure 5-20. Customers' requests are directed to either database based on workload determined by the load balancer. Hence, data conflict is unavoidable. A classical data conflict may occur due to updates occurring at the time. When a conflict occurs, several resolutions are considered and should be governed by business rules.

- Use the net difference instead of using the after image value
- Perform comparison on other columns to accept, reject, or resolve transaction modifications
- Use the timestamp (date and time) then decide to accept or to reject the transaction modifications
- Manually handle the conflict by mapping data to an exception table

Conflicts Resolution Using SQLEXEC

When simultaneous bi-directional transactions update the same non-key row value on node-1 and on node-2, data divergence occurs slightly. This is a form of data conflict and must be detected and resolved.

Data Divergence Scenario

The customer has sent a change of email address request (non-key column). The request is received and routed to an agent at Boston. Shortly thereafter, the same customer sends another change of email address request with a different email from the previous request. The request is received and routed to an agent at Albany. Here is an illustration of processing this request without data conflict handling in place.

Node-1 Database: Boston	Node-2 Database: Albany
<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200;</pre> CUST_EMAIL ----- fawzi@yahoo.com SQL>	<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200;</pre> CUST_EMAIL ----- fawzi@yahoo.com SQL>
Data pump PS1E203 is down due to network failure	Data pump PT1E203 is down due to network failure
<pre>SQL> UPDATE customers SET 2 cust_email='fawzi@gmail.com' 3 WHERE cust_no=100200;</pre> 1 row updated. SQL> COMMIT; Commit complete. SQL>	<pre>SQL> UPDATE customers SET 2 cust_email='fawzi@mail.com' 3 WHERE cust_no=100200;</pre> 1 row updated. SQL> COMMIT; Commit complete. SQL>
<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200;</pre> CUST_EMAIL	<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200;</pre> CUST_EMAIL

----- fawzi@gmail.com SQL>	----- fawzi@mail.com SQL>
Network restored. The data pump PS1E203 started automatically.	Network restored. The data pump PT1E203 started automatically.
SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200; CUST_EMAIL ----- fawzi@mail.com SQL>	SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=100200; CUST_EMAIL ----- fawzi@gmail.com SQL>

There is no error, but notice that the change of address resulted in mismatched CUST_EMAIL values between the two databases. This is a simple data conflict, yet critical to be addressed to avoid a data integrity problem.

Data Divergence Handling

Oracle GoldenGate captures before and after image of updates to support custom and built-in conflict resolutions by performing a compare operation before applying the transactions on the target database. Data divergence conflicts are prevented. Figure 5-21 shows the workflow of updating the email for CUST_NO=100200. Since there is no data segmentation at the application level, a custom data conflict is developed.

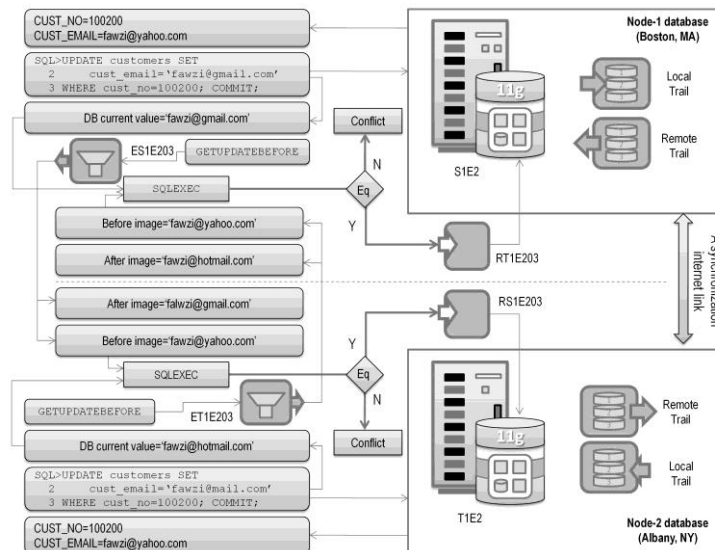


Figure 5-21: Resolving Conflict Using SQLEXEC

The Replicat applies the transaction to the database and the conflict occurs silently. The conflict is resolved either using the built-in conflict detection and resolution (CDR) or customer resolution using the SQLEXEC parameter. Using either technique, the database current value is compared with before image value from the trail record. If the two values are equal, then the transaction is applied. Otherwise, a data conflict is detected and needs to be handled accordingly.

Using the parameter SQLEXEC to handle the above data conflict requires using a query or a procedure to fetch the current database value and compare with the before image from the trail record. If the two values are identical, then apply the transaction. Otherwise report or handle data conflict. The stored procedure GET_EMAIL for handling this type of update conflict follows.

```
SQL> CREATE OR REPLACE PROCEDURE get_email(pcust_no    IN NUMBER,
2                                     pcust_email OUT VARCHAR2) IS
3 BEGIN
4     SELECT cust_email into pcust_email
5     FROM   osm$repapi.customers
6     WHERE  cust_no = pcust_no;
7 END;
8 /
```

Procedure created.

SQL>

The Extract ES1E203 uses the parameter GETUPDATEBEFORES to force capture the transaction post values and before image values for use by the Replicat to detect conflict. Also, to support operation granularity, the parameter GETBEFORECOLS is used in conjunction with GETUPDATEBEFORES to associate the before image with the operation type of UPDATE, DELETE, or both. The argument ALL specifies the before image value is for all columns. The example below illustrates using GETBEFORECOLS to capture the before image values for update and delete operations on the CUST_EMAIL column only. The argument KEYINCLUDING includes capture of the primary key before image value.

```
TABLE osm$repapi.customers &
      GETBEFORECOLS(ON UPDATE KEYINCLUDING (cust_email), &
                    ON DELETE KEYINCLUDING (cust_email));
```

The parameter GETUPDATEBEFORES remains in effect for all subsequent listed tables until the parameter IGNOREUPDATEBEFORES is encountered – enabling control of the before image capture for a specific list of tables.

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME=/d01/app/oracleSE/product/11.2.0/dbhome_1)
SETENV (NLS_LANG=AMERICAN AMERICA.AL32UTF8)
```

```

USERID ggs admin, PASSWORD oracle
EXTTRAIL ./dirdat/sc
TRANLOGOPTIONS EXCLUDEUSER ggs_admin
GETUPDATEBEFORES
TABLE osm$repapi.customers GETBEFORECOLS(ON UPDATE ALL);
IGNOREUPDATEBEFORES
TABLE osm$repapi.policies;

```

The Replicat RS1E203 parameter file includes the SOURCEDEFS parameter for mapping non-identical tables structure. Using SQLEXEC to invoke the procedure GET_EMAIL returns the current database value for compression with the before image value. The COLMAP clause includes reference to the Oracle GoldenGate built-in function to detect conflict. If the built-in function @STREQ returns 1 (true) then no conflict was detected. Otherwise, a conflict has been detected and the column CUST_EMAIL assigned the value 'conflict@pt.com' as a means of notification and intervention.

```

cust_email      = @IF(@STREQ(BEFORE.cust_email,
                             @GETVAL(get_email.pcust_email)),
                             cust_email, 'conflict@pt.com'),

```

```

REPLICAT rs1e203
SETENV (ORACLE_SID=T1E2)
SETENV (NSL_LANG=AMERICAN AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
SOURCEDEFS /d01/oracle/ggs/dirdef/defgen.def
--ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rs1e203.dsc, PURGE
GETUPDATES
GETINSERTS
GETDELETES
MAP osm$repapi.customers, TARGET osm$repapi.customers, &
  SQLEXEC (ID get_email, &
    SPNAME osm$repapi.get_email, &
    PARAMS (pcust_no = cust_no)), &
  COLMAP (cust_no      = cust_no,
    cust_name      = cust_name,
    cust_email     = @IF(@STREQ(BEFORE.cust_email,
                                @GETVAL(get_email.pcust_email)),
                                cust_email, 'conflict@pt.com'),
    cust_mobile    = cust_mobile,
    cust_address   = cust_address,
    cust_commit_ts = @GETENV('GGHEADER', 'COMMITTIMESTAMP'));
--
MAP osm$repapi.policies, TARGET osm$repapi.policies, &
  SQLEXEC (SPNAME osm$repapi.lookup_pt, &
    ID lookup_pt, &
    PARAMS (pt_code_param = pt_code)), &
  COLMAP (USEDEFAULTS, &
    pt_desc = @GETVAL(lookup_pt.pt_desc_param));

```

The following scenario illustrates conflict detection for an update operation. The Replicat applying the transaction detects the conflict and assigns the

CUST_EMAIL column with temporary email 'conflict@pt.com' for further action.

Node-1 Database: Boston	Node-2 Database: Albany
<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST_EMAIL ----- deema@yahoo.com SQL></pre>	<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST_EMAIL ----- deema@yahoo.com SQL></pre>
Data pump PS1E203 is down due to network failure	Data pump PT1E203 is down due to network failure
<pre>SQL> UPDATE customers SET 2 cust_email='deema@gmail.com' 3 WHERE cust_no=4846; 1 row updated. SQL> COMMIT; Commit complete. SQL></pre>	<pre>SQL> UPDATE customers SET 2 cust_email='deema@hotmail' 3 WHERE cust_no=4846; 1 row updated. SQL> COMMIT; Commit complete. SQL></pre>
<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST EMAIL ----- deema@gmail.com SQL></pre>	<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST EMAIL ----- deema@hotmail.com SQL></pre>
Network restored. The data pump PS1E203 started automatically.	Network restored. The data pump PT1E203 started automatically.
<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST_EMAIL ----- deema@hotmail SQL></pre>	<pre>SQL> SELECT cust_email 2 FROM customers 3 WHERE cust_no=4846; CUST_EMAIL ----- conflict@pt.com SQL></pre>

Refer to Chapter 11 to learn more regarding generating the definition file and advanced transformation.

Applying Net Difference Resolution Using SQLEXEC

When the application manipulates a single entity such as inventory level, bank account, order entry, etc., in an active-active bi-directional Oracle GoldenGate topology, conflicts may occur and must be handled. Figure 5-22 illustrates an ATM workflow for simultaneous transactions executed at the same time.

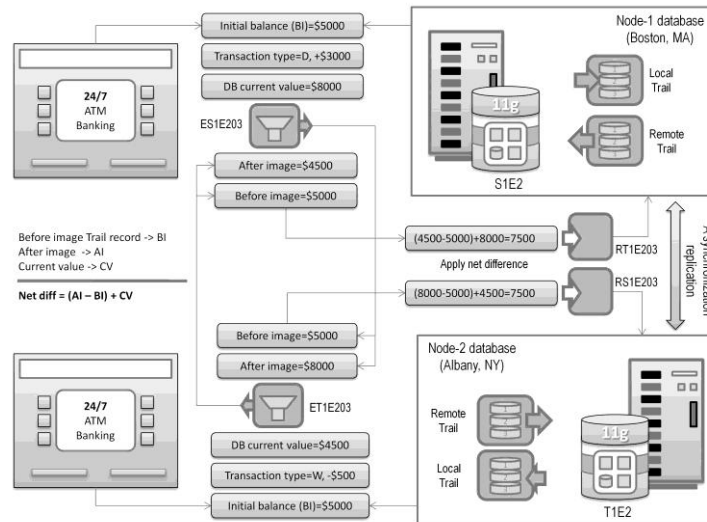


Figure 5-22: Applying Net Difference Resolution Using SQLEXEC

The initial account balance is \$5,000. A deposit transaction (D) for the amount of \$3,000 is entered using the primary ATM card. This transaction is located at Boston, MA increasing the account balance to \$8,000.

Almost simultaneously, while the initial balance is still \$5000, a withdrawal transaction (W) for the amount of \$500 is entered using the secondary ATM card. This transaction is located at Albany, NY decreasing the account balance to \$4,500.

When the Replicat applies the transaction on either database, conflict occurs. The database located at Boston, MA has a current value of \$8,000 and the database located at Albany, NY has a current value of \$4,500. By comparing the trail record and the current database values, the data conflict is detected and needs to be resolved — using either SQLEXEC for applying the net difference or Oracle GoldenGate built-in conflict detection and resolution (CDR) to calculate and apply the transaction correctly.

The resolution below is developed using SQLEXEC. The Replicat COLMAP clause determines and applies the net difference of: $\$5000 + \$3000 - \$500 = \7500 .

The net balance applied at Boston, MA is calculated as:

$$\begin{aligned}\text{Net balance} &= ((\text{Trail after image} - \text{Trail before image}) + \text{current value}) \\ &= ((4500 - 5000) + 8000) = \$7,500\end{aligned}$$

The net balance applied at Albany, NY is calculated as:

$$\begin{aligned}\text{Net balance} &= ((\text{Trail after image} - \text{Trail before image}) + \text{current value}) \\ &= ((8000 - 5000) + 4500) = \$7,500\end{aligned}$$

To demonstrate data conflict by applying the net difference value, the table CURRENT_ACCOUNTS exists on node-1 database (S1E2) and node-2 database (T1E2). It is configured on bi-directional configuration with transactions entered from both nodes.

```
SQL> DESC current_accounts
Name                                     Null?      Type
-----
ACCT_NO                                NOT NULL   NUMBER
ACCT_NAME                              NOT NULL   VARCHAR2(64)
ACCT_AMOUNT                            NUMBER
ACCT_TS                                TIMESTAMP(6)

SQL>
SQL> SELECT ACCT_AMOUNT
       2 FROM   current_accounts
       3 WHERE  acct_no=49664996;

ACCT_AMOUNT
-----
          5000

SQL>
```

The Extract parameter file lists the tables. The parameters GETUPDATEBEFORES and GETBEFORECOLS enable capture of the before image values.

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME=/d01/app/oracleSE/product/11.2.0/dbhome_1)
SETENV (NLS_LANG=AMERICAN AMERICA.AL32UTF8)
USERID ggs admin, PASSWORD oracle
EXTTRAIL ./dirdat/sc
TRANLOGOPTIONS EXCLUDEUSER ggs_admin
GETUPDATEBEFORES
TABLE osm$repapi.customers &
      GETBEFORECOLS(ON UPDATE KEYINCLUDING (cust_email), &
                    ON DELETE KEYINCLUDING (cust_email));
IGNOREUPDATEBEFORES
TABLE osm$repapi.policies;
```

```

GETUPDATEBEFORES
TABLE osm$repapi.current_accounts &
    GETBEFORECOLS(ON UPDATE ALL);
IGNOREUPDATEBEFORES

```

The Replicat parameter file uses SQLEXEC to execute a query then compares the before image with the query return value. The COLMAP clause uses Oracle GoldenGate built-in @COMPUTE function to determine the net difference.

```

REPLICAT rsle203
SETENV (ORACLE_SID=T1E2)
SETENV (NSL_LANG=AMERICAN AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
SOURCEDEFS /d01/oracle/ggs/dirdef/defgen.def
DISCARDFILE ./dirrpt/rsle203.dsc, PURGE
GETUPDATES
GETINSERTS
GETDELETES
MAP osm$repapi.customers, TARGET osm$repapi.customers, &
    SQLEXEC (ID      get_email, &
             SPNAME osm$repapi.get_email, &
             PARAMS (pcust_no = cust_no)), &
    COLMAP (cust_no      = cust_no,
            cust_name     = cust_name,
            cust_email    = @IF(@STREQ(BEFORE.cust_email,
                                     @GETVAL(get_email.pcust_email)), cust_email,
                                'conflict@pt.com'),
            cust_mobile   = cust_mobile,
            cust_address  = cust_address,
            cust_commit_ts = @GETENV('GGHEADER', 'COMMITTIMESTAMP'));
--
MAP osm$repapi.policies, TARGET osm$repapi.policies, &
    SQLEXEC (SPNAME osm$repapi.lookup_pt, &
             ID      lookup_pt, &
             PARAMS (pt_code_param = pt_code)), &
    COLMAP (USEDEFAULTS, &
            pt_desc = @GETVAL(lookup_pt.pt_desc_param));
--
MAP osm$repapi.current_accounts, TARGET osm$repapi.current_accounts,
    SQLEXEC (ID lookup_acct_amount, &
             QUERY " SELECT acct_amount FROM osm$repapi.current_accounts
                     WHERE acct_no = :vacct_no ", &
             PARAMS (vacct_no = acct_no)), &
    COLMAP (acct_no      = acct_no,
            acct_name     = acct_name,
            acct_amount   = @COMPUTE(((acct_amount - BEFORE.acct_amount) +
                                     @GETVAL(lookup_acct_amount.acct_amount))),
            acct_ts       = @GETENV('GGHEADER', 'COMMITTIMESTAMP'));

```

The below sequence of query and update statements demonstrates the same ACCT_NO is updated when the data pump is ABENDED due to network failure, which should lead to data conflict when the data pump is re-started. However, by applying the net difference value, the ACCT_AMOUNT value column is correctly calculated and converged.

Node-1 database, Boston	Node-2 database, Albany
<pre>SQL> SELECT ACCT AMOUNT 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 5000 SQL></pre>	<pre>SQL> SELECT ACCT AMOUNT 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 5000 SQL></pre>
Data pump PS1E203 is down due to network failure	Data pump PT1E203 is down due to network failure
<pre>SQL> UPDATE current_accounts SET 2 acct_amount=acct_amount+3000 3 WHERE acct_no=49664996; 1 row updated. SQL> COMMIT;</pre>	<pre>SQL> UPDATE current_accounts SET 2 acct_amount=acct_amount-500 3 WHERE acct_no=49664996; 1 row updated. SQL> COMMIT;</pre>
<pre>SQL> SELECT acct_amount 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 8000</pre>	<pre>SQL> SELECT acct_amount 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 4500</pre>
Network restored. The data pump PS1E203 started automatically.	Network restored. The data pump PT1E203 started automatically.
<pre>SQL> SELECT acct_amount 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 7500 SQL></pre>	<pre>SQL> SELECT acct_amount 2 FROM current_accounts 3 WHERE acct no=49664996; ACCT_AMOUNT ----- 7500 SQL></pre>

DML Conflict Types

Fundamentally, conflicts occur and must be handled. Why? It is because Oracle GoldenGate Replicat preserves data integrity when applying transactions. It does so by maintaining the transactions commit order and uses the target database native calls when applying transactions, which have database constraints enabled.

An active-active bi-directional configuration must implement, detect, and resolve data conflict techniques. These conflicts arising from DML transactions manipulate the same row stored on the source and target databases. The resolution is detected and resolved using either built-in or custom developed conflict detection and resolution techniques.

The resolution is developed as part of the transaction integrity or by applying basic resolution rules such as timestamps. There are (3) types of conflicts that occur when executing SQL statements on asynchronous mode:

Insert Conflicts

Due to enabled and valid primary key or unique key constraints on tables, insert conflicts occur when duplicate values violate the constraint. Without conflict handling the Replicat terminates with an error.

Update Conflicts

Modifying the same row at the same time results in update conflicts. However, because access to the updated row's before and after image is available, conflicts are resolved by applying suitable resolution methods.

Delete Conflicts

This type of conflict commonly occurs when the Replicat attempts to apply a delete transaction, but the row is not found on the target database. This is because the row is already deleted or updated by another transaction.

Data Conflict and Resolution Roadmap

The fundamental rule when developing conflict resolution is to consider the technique consistency. This allows applying the same technique across the vast majority of the database tables. There are several approaches to data conflict avoidance, detection, and resolution. The combination of avoidance and resolution promotes data convergence and data consistency. Oracle GoldenGate provides its own built-in function and conflict detection and resolution (CDR). It is the distinctive feature of Oracle GoldenGate 11g Release 2, explained and demonstrated shortly.

The techniques and order for handling data conflicts in an active-active bi-directional topology is shown in Figure 5-23. Since Oracle GoldenGate is mostly deployed after the application, it is less likely the application is conflict-aware. To minimize the application changes and make Oracle GoldenGate data replication transparent on bi-directional topology, the roadmap begins by implementing data segmentation as an avoidance technique. When data segmentation may not totally avoid data conflict, partitioning the applications provides the second line of defense.

In addition to data segmentation and application partitioning, Oracle GoldenGate

built-in conflict detection and resolution applies consistent methods to resolve conflicts using declarative parameter settings for all types of DML conflict types.

For complex data conflict detection and resolution, using the SQLEXEC parameter requires developing PL/SQL sub-programs and SQL queries that are invoked from within Oracle GoldenGate.

When SQLEXEC is not meeting the requirements, user exits is another option to consider for handling sophisticated conflicts beyond those provided by Oracle GoldenGate built-in functions and techniques. User exits are developed in C and C++ programming language and invoked from the Extract and Replicat, but mostly adapted for advanced integration and data transformation.

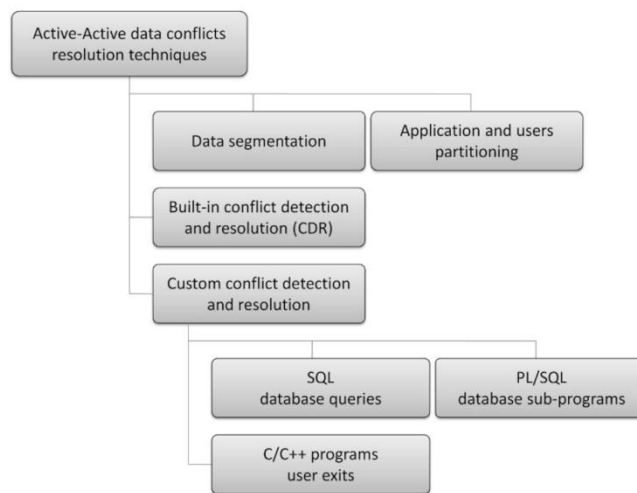


Figure 5-23: Conflicts Resolutions Roadmap

Data Segmentation

When the application is not replication-aware, then implementing data segmentation is the first approach of the data conflict avoidance technique. It is developed and deployed within the database only. An example is implementing Oracle Virtual Private Database (VPD) to dynamically apply a table-level security policy function to avoid the likelihood of having the same row concurrently accessed by multiple users.

Application Segmentation

Application segmentation is implemented from within the virtual private database – making the application replication-aware. Then by segregating applications so that

groups of users are associated with an application, accessing a subset of tables reduces the possibility of data conflicts. Application segmentation is developed using application contexts such as locations, transaction type, etc. This is an avoidance technique to minimize and not to entirely eliminate data conflicts.

Built-In Conflict Detection and Resolution

Oracle GoldenGate 11g Release 2 is shipped with built-in conflict detection and resolution (CDR). This is a core feature of any data replication software. Using simple declarative conflicts and methods, Oracle GoldenGate handles the majority of conflicts for database applications accessing active-active bi-directional topology.

Custom Conflict Resolution

Using queries, PL/SQL sub-programs, and user exits enable developing conflict resolution beyond the capabilities of Oracle GoldenGate built-in conflict resolution methods. The technique extends the capabilities of Oracle GoldenGate to include using the database built-in features for events handling.

Built-In Conflict Detection and Resolution

While complex business logic custom data conflicts resolutions are developed using PL/SQL, SQL queries, and C/C++ user exits, the majority of data conflicts can be handled by adapting Oracle GoldenGate automatic conflicts detection and resolution (CDR) techniques.

The advantages of employing the built-in CDR are performance, consistency, and lower maintenance cost. Resolving conflicts using CDR delivers better performance when compared to custom resolution using SQLEXEC to invoke database PL/SQL procedures or SQL executing queries. The built-in resolution and detection avoid runtime context switching. Since the vast majority of tables are handled using similar conflict resolution methods, CDR delivers consistency using declarative conflict types and methods. Lower maintenance cost is also an advantage of using CDR. For example, when modifying CDR methods, deploying CDR new features, adding new table columns and database tables, maintenance is executed within GoldenGate only. Oracle GoldenGate built-in conflict detection and resolution uses the four parameters listed in Table 5-4, promoting ease of configuration and deployment.

Parameter	Description
GETUPDATEBEFORES	An Extract parameter. It forces capture of the transaction post values (after image) and before image values for use by the

GETBEFORECOLS	Replicat to detect conflicts. An Extract parameter. It associates the before image with the operation type of UPDATE, DELETE, or both and the column list.
COMPARECOLS	A Replicat parameter. It lists the columns to check for conflicts with the target database current value. A mismatch is a conflict and handled by the RESOLVECONFLICT settings.
RESOLVECONFLICT	A Replicat parameter. It specifies the operation type, the resolution type, and the method applied to resolve the conflict. The parameter is repeated for each resolution to handle the different operations.

Table 5-4: Built-In Conflict Detection and Resolution Parameters

For tables referenced by transactions on an active-active bi-directional topology, the parameter RESOLVECONFLICT is configured to handle conflicts. Within the same MAP statement, it resolves all types of data conflict transactions for UPDATE, DELETE, and INSERT SQL statements. The below illustrates resolving update conflicts. The named resolution DELTA_RESOLUTION_METHOD uses the method USEDELTA for the column ACCT_AMOUNT. The resolution MAX_RESOLUTION_METHOD uses the method USEMAX for the columns ACCT_NAME and ACCT_TS.

```
RESOLVECONFLICT (UPDATEROWEXISTS,
                  (DELTA_RESOLUTION_METHOD, USEDELTA, COLS (acct_amount)),
                  (MAX_RESOLUTION_METHOD, USEMAX (acct_ts),
                   COLS (acct_name, acct_ts))
```

Figure 5-24 depicts the hierarchy of using Oracle GoldenGate built-in conflict detection and resolution (CDR).

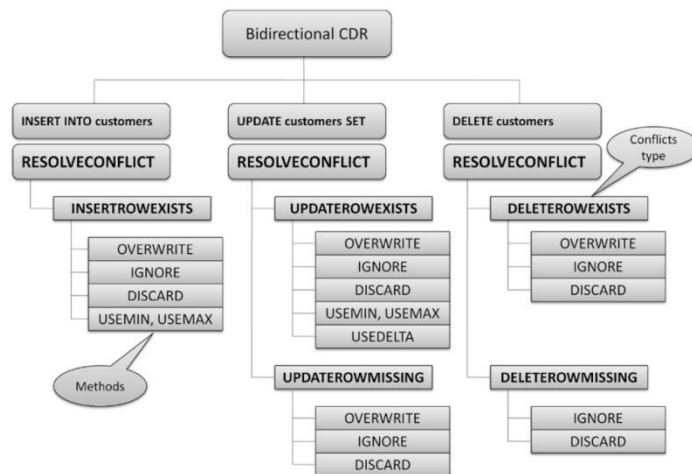


Figure 5-24: Built-In Conflicts Detection and Resolution

The parameter **RESOLVECONFLICT** takes any of the arguments listed in Table 5-5. Depending on the business rules for data conflict resolution handling, associate the argument with a resolution method.

Conflict type	Description
INSERTROWEXISTS	Handles unique constraint violation. The target database has primary key or unique key constraint enabled.
UPDATEROWEXISTS	Handles update data conflict when the before image value for one or more columns differ from the current database value.
UPDATEROWMISSING	Handles data conflict when the row does not exist on the target database.
DELETEROWEXISTS	Handles delete data conflict when before image value for one or more columns differ from the current database value.
DELETEROWMISSING	Handles data conflict when the row deleted on source does not exist on the target database.

Table 5-5: RESOVLECONFLICT Parameter Conflict Types

Each conflict type is associated with one resolution method. The following is the list of methods applied to a **DEFAULT** or named resolution:

- **OVERWRITE**
- **IGNORE**
- **DISCARD**
- **USEMIN**
- **USEMAX**
- **USEDELTA**

Figure 5-25 outlines the syntax of the Replicat built-in conflict detection and resolution. The combination of **COMPARECOLS** and **RESOLVECONFLICT** parameters handles the operation types and the conflict types. For each conflict type, a resolution method is selected and may be applied to one or more columns.

```
MAP <source table>, TARGET <target table>,  
COMPARECOLS (  
  { ON UPDATE {ALL | KEY | KEYINCLUDING(col1, col2,..., coln) |  
    KEYEXCULding(col1, col2,..., coln)} |  
    ,ON DELETE {ALL | KEY | KEYINCLUDING(col1, col2,..., coln) |  
      KEYEXCULding(col1, col2,..., coln)} } )  
RESOLVECONFLICT (  
  { INSERTROWEXISTS |  
    UPDATEROWEXISTS | UPDATEROWMISSING |  
    DELETEROWEXISTS | DELETEROWMISSING }  
  (  
    { DEFAULT | <resolution name>},  
    { USEMAX (column) |  
      USEMIN (column) |  
      DELTA |  
      DISCARD |  
      OVERWRITE }  
  ),  
  COLS (col1, col2,..., coln)  
)
```

Figure 5-25: Built-In Conflict Detection and Resolution Syntax

Configuring Built-In Conflict Detection and Resolution

This section implements active-active configuration using Oracle GoldenGate built-in conflict detection and resolution (CDR). Consider the CURRENT_ACCOUNTS table, the Extract and Replicat are configured for CDR. The ACCT_AMOUNT column is handled using the DELTA method. The ACCT_NAME and ACCT_TS columns are handled using the USEMAX method.

```
SQL> DESC current_accounts  
Name Null? Type  
-----  
ACCT_NO NOT NULL NUMBER  
ACCT_NAME NOT NULL VARCHAR2(64)  
ACCT_AMOUNT NUMBER  
ACCT_TS TIMESTAMP(6)  
  
SQL>
```

As part of the change data capture (CDC), ensure to run the ADD TRANDATA command on node-1 and node-2 databases using the syntax below.

```
GGSCI (apps-srv1) 4> ADD TRANDATA osm$repapi.current_accounts ALLCOLS
```

Node-1 Extract and Replicat Parameter Files

The parameter files for node-1 configured to support the Extract ES1E203, the data pump PS1E203 and the Replicat RT1E203.

The use of the parameters GETUPDATEBEFORES, GETBEFORECOLS and IGNOREUPDATEBEFORES is for supporting capture of the before image on the trail records.

```
EXTRACT es1e203
SETENV (ORACLE_SID=S1E2)
SETENV (ORACLE_HOME=/d01/app/oracleSE/product/11.2.0/dbhome_1)
SETENV (NLS_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
EXTTRAIL ./dirdat/sc
TRANLOGOPTIONS EXCLUDEUSER ggs_admin
GETUPDATEBEFORES
TABLE osm$repapi.customers &
    GETBEFORECOLS(ON UPDATE KEYINCLUDING (cust_email), &
                  ON DELETE KEYINCLUDING (cust_email));
IGNOREUPDATEBEFORES
TABLE osm$repapi.policies;
GETUPDATEBEFORES
TABLE osm$repapi.current_accounts &
    GETBEFORECOLS(ON UPDATE ALL, ON DELETE ALL);
IGNOREUPDATEBEFORES
```

The data pump PS1E203 is configured on PASSTHRU mode, sending data to node-2 without transformation.

```
EXTRACT ps1e203
USERID ggs_admin, PASSWORD oracle
PASSTHRU
RMTHOST apps-srv2, MGRPORT 7806
RMTTRAIL ./dirdat/tc
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
TABLE osm$repapi.current_accounts;
```

The Replicat RT1E203 applies transactions to the node-1 database.

```
REPLICAT rt1e203
SETENV (ORACLE_SID=S1E2)
SETENV (NLS_LANG=AMERICAN_AMERICA.AL32UTF8)
USERID ggs admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rt1e203.dsc, PURGE
MAP osm$repapi.customers, TARGET osm$repapi.customers;
MAP osm$repapi.policies, TARGET osm$repapi.policies;
MAP osm$repapi.current_accounts, TARGET osm$repapi.current_accounts,
    COMPARECOLS (ON UPDATE ALL, ON DELETE ALL),
    RESOLVECONFLICT (UPDATEROWEXISTS,
```

```

        (DELTA_RESOLUTION_METHOD, USEDELTA, COLS (acct amount)),
        (MAX_RESOLUTION_METHOD, USEMAX(acct_ts),
        COLS (acct_name, acct_ts)),
        (DEFAULT, USEMAX(acct_ts))),
RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)),
RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)),
RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)),
RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX(acct_ts)));

```

Node-2 Extract and Replicat Parameter Files

The set of node-2 parameter files are similar to node-1. The parameter files support the Extract ET1E203, the data pump PT1E203 and the Replicat RS1E203.

The Extract ET1E203 parameter file.

```

EXTRACT et1e203
SETENV (ORACLE_SID="T1E2")
SETENV (ORACLE_HOME="/d01/app/oracleSE/product/11.2.0/dbhome_1")
SETENV (NLS_LANG="AMERICAN AMERICA.AL32UTF8")
USERID ggs admin, PASSWORD oracle
TRANLOGOPTIONS EXCLUDEUSER ggs admin
EXTTRAIL ./dirdat/sd
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
GETUPDATEBEFORES
TABLE osm$repapi.current_accounts &
    GETBEFORECOLS (ON UPDATE ALL, ON DELETE ALL);
IGNOREUPDATEBEFORES

```

The data pump PT1E203 parameter file.

```

EXTRACT pt1e203
USERID ggs admin, PASSWORD oracle
PASSTHRU
RMTHOST apps-srv1, MGRPORT 7805
RMTRAIL ./dirdat/td
TABLE osm$repapi.customers;
TABLE osm$repapi.policies;
TABLE osm$repapi.current_accounts;

```

The Replicat RS1E203 parameter file.

```

REPLICAT rs1e203
SETENV (ORACLE_SID=T1E2)
SETENV (NSL_LANG=AMERICAN AMERICA.AL32UTF8)
USERID ggs_admin, PASSWORD oracle
ASSUMETARGETDEFS
DISCARDFILE ./dirrpt/rs1e203.dsc, PURGE
GETUPDATES
GETINSERTS
GETDELETES

```

```

MAP osm$repapi.customers, TARGET osm$repapi.customers, &
  SQLEXEC (ID get_email, &
    SPNAME osm$repapi.get_email, &
    PARAMS (pcust_no = cust_no), &
    COLMAP (cust_no      = cust_no,
            cust_name    = cust_name,
            cust_email    = @IF(@STREQ(BEFORE.cust_email,
            @GETVAL(get_email.pcust_email)),
            cust_email, 'conflict@pt.com'),
            cust_mobile   = cust_mobile,
            cust_address  = cust_address,
            cust_commit_ts = @GETENV('GGHEADER',
            'COMMITTIMESTAMP')));
--
MAP osm$repapi.policies, TARGET osm$repapi.policies, &
  SQLEXEC (SPNAME osm$repapi.lookup_pt, &
    ID      lookup_pt, &
    PARAMS (pt_code_param = pt_code)), &
    COLMAP (USEDEFAULTS, &
            pt_desc = @GETVAL(lookup_pt.pt_desc_param));
--
MAP osm$repapi.current_accounts, TARGET osm$repapi.current_accounts,
  COMPARECOLS (ON UPDATE ALL, ON DELETE ALL),
  RESOLVECONFLICT (UPDATEROWEXISTS,
    (DELTA_RESOLUTION_METHOD, USEDELTA, COLS (acct_amount)),
    (MAX_RESOLUTION_METHOD, USEMAX(acct_ts),
    COLS (acct_name, acct_ts)),
    (DEFAULT, USEMAX(acct ts))),
  RESOLVECONFLICT (UPDATEROWMISSING, (DEFAULT, OVERWRITE)),
  RESOLVECONFLICT (DELETEROWEXISTS, (DEFAULT, OVERWRITE)),
  RESOLVECONFLICT (DELETEROWMISSING, (DEFAULT, DISCARD)),
  RESOLVECONFLICT (INSERTROWEXISTS, (DEFAULT, USEMAX(acct ts)));

```

Highly Available Applications

Most call center's applications are the front line for customer service companies that provide 24/7 services. Requests and inquiry calls coming from various input devices such as mobile phones, smart phone applications, and browser-based applications are system received and routed to the next available agent. Figure 5-26 shows the workflow for incoming customer calls. This is a high level description to demonstrate using Oracle Virtual Private Database (VPD) in supporting the data conflict avoidance technique for this kind of business environment.

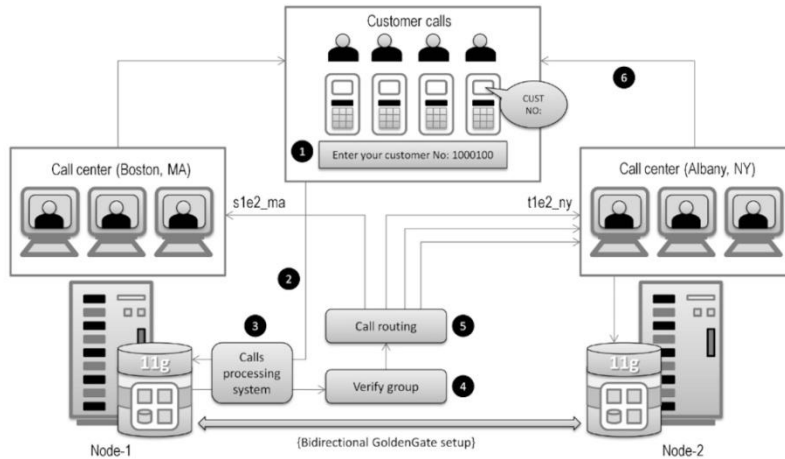


Figure 5-26: Call Center Applications Workflow

The customer calls in for inquires, such as requesting a policy renewal quotation. An automated system welcomes the customer. Then, the customer call is processed by assigning an agent to fulfill the request as follows:

1. The caller enters their customer number, which is the unique identifier.
2. The number is sent to the current database dedicated for receiving calls.
3. The call processing system validates the call and updates the calls log.
4. Based on the customer number, the owner group is determined.
5. Call routing is made based on the group name of the customer number.
6. An assigned agent receives the routed call and is locked with the customer.

The next section is developed on the basis of the above applications workflow. Data separation is achieved by routing requests and inquires to either Boston or Albany call centers. The objective is to maximize the return of investment (ROI) from licensing Oracle Database Enterprise Edition and Oracle GoldenGate software bundles for achieving a two-node architecture as an active-active topology applied for high availability and disaster recovery business requirements.

Transparent Avoidance Using Virtual Private Database

Virtual Private Database (VPD) is an advanced data conflict avoidance technique for transparent data segmentation. After creating database's table polices and implementing fine-grain access control and application contexts, SQL statements are dynamically modified to control rows (and columns) level access by database users. Figure 5-27 shows how users on node-1 and on node-2 access different subset of data, even when

the session is connected to the same database. Fine-grain access control limits rows access by implementing a database function that returns a predicate clause (WHERE) to virtually limit row access by the database users.

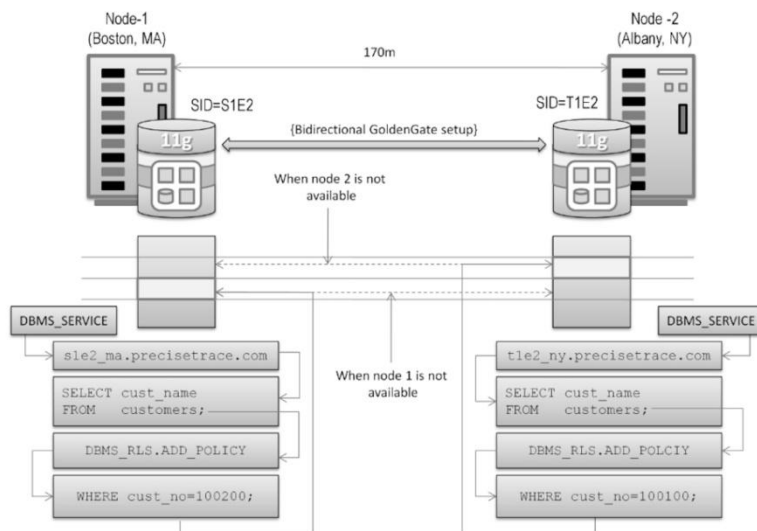


Figure 5-27: Data Segmentation Using Virtual Private Database

Our objective for using Virtual Private Database is to enforce data access based on ownership, so that a group of users within a database sees only their own accounts. The implementation is transparent – defined using policies that are independent of the applications. These active policies are cached in the SGA shared pool allowing in-memory access for faster evaluation and processing of fine-grain access control using SYS_CONTEXT function.

Begin by using DBMS_SERVICE PL/SQL supplied package to create non-default service names which will be used to associate a database user to one-and-only-one group at any time. However, in the event of a node failure, the user connects to the surviving node using the same service name but referencing the surviving node database service. Figure 5-27 shows the service name s1e2_ma for users connecting to node-1 database and t1e2_ny for users connecting to node-2 database.

The calls to DBMS_SERVICE.CREATE_SERVICE create the service names s1e2_ma and t1e2_ny, then start these service names for user connections. The procedure DBMS_SERVICE.CREATE_SERVICE requires the service name and service network name, use the service name to manage the service name.

```
SQL> BEGIN
  2   DBMS_SERVICE.CREATE_SERVICE(service_name=>'s1e2_ma',
  3                               network_name=>'s1e2_am.precisetrace.com');
  4 END;
  5 /
```

PL/SQL procedure successfully completed.

```
SQL> EXEC DBMS_SERVICE.START_SERVICE('s1e2_ma');
```

PL/SQL procedure successfully completed.

```
SQL> BEGIN
  2   DBMS_SERVICE.CREATE_SERVICE(service_name=>'t1e2_ny',
  3                               network_name=>'t1e2_ny.precisetrace.com');
  4 END;
  5 /
```

PL/SQL procedure successfully completed.

```
SQL> EXEC DBMS_SERVICE.START_SERVICE('t1e2_ny');
```

PL/SQL procedure successfully completed.

Use 'netca utility' to create the database connect string for the new service names and perform quick tests. By creating the two service names on node-1 and node-2 instances, they are readily available in the event of either node failure. The connect string for s1e2_ma is configured (t1e2_ny is not shown), followed by a successful test using the connect string s1e2_ma and t1e2_ny.

```
S1E2_MA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = ggs_source) (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = s1e2_ma.precisetrace.com)
    )
  )
```

```
SQL> CONN fawzi/oracle@s1e2_ma
Connected.
SQL> CONN deema/oracle@t1e2_ny
Connected.
SQL>
```

How Does Virtual Private Database Work?

The major advantage for using Virtual Private Database is ease of deployment and flexibility. After the policy is created, developed, and enabled, it becomes part of the database and access control applies across all applications of the table or view.

The workflow of executing an SQL application enforced by the Virtual Private Database security policy on the CUSTOMERS table is as follows:

1. The user accesses the CUSTOMERS table using SQL applications.
2. The application invokes the package for preparing and setting the context attribute values.
3. The policy function returns the predicate. It uses the session application context attributes stored in memory.
4. The database server dynamically modifies the statement by appending the returned predicate to SQL applications of step 1.
5. The session executes the dynamically generated SQL applications.

Implementing Virtual Private Database

The implementation of Virtual Private Database may require designing the database to provide the application context attribute values at runtime. Figure 5-28 shows a modified data model where customers are created by a group, meaning the customers and policies are owned and modified by a specific location (group). This model provides data conflict avoidance as the data is modified by transactions within the same database.

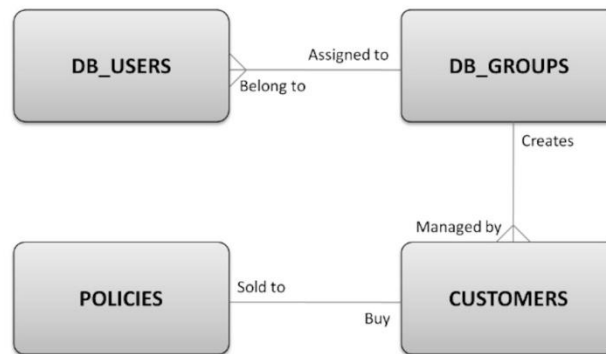


Figure 5-28: Data Modeling for VPD

The table structure of Figure 5-28 is listed below. The column GROUP_ID for each customer supports developing a conflict avoidance by group (location) and not by users. Transactions are processed as one unit of work and sent to the remote database after being processed by the local database.

```

SQL> DESC db_groups
Name                               Null?    Type
-----
GROUP_ID                           NOT NULL VARCHAR2 (30)
GROUP_DESC                         NOT NULL VARCHAR2 (128)
  
```

```

SQL> DESC db_users

```

Name	Null?	Type
USER_ID	NOT NULL	VARCHAR2 (30)
USER_NAME	NOT NULL	VARCHAR2 (128)
GROUP_ID	NOT NULL	VARCHAR2 (30)

```

SQL> DESC customers

```

Name	Null?	Type
CUST_NO	NOT NULL	NUMBER
CUST_NAME	NOT NULL	VARCHAR2 (128)
CUST_EMAIL	NOT NULL	VARCHAR2 (128)
CUST_MOBILE		VARCHAR2 (16)
CUST_ADDRESS		VARCHAR2 (128)
GROUP_ID	NOT NULL	VARCHAR2 (30)

```

SQL> DESC policies

```

Name	Null?	Type
POL_NO	NOT NULL	NUMBER
POL_FROM	NOT NULL	DATE
POL_TO	NOT NULL	DATE
POL_VALUE	NOT NULL	NUMBER (8,2)
POL_SUB_TOTAL	NOT NULL	NUMBER (8,2)
PT_CODE	NOT NULL	VARCHAR2 (8)
DT_CODE		VARCHAR2 (8)
POL_TOTAL	NOT NULL	NUMBER (8,2)
CUST_NO	NOT NULL	NUMBER

```

SQL>

```

Next, using the PL/SQL supplied package DBMS_RLS to create the CUSTOMERS table, access control policy named CUST_POLICY_VPD adapted for transactions of type SELECT, UPDATE, INSERT, and DELETE.

Creating the Application Groups and Users

The groups are created to identify users' location. Each group is associated with a specific instance service name. The below INSERT statements creates two groups. Each correspond to a service name.

```

SQL> INSERT INTO db_groups VALUES('sle2_ma', 'sle2_ma.precisetrace.com');

1 row created.

SQL> INSERT INTO db_groups VALUES('tle2_ny', 'tle2_ny.precisetrace.com');

1 row created.

SQL> COMMIT;

Commit complete.

```

```
SQL>
```

Then, the users are created using the format to associate them with the service network name for node 1 or node 2, but not both. For the database username 'fawzi', the application username is 'fawzi\$s1e2_ma.precisetrace.com.'

```
SQL> INSERT INTO db_users VALUES
      2 ('fawzi$s1e2_ma.precisetrace.com', 'F. Alswaimil', 's1e2_ma');
```

```
1 row created.
```

```
SQL> INSERT INTO db_users VALUES
      2 ('deema$t1e2_ny.precisetrace.com', 'D. Alswaimil', 't1e2_ny');
```

```
1 row created.
```

```
SQL> COMMIT;
```

```
Commit complete.
```

```
SQL>
```

Virtual Private Database Prerequisite

There are several related virtual private policy privileges that need to be granted to certain users for managing and troubleshooting objects encompassed by virtual private database policies. The following DDL statements grant capabilities to the application owner, Oracle GoldenGate database users, and SYSTEM administrative when working with tables and views enforced by Virtual Private Database.

```
SQL> CONN / as sysdba
Connected.
```

```
SQL> GRANT EXEMPT ACCESS POLICY TO SYSTEM;
```

```
Grant succeeded.
```

```
SQL> GRANT EXEMPT ACCESS POLICY TO ggs_admin;
```

```
Grant succeeded.
```

```
SQL> GRANT EXEMPT ACCESS POLICY TO osm$repapi;
```

```
Grant succeeded.
```

```
SQL> GRANT EXECUTE ON DBMS_RLS to osm$repapi;
```

```
Grant succeeded.
```

```
SQL>
```

Create the Security Policy

Use the procedure DBMS_RLS.ADD_POLICY to create the policy CUST_POLICY_VPD. This implements fine grain row level access on the CUSTOMERS table. The STATEMENT_TYPES parameter lists the SQL application types enforced by this policy. The POLICY_FUNCTION parameter specifies the function name that returns the predicate.

```
SQL> BEGIN
  2     DBMS_RLS.ADD_POLICY(
  3         object_schema=>'system',
  4         object_name=>'customers',
  5         policy_name=>'cust_policy_vpd',
  6         function_schema=>'system',
  7         policy_function=>'ggs_security_p.cust_order',
  8         statement_types=>'select, insert, update, delete');
  9 END;
 10 /
```

PL/SQL procedure successfully completed.

SQL>

Create the Context Management Package

Develop the package to prepare the application context using the function SYS_CONTEXT and set the context attributes values using the procedure DBMS_SESSION.SET_CONTEXT. For all transaction types, the package set the value for context attribute GROUP_ID according to the USER_ID.

```
SQL> CREATE OR REPLACE PACKAGE ggs_security is
  2     PROCEDURE set_user_group;
  3 END;
  4 /
```

Package created.

```
SQL> CREATE OR REPLACE PACKAGE BODY ggs_security is
  2     PROCEDURE set_user_group(puser_id varchar2) IS
  3         vgroup_id VARCHAR2(30);
  4 BEGIN
  5     SELECT group_id into vgroup_id
  6     FROM   db_users
  7     WHERE  UPPER(user_id) =
  8           UPPER(puser_id)||'$'||UPPER(SYS_CONTEXT('USERENV',
  9           'SERVICE_NAME'));
 10     DBMS_SESSION.SET_CONTEXT('oeapp', 'group_id', vgroup_id);
 11 EXCEPTION
 12     WHEN OTHERS THEN
 13         DBMS_SESSION.SET_CONTEXT('oeapp', 'group_id', NULL);
 14 END;
```

```
15 /  
  
Package body created.  
  
SQL>
```

Create the Security Function

Develop the function to return the predicate clause (WHERE). The return predicate is appended to the existing where clause is using AND as logical operator.

```
SQL> CREATE OR REPLACE PACKAGE ggs_security_p is  
2     FUNCTION cust_order(object_schema VARCHAR2, object_name VARCHAR2)  
3     RETURN VARCHAR2;  
4 END;  
5 /
```

Package created.

```
SQL> CREATE OR REPLACE PACKAGE BODY ggs_security_p is  
2     FUNCTION cust_order(object_schema VARCHAR2, object_name VARCHAR2)  
3     RETURN VARCHAR2 IS  
4     BEGIN  
5         RETURN 'group_id=SYS_CONTEXT(''oeapp'', 'group_id')';  
6     END;  
7 END;  
8 /
```

Package body created.

SQL>

Create the Database Application Context

Connect as privileged user to create the context. The context references the package created above.

```
SQL> CREATE CONTEXT oeapp USING ggs_security;
```

Context created.

SQL>

Execute the Application

Now, it is time to run the application accessing tables enforced by Virtual Private Database. Depending on the database username, the context attributes assign runtime values allowing dynamic modified SQL application to be executed.

Figure 5-29 shows the original SQL statement tunnel through the security policy CUST_POLICY_VPD, the ultimate modified SQL statement restricts access only to rows that are within the context of the username session.



Figure 5-29: Dynamic SQL Modification for Data Access Control

The application starts by invoking the package that prepares and initializes the context attributes, which are used by the security function that return the predicate clause. The application heavily relies on the built-in database function SYS_CONTEXT which uses a pre-defined list of attributes such as DB_NAME and SERVICE_NAME for application runtime.

The following shows how data segmentation is achieved. It considers the combination of the database username and the service name to produce an active row set specific to the user's session. The following demonstrates the row set when connected to the Albany, NY system.

```

SQL> CONN deema/oracle@tle2_ny
Connected.
SQL> DECLARE
  2     CURSOR c1 IS SELECT * FROM system.customers;
  3     vcust system.customers%ROWTYPE;
  4 BEGIN
  5     SYSTEM.GGS_SECURITY.SET_USER_GROUP(USER);
  6     OPEN c1;
  7     LOOP
  8         FETCH c1 into vcust;
  9         EXIT WHEN c1%NOTFOUND;
 10         DBMS_OUTPUT.PUT_LINE(vcust.cust_name);
 11     END LOOP;
 12     CLOSE c1;
 13 END;
 14 /
CAROL P. THOMPSON
MARK B. MILLER
RONALD H. ALLEN
NANCY U. WILSON
JESSICA Z. WILSON
DAVID O. JOHNSON
SANDRA J. ALSWAIMIL
SARAH N. KING
KENNETH O. ROBINSON
LINDA K. WILSON
  
```

PL/SQL procedure successfully completed.

The following demonstrates when the user connects to the Boston, MA system. It is the same application driven by dynamic context attributes and values to retrieve different active row sets.

```
SQL> CONN fawzi/oracle@sle2_ma
Connected.
SQL> DECLARE
  2     CURSOR c1 IS SELECT * FROM system.customers;
  3     vcust system.customers%ROWTYPE;
  4 BEGIN
  5     SYSTEM.GGS_SECURITY.SET_USER_GROUP(USER);
  6     OPEN c1;
  7     LOOP
  8         FETCH c1 into vcust;
  9         EXIT WHEN c1%NOTFOUND;
 10         DBMS_OUTPUT.PUT_LINE(vcust.cust_name);
 11     END LOOP;
 12     CLOSE c1;
 13 END;
 14 /
JOHN V. KING
NOUR Z. LEWIS
KAREN Y. WALKER
KIMBERLY O. THOMPSON
KAREN Q. ALIBRAHIM
SARAH P. WHITE
JOHN C. DAVIS
KEVIN H. ALIBRAHIM
MARK L. MILLER
CHARLES N. WALKER
```

PL/SQL procedure successfully completed.

SQL>

Virtual Private Database Considerations

Be aware of Virtual Private Database (VPD) as an Oracle Database Enterprise feature. It is adapted mutually for data security by developing fine-grain access control at the row level and enhances application performance by reading metadata, context attribute values from Oracle shared pool memory of the System Global Area (SGA). The length of the returned predicate by the security function is limited to 4000 characters only, which may not be sufficient for some complex WHERE clauses. An alternative of DBMS_RLS is to use Oracle Enterprise Manager to create and manage policies from the graphical user interface (GUI).

Oracle GoldenGate High Availability Runtime

The following illustrates the runtime for high availability on an active-active bi-directional configuration. The building blocks for the demonstration are:

- Identical Tables Structure
- Oracle GoldenGate Active-Active Bi-Directional Topology Architecture
- Oracle GoldenGate Built-In Conflict Detection and Resolution
- Database Sequences, odd numbers for node-1 and even numbers for node-2
- Database Connections load-balancer for two-node Oracle GoldenGate high availability configuration

Figure 5-30 shows the load balancer used to handle high availability and workload balancing across the two databases, S1E2 and T1E2.

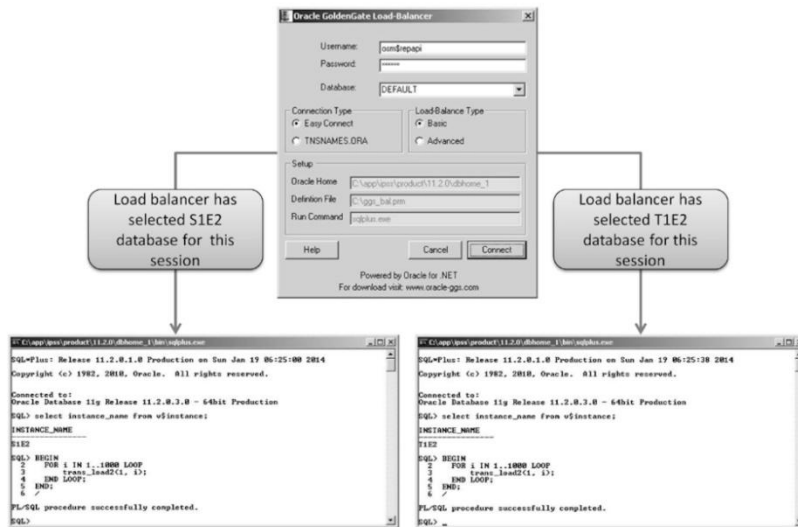


Figure 5-30: Runtime on High Availability Environment

Verify Transactions

The TRANS_LOAD2 is a workload program for generating database transactions. Oracle GoldenGate active-active bi-directional configuration synchronizes node-1 and node-2 databases, keeping the two databases identical using real-time data movement.

The test was completed successfully. The transaction size of 1000 performs concurrent 1000 inserts and replication. The logging table records conflict detection and error, if any.

Database (S1E2)	Database (T1E2)
<pre>SQL> BEGIN 2 FOR i IN 1..1000 LOOP 3 trans load2(1, i); 4 END LOOP; 5 END 6 /</pre>	<pre>SQL> BEGIN 2 FOR i IN 1..1000 LOOP 3 trans load2(1, i); 4 END LOOP; 5 END 6 /</pre>
<pre>SQL> SELECT cust no 2 FROM customers 3 WHERE rownum < 11 4 ORDER BY 1;</pre> <pre> CUST NO ----- 1006007 1006009 1006010 1006011 1006012 1006013 1006014 1006015 1006016 1006017 10 rows selected. SQL></pre>	<pre>SQL> SELECT cust no 2 FROM customers 3 WHERE rownum < 11 4 ORDER BY 1;</pre> <pre> CUST NO ----- 1006007 1006009 1006010 1006011 1006012 1006013 1006014 1006015 1006016 1006017 10 rows selected. SQL></pre>

To effectively monitor data divergence and proactively detect data inconsistency, Oracle GoldenGate Veridata is a robust tool for detecting missing rows between source and target tables. Refer to Chapter 6 to learn about implementing and using Oracle GoldenGate Veridata.

Workload Simulator

Yes, Oracle GoldenGate is a non-destructive software. It is the software installable on a production environment without major negative performance impact. However, use of the workload simulator is not to be underestimated and can be used for:

- Data Replication Testing Environments
- Benchmarking the server performance for different levels of transaction volume
- Ejection of Data Conflict for detection and resolution

To download the TRANS_LOAD2 workload simulator, visit <http://www.oracle-ggs.com>.

Summary

The use of Oracle GoldenGate 12c for business continuity has advantages and disadvantages in using the same bi-directional topology architecture to build the foundation for a disaster recovery site and high availability on supported platforms.

When using Oracle GoldenGate for disaster recovery (DR), the major point of discussion is unsupported data types. For applications not using any of the unsupported data types, Oracle GoldenGate is an option for disaster recovery solutions. Switchover and failover from the primary system to the live standby system and vice versa is an interactive process and requires manual execution of commands or scripts.

The downtime associated with switchover and failover is very low and varies from site-to-site. The chapter has illustrated the following:

- Planned moving of users from primary to live standby system for switchover
- Planned moving of users back from live standby to primary system for switchover
- Unplanned moving of users from primary to live standby system for failover
- Unplanned moving of users back from live standby to primary system for failover

When using Oracle GoldenGate for high availability (HA), detailed analysis is carried out to evaluate Oracle GoldenGate capability of achieving the level of scalability for low-high transactions volume. When Oracle GoldenGate is deployed from the ground up, the applications become replication-aware (transparent replication) which promote Oracle GoldenGate for a high availability platform. However, deploying Oracle GoldenGate at a later stage results in the challenge of handling data conflict.

This chapter has discussed and illustrated techniques of handling data conflicts using:

- Data and Applications Segmentation
- Oracle GoldenGate Built-In Conflict Detection and Resolution (CDR)
- Develop PL/SQL Sub-Programs and implement callout using SQLEXEC
- Oracle Virtual Private Database (VPD) for total segmentation

Throughout the chapter, a two-node bi-directional topology architecture is demonstrated. A three-node topology architecture, multi-master brings challenges and should be carefully qualified for disaster recovery and high availability solutions. Oracle GoldenGate considerations, in respect of disaster recovery and high availability, are discussed to lead to a sound decision and minimize the impact of post-deployment changes.

The chapter also compared Oracle GoldenGate with Oracle Real Application Clusters (RAC) and Oracle Data Guard (DG). It was made clear, for an enterprise, Oracle GoldenGate complements RAC and DG rather than replaces them.