

# Oracle9i Data Guard: SQL Apply Best Practices

*An Oracle White Paper  
September 2003*

# Oracle9i Data Guard: SQL Apply Best Practices

Introduction .....	3
SQL Apply Best Practices & Observations – At a Glance .....	4
Transaction Consistency .....	4
Database Configuration .....	4
Database Parameters .....	4
SQL Apply Parameters .....	4
Switchover and Failover Timings .....	5
Additional Information .....	5
Characteristics of the SQL Apply database .....	6
SQL Apply Process Flow .....	7
SQL Apply Process description .....	7
SQL Apply Consistency Options .....	9
Full-Consistency (FULL) .....	9
Read_Only-Consistency (READ_ONLY) .....	9
No-Consistency (NONE) .....	10
SQL Apply Consistency – Example .....	11
Results from the different transactional concurrency settings: .....	11
SQL Apply Consistency Recommendation .....	13
SQL Apply Best Practices .....	14
SQL Apply Database Configurations .....	14
Role Reversal .....	20
SQL Apply Switchover .....	20
SQL Apply Failover .....	22
SQL Apply engine Performance results .....	24
Results from the OLTP workload .....	24
Monitoring and Tuning the SQL Apply engine .....	25
SQL Apply progress .....	25
“lcr cache” .....	27
Capturing Statistics on the SQL Apply Instance .....	28
Configuring the standby database .....	28
Capturing statistics in the standby database .....	29
Troubleshooting the SQL Apply engine .....	30
Troubleshooting SQL*Loader sessions .....	30
Troubleshooting inconsistent data .....	31
Troubleshooting Pageouts .....	32
Troubleshooting Long Running Transactions .....	33
Troubleshooting ITL pressure .....	35
Conclusion .....	37
Appendix A – Populating a new Schema .....	38
Appendix B – Test Environment .....	41
Database Server Hardware and Software .....	41

# Oracle9i Data Guard: SQL Apply Best Practices

## INTRODUCTION

With Oracle Data Guard SQL Apply, Oracle is addressing the requirements of the business community for an online disaster recovery solution that also provides a means to offload reporting and decision support operations from the primary database.

Oracle SQL Apply has the following benefits:

- Allows the standby database to be open for normal operations.
- Allows additional objects to be built and maintained.
- Performs all of its operations on the standby node, and requires minimal additional processing on the primary nodes.

This white paper describes the characteristics of the SQL Apply engine, and determines the configuration best practices for Oracle Data Guard SQL Apply, thereby optimizing the application of the changes to the standby database.

Updates to this white paper may be found on the Oracle Technology Network Website at <http://otn.oracle.com/deploy/availability/htdocs/maa.htm>.

## SQL APPLY BEST PRACTICES & OBSERVATIONS – AT A GLANCE

The following section provides a summary of the best practices for Oracle Data Guard SQL Apply. For additional information, please see “[SQL Apply Best Practices](#)” on page 14.

### Transaction Consistency

Oracle SQL Apply supports the following methods of data application:

- For a reporting or decision support system, FULL or READ\_ONLY transaction consistency is provided.
- For a disaster recovery solution or when the SQL Apply engine needs to catch-up, NONE transaction consistency is provided.

If the standby database will be used for reporting or decision support operations, then:

If the standby database will be used for query only operations, then choose READ\_ONLY.

If the standby database also needs to support local write operations then choose FULL.

The Read\_Only-consistency model provided a 55% transactional apply rate improvement over the Full-consistency, and the No-consistency model provided a 74% transactional apply rate improvement over the Full-consistency model.<sup>1</sup>

### Database Configuration

It is recommended that the primary database be configured with “force logging” enabled.

```
SQL> alter database force logging;
```

### Database Parameters

Set the initialization parameter SHARED\_POOL\_SIZE on the standby database to the same as the primary database.

Increase the initialization parameter PARALLEL\_MAX\_SERVERS by the larger of 9 or (3 + (3 x CPU)) on both the primary and standby instances -

```
PARALLEL_MAX_SERVERS=current value +  
max(9, (3+ (3 x CPU)))
```

### SQL Apply Parameters

Leave the SQL Apply parameter MAX\_SGA unset.

Set the SQL Apply parameter TRANSACTION\_CONSISTENCY as determined above.

---

<sup>1</sup> For more details see “[SQL Apply Consistency Recommendation](#)”

Set the SQL Apply parameter `MAX_SERVERS` to the larger of 9 or 3 x CPU -  
`MAX_SERVERS=max(9, (3+ (3 x CPU)))`

Set the SQL Apply parameter `APPLY_DELAY` to the length in time for a user error or failure to be detected and for someone to stop the apply engine.

Set the SQL Apply parameter `_EAGER_SIZE` to 1000.

Explicitly declare the database objects that do not need to be replicated to the standby database, using the `DBMS_LOGSTDBY.SKIP` procedure.

### **Switchover and Failover Timings**

Switchover and Failover operations in a SQL Apply environment have been recorded in sub 25 seconds and sub 15 seconds respectively - see “Oracle9i Data Guard: Role Management Best Practices” paper at <http://otn.oracle.com/deploy/availability/htdocs/maa.htm> for more details.

### **Additional Information**

This white paper builds on the information in the “Oracle Data Guard: Concepts and Administration Release 2 (9.2)” manual, and in particular the information provided in the chapters “Creating a Logical Standby Database” and “Managing a Logical Standby Database”.

For Oracle’s Maximum Availability Architecture blueprint based on proven Oracle high availability (HA) technologies and best practice recommendations to maximize systems availability and make it easy to set up and configure please see <http://otn.oracle.com/deploy/availability/htdocs/maa.htm>.

## **CHARACTERISTICS OF THE SQL APPLY DATABASE**

A SQL Apply database is created from a backup of the primary database, but unlike a Redo Apply database, a SQL Apply database is available for read/write operations that differ from the primary database. Understanding the characteristics of the SQL Apply database is essential to properly monitoring, tuning, and troubleshooting it.

In the simplest configuration, the SQL Apply engine reads the archived redo files sent from the production database, and translates the information into SQL statements that are then executed against the SQL Apply database. For this purpose, a SQL Apply database is also referred to as a Logical Standby database. The standby database is a logical copy of the primary database, but since the database is open, users can query the objects at the same time the SQL is applied. Thus, a logical standby database can be used for data protection as well as reporting.

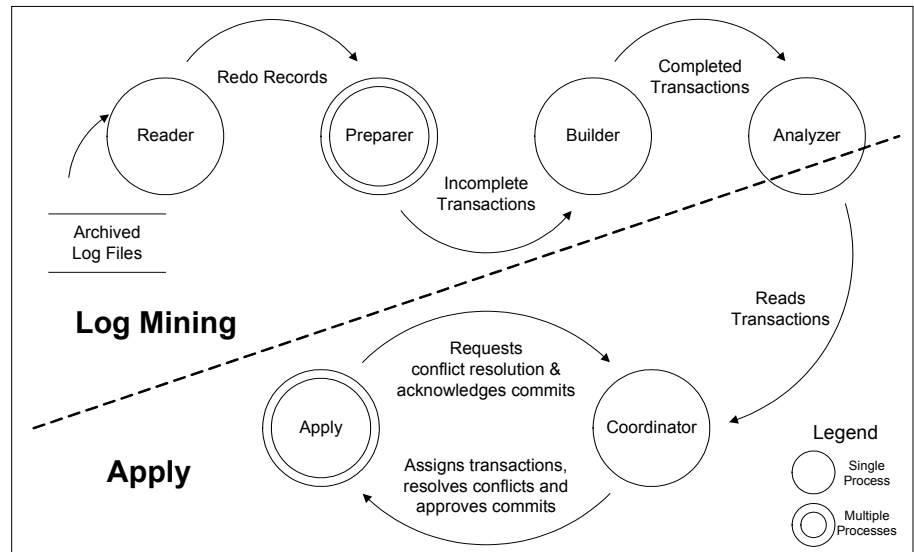
One of the benefits of the logical standby database is the ability of the standby database to support additional objects, such as indexes and materialized views, thereby allowing for the optimization of queries for reporting or decision support systems.

It is also possible for the SQL Apply engine to skip selected database objects thereby allowing the objects to be manipulated locally on the logical standby database. An example of this would be the population of auditing or temporary tables that might be maintained by a report or job scheduler.

## SQL Apply Process Flow

The SQL Apply engine consists of six distinct processes that together perform the process of applying the changes from the primary database to the standby database. The following diagram shows the flow of information, and the role that each distinct process performs.

## SQL Apply Process description



The following descriptions represent the typical purposes of the different processes that constitute the SQL Apply engine.

1. The **Reader process** scans the redo in the archived redo logs that have been registered with the SQL Apply database. The redo is then loaded into the shared pool.
2. One or more **Preparer processes** translate the redo into Logical Change Records (LCR) that are stored in a portion of the shared pool called the “lcr cache”. At this time, the Logical Change Records do not represent any specific transactions. Additionally, the Preparer processes start to identify dependencies between the LCR’s, based upon primary key values.
3. The **Builder process** constructs complete transactions from the individual Logical Change Records.
4. The **Analyzer process** takes the completed transactions from the “lcr cache” and completes the work of identifying dependencies between the different transactions, and then passes them onto the Coordinator processes.
5. The **Coordinator process** performs three key functions:
  - a) First, the coordinator process is responsible for assigning transactions to the apply slaves.

- b) Second, the coordinator process monitors the dependencies between transactions and coordinates the correct scheduling of LCR's in different transactions.
  - c) Finally, the coordinator process is optionally responsible for authorizing the commit of the changes to the SQL Apply database. This is only applicable when the SQL Apply engine is running in the "FULL" or "Read Only" consistency model.
6. One or more **Apply processes** perform three key functions:
- a) Applies the Logical Change Records for the assigned transaction to the SQL Apply database.
  - b) If there are any unresolved dependencies in a transaction, the apply process will ask the coordinator for approval to apply the change.
  - c) Depending upon the consistency model of the SQL Apply engine, the Apply process will either commit the transaction immediately, or ask for approval from the coordinator process before committing the changes to the standby database.

The "lcr cache" is a part of the variable shared pool that is allocated dynamically. As previously mentioned, this is used to stage redo records read by the reader process, as well as to stage the LCRs (Logical Change Records) that are created by the preparer processes as a result of the transformation of the redo records.

An LCR usually stays in the "lcr cache" until the corresponding transaction has been successfully applied by the apply process and released. There are times when a transaction may be deemed "uninteresting" (e.g., transactions that only modify tables not getting applied to the standby database) and these LCR's are released without being applied. There are other times when the "lcr cache" gets full, and an LCR will be written out to disk to make room so that log mining can continue (e.g., a case where a 200MB LOB is inserted into a table while the lcr cache size is set to 20MB). See "[Troubleshooting Pageouts](#)" on page 32.

## SQL APPLY CONSISTENCY OPTIONS

The SQL Apply engine is capable of applying the changes to the standby database in three different ways.

- [Full-Consistency](#)            Strict transaction ordering
- [Read\\_Only-Consistency](#)    Granular ordering to within a second
- [No-Consistency](#)             Disaster Recovery

Each option has its pros and cons and should be used based on how it best fits the requirements. Use the `TRANSACTION_CONSISTENCY` parameter of the `DBMS_LOGSTDBY.APPLY_SET` procedure to control how transactions are applied to the logical standby database. The default setting is `FULL`, which applies transactions to the logical standby database in the same order in which they were committed on the primary database. See the “[SQL Apply Best Practices](#)” on page 14 for the proper syntax.

Regardless of the consistency model chosen, the data in the logical standby database will always be transactionally consistent with the primary database when the SQL Apply engine is stopped normally.

Additionally, if two transactions update the same record in the database, then these two transactions are always applied in the correct order, regardless of the consistency model chosen.

### **Full-Consistency (FULL)**

The Full consistency model, which is the default, ensures that the transactions are applied to the SQL Apply database in the exact same sequence, as they were committed on the primary database. Using the Full consistency model imposes strict ordering of commits between transactions based on the ordering seen in the archive redo log files whether or not such orderings were intentionally imposed by the user on the primary database.

### **Read\_Only-Consistency (READ\_ONLY)**

The Read\_Only-consistency model (set `TRANSACTION_CONSISTENCY=READ_ONLY`) does not enforce the strict ordering in which transactions are applied to the SQL Apply database. Under the Read Only model, the apply processes are assigned transactions as they become available, similar to the No consistency model, but once a second, the coordinator process determines a consistent point, and publishes the SCN. All transactions are directed to use this consistent SCN.

This allows the transactions to be applied to the standby database at a faster speed than possible under the Full-consistency model, but also allows for queries to be executed on the standby that reflect a consistent view of the data.

However a user connected to the database as user SYS or with SYSDBA privileges will not see a consistent view of the data.

**No-Consistency (NONE)**

The No consistency model (set `TRANSACTION_CONSISTENCY=NONE`), does not enforce the order in which transactions are applied to the SQL Apply database, and there is no attempt to hide this fact from the users who are querying the database. This allows the transactions to be applied to the standby database in the shortest possible time, for example, when the standby database needs to catch up to the latest transactions during failover.

Note: If two transactions update the same record in the database, then these two transactions will be applied in the correct order.

## SQL Apply Consistency – Example

Consider the following example:

Session 1's transaction executes the SQL Statement:

```
SQL> update emp
2     set sal = sal * 1.1
3     where deptno = 10
4         and hiredate between to_date('01/01/01')
5                                 and to_date('12/31/02');

SQL> commit;
```

This statement takes approx 10 seconds to complete.

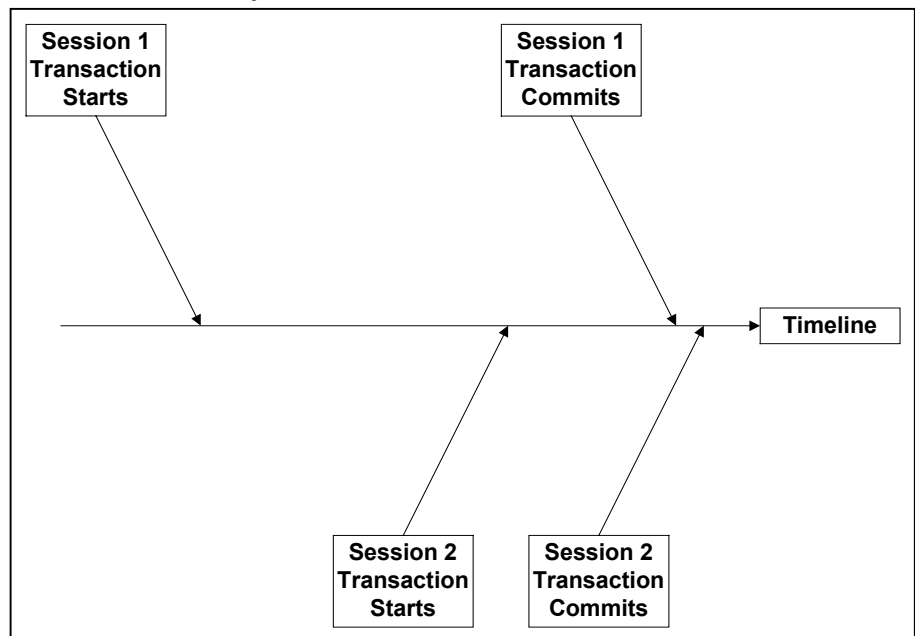
Session 2's transaction executes the SQL Statement:

```
SQL> insert into emp
2     (empno, name, deptno, sal, hiredate)
3     values ( 15123, 'Joe', 10, 9000, to_date('06/01/03'));

SQL> commit;
```

This statement takes less than 1 second to complete.

### Time Line on the Primary



From the timeline, we can see that Session 2's transaction starts towards the end of Session 1's transaction, and commits soon after session 1's transactions commits.

### Results from the different transactional concurrency settings:

#### Full-Consistency

Under the Full Consistency model, the SQL Apply engine guarantees that transaction 1 will commit before transaction 2 on the standby database.

**Read\_Only-Consistency**

Under the Read Only consistency model, the SQL Apply engine will guarantee either of the following on the standby database;

- a) Transaction 1 will commit before transaction 2
- b) Transaction 1 and transaction 2 will be seen at the same time.

Note: A query executed against the Logical Standby database will never show transaction 2 without also showing transaction 1, when running in Read\_Only consistency.

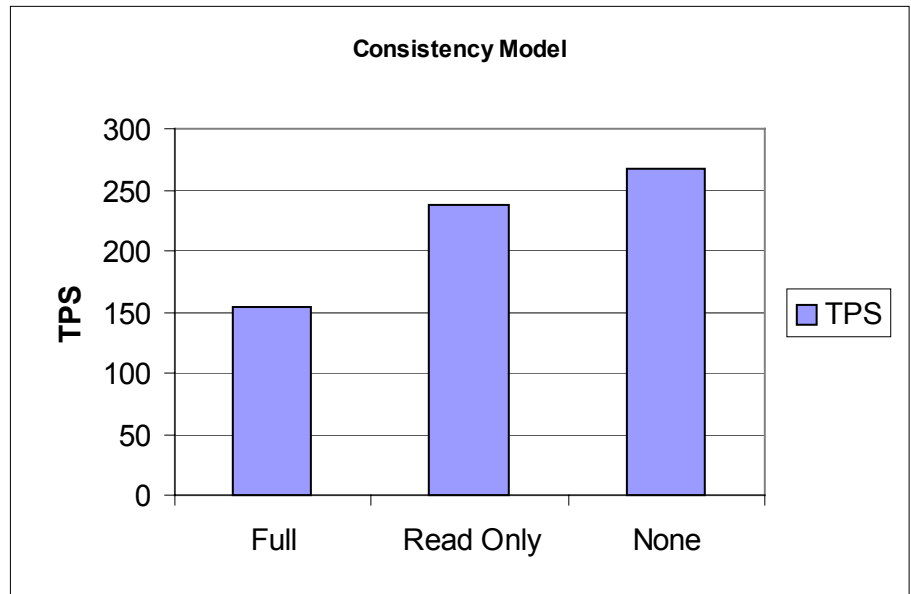
**No-Consistency**

Under the No-Consistency model, the SQL Apply engine does not guarantee ordering. Therefore, it is probable that transaction 2 will be committed before transaction 1 on the standby database, and a query executed on the logical standby database will show transaction 2 results but not transaction 1 results.

## SQL Apply Consistency Recommendation

If the standby database is to be used purely for disaster recovery purposes, then use the No-consistency model. However, you should also consider Oracle Data Guard Redo Apply to see if this better fits their requirements. If the logical standby database will be for offloading work from the primary machine, such as reporting, then they should consider either the Full consistency model or the Read Only consistency model.

Using a consistent workload, the following chart shows the transactional throughput that was achieved with the different consistency models.



The Read\_Only consistency model provided a 55% transactional apply rate improvement over the Full-consistency, and the No-consistency model provided a 74% transactional apply rate improvement over the Full-consistency model<sup>2</sup>.

---

<sup>2</sup> The results you experience will depend on your redo generation rate and commit rate. Testing with your own application(s) and systems is recommended to validate your service level requirements.

## SQL APPLY BEST PRACTICES

The following are the best practices for the configuration of the SQL Apply engine based upon the testing that has been performed by the High Availability Systems Group within Oracle.

### SQL Apply Database Configurations

There are four key aspects to the configuration of the SQL Apply database:

- [Schema Best Practices](#)
- [Role Reversal Best Practices](#)
- [Database Parameter Best Practices](#)
- [SQL Apply Parameter Best Practices](#)

#### Schema Best Practices

Apart from the object and data type restrictions that are discussed in the Oracle Data Guard Concepts and Administration manual, the following schema best practices are recommended.

#### Interested Transaction List (ITL) pressure

To avoid ITL pressure customers may need to increase the number of INITRANS for a particular database object.

ITL pressure may occur when the production application has many small transactions modifying rows in the same Oracle block, and due to the parallel capabilities of SQL Apply engine, many Apply processes are responsible for the insertion of the data, creating contention in the standby database. This results in a degree of parallelism that was not present on the primary database and this put pressure on the objects interested transaction list (ITL).

Additional information on identifying and resolving this problem is included in the “[Troubleshooting the SQL Apply engine](#)” on page 30.

It is recommended that the current production application be analyzed to identify processes that exhibit the behavior mentioned. For objects that the process operates on, the INITRANS value should be increased to a number not greater than the number of apply slaves available to the SQL Apply engine. Due to role reversal, the change should occur on both production and logical standby databases.

#### System generated constraint names

To prevent DDL replication problems, customers are advised to avoid using system-generated names for Primary and Unique key constraints.

The problem occurs when the customer wishes to rebuild the index for performance reasons. To rebuild the index, it is possible that an administrator specifies the system generated constraint name as created on the primary database.

However, since the standby database may support additional indexes or materialized views, the name generated when the primary key constraint is built on

the primary database may be different from the constraint name used on the standby database.

To ensure that the DDL operation on the primary database is correctly replicated to the standby database, all primary and unique key constraint should be created with a user specified constraint name.

Alternatively, skip handlers (DBMS\_LOGSTDBY.SKIP) may be defined to skip these operations, and these constraints and indexes can be managed separately on the logical standby database.

### Role Reversal Best Practices

To ensure a simple switchover operation, it is recommended that Oracle database links, in both directions, be created at the time the SQL Apply database is instantiated. The following steps should be completed against both the primary and standby database.

```
SQL> connect sys/<password>
SQL> create database link <location>
  2   connect to system
  3   identified by <password>
  4   using '<tns alias>';
```

where 'tns\_alias' references an Oracle Net connection descriptor that connects to the remote database. The 'tns\_alias' string is the same as the Log Transport Service string – see “[Oracle9i Data Guard: Primary Site and Network Configuration Best Practices](http://otn.oracle.com/deploy/availability/htdocs/maa.htm)” paper at <http://otn.oracle.com/deploy/availability/htdocs/maa.htm> for more details.

It is recommended that the Oracle database link be the same name on both the primary and standby databases, and that the name reflects the role that the target database will be performing.

The following examples show the results of the same query executed against the two different nodes in the Logical Standby environment.

When executed on the primary site

```
SQL> select local.host_name local_host
  2   , remote.host_name remote_host
  3   from v$instance@<location> remote
  4   , v$instance local;
```

LOCAL_HOST	REMOTE_HOST
primary	secondary

When executed on the secondary site

```
SQL> select local.host_name local_host
  2   , remote.host_name remote_host
  3   from v$instance@<location> remote
  4   , v$instance local;
```

LOCAL_HOST	REMOTE_HOST
secondary	primary

## **Database Parameter Best Practices**

Oracle Data Guard SQL Apply requires very few changes to the database system parameter file. The Creating a Logical Standby Database chapter of the Oracle Data Guard Concepts and Administration manual provides a list of the parameters that should be changed.

In addition to this list, the following is a discussion on the setting of two init.ora parameters that are critical to the operation of the SQL Apply engine.

### **SHARED POOL SIZE**

The SHARED\_POOL\_SIZE parameter has a direct influence on the amount of memory that can be assigned to the SQL Apply engine and in particular the “lcr cache” memory region. On an SMP machine, the amount of memory allocated to the “lcr cache” memory region by default is 1/4<sup>th</sup> the Shared Pool. N.B. The “lcr cache” must not be greater than 1/4<sup>th</sup> of the Shared Pool.

When deciding on the size of the shared pool, it should be noted that the SQL Apply instance does not parse SQL statements that resulted from queries on the production database. In addition, the SQL statements constructed by the SQL Apply engine are generic for each object, so there will only ever be three SQL Statements (INSERT, UPDATE, DELETE) for each table managed by the SQL Apply engine. The number of SQL statements required by the SQL Apply engine is likely to be significantly lower than the number of SQL statements found on the production database.

Finally, following role reversal of the Logical Standby database, in the event of a scheduled maintenance SWITCHOVER or FAILOVER, the “lcr cache” memory region will no longer exist on the new primary database, so the memory will be allocated to other components of the shared pool such as the “sql area” and “library cache”.

It is recommended that the “SHARED\_POOL\_SIZE” for the SQL Apply instance be initially the size of the production instance, to support role reversal. In addition, if the application supports large batch processing, with infrequent commits, then SQL Apply paging may occur (see “[Troubleshooting Pageouts](#)” on page 32), in which case, the SHARED\_POOL\_SIZE may need to be increased.

### **PARALLEL MAX SERVERS**

The PARALLEL\_MAX\_SERVERS parameter specifies the maximum number of parallel query processes that can be created on the database instance. With the exception of the coordinator process, all the processes that constitute the SQL Apply engine are created from the pool of parallel query processes. The SQL Apply engine, by default, uses all the parallel query processes available on the database instance. This behavior can be overridden using the logical standby parameters “MAX\_SERVERS” (see page 17).

It is recommended that the PARALLEL\_MAX\_SERVERS parameter be increased by the value of the SQL Apply parameter MAX\_SERVERS.

### **SQL Apply Parameter Best Practices**

The SQL Apply engine has its own dedicated set of parameters that are set via the DBMS\_LOGSTDBY package procedure.

Calling the APPLY\_SET procedure as follows sets the parameters;

```
execute dbms_logstdby.apply_set('<parameter name>',value);
```

To cancel or unset a specific parameter use the APPLY\_UNSET procedure as follows;

```
execute dbms_logstdby.apply_unset('<parameter name>')
```

N.B. The APPLY\_SET and APPLY\_UNSET procedure may only be executed when the SQL Apply engine is not running. If the SQL Apply engine is running when the command is executed, an ORA-16103 error will be generated.

The following parameters were found to provide the most benefit.

#### **MAX\_SGA**

The MAX\_SGA parameter, specified in terms of megabytes, allows the default allocation for the “lcr cache” of 1/4<sup>th</sup> the Shared Pool to be modified. This parameter must not exceed 1/4<sup>th</sup> of the Shared Pool limit.

It is recommended that this parameter be left unset, and be allowed to default to 1/4<sup>th</sup> the Shared Pool.

#### **TRANSACTION\_CONSISTENCY**

The TRANSACTION\_CONSISTENCY parameter specifies the consistency model under which the SQL Apply engine should run. The possible values for this parameter are FULL, READ\_ONLY and NONE. For a detailed description of these values and the effect they have on the SQL Apply engine, please see the “[SQL Apply Consistency Options](#)” on page 9.

#### **MAX\_SERVERS**

The parameter specifies the number of parallel query servers that the SQL Apply engine will reserve when started.

If the SQL Apply database is being used for the purpose of offloading reporting or decision support operations from the primary database, then it is likely that you will want to reserve some of the parallel query slaves for such operations. Since the SQL Apply engine by default uses all the parallel query slaves, setting the logical standby MAX\_SERVERS parameter allows for a certain number of parallel query slaves to be held back for such operations.

It is recommended that this parameter be initially set to the larger of 9 or  $(3 + (3 \times \text{CPU}))$  initially –  $\text{MAX\_SERVERS} = \max(9, (3 + (3 \times \text{CPU})))$ .

The following table shows the number of servers reserved for Parallel Query Operations and the number of servers used by the SQL Apply engine based upon the setting of the init.ora parameter PARALLEL\_MAX\_SERVERS and the SQL Apply parameter MAX\_SERVERS.

init.ora parameter <b>PARALLEL_MAX_SERVERS</b> on the standby	SQL Apply parameter <b>MAX_SERVERS</b> on the standby	Number of Servers reserved for Parallel Query Operations	Number of Servers reserved for SQL Apply Operations
27	Unset	0	27
27	27	0	27
51	27	24	27

#### **APPLY DELAY**

The `APPLY_DELAY` parameter specifies how long the SQL Apply engine should delay the application of the archived redo log relative to the time the log file was registered with the logical standby database. The delay time should be set to a value large enough for the administrators to identify, report, and ultimately stop the SQL Apply engine, thereby preventing the error from being replicated to the logical standby database.

This parameter is specified in terms of minutes.

There is no recommended value for this parameter.

Note: An alternative method of setting the `DELAY` for the application of an archive redo log, is to use the `DELAY=<time>` clause of the `LOG_ARCHIVE_DEST_n` parameter on the Primary database.

#### **EAGER SIZE**

The `_EAGER_SIZE` parameter specifies the minimum number of rows that can be modified by a single transaction, before which the transaction is deemed an eager transaction. An eager transaction differs from the typical transaction processed by the SQL Apply engine as follows: Under normal circumstances, the SQL Apply engine passes committed transactions to the coordinator and ultimately onto the apply slaves. When an eager transaction is identified, the coordinator associates an Apply slave to service the transaction, and the SQL statements are passed to the apply slave as they are prepared, removing the need for the complete transaction to be built.

The identification of an eager transaction is required for two reasons;

1. If a transaction updated 1 million rows on the primary database, then the size of the “lcr cache” might be insufficient to support the 1 million SQL statements that are required to apply the transaction to the standby database.
2. By allowing the SQL statements to be applied to the standby database, as they are prepared, the length of time for the transaction to be committed is significantly less than if the transaction was only assigned to the Apply slave after the complete transaction had been built. This means that the SQL

Apply engine is able to continue applying new transactions to the standby database much faster, assuming Full-consistency.

`_EAGER_SIZE` is specified in terms of rows modified.

It is recommended that `_EAGER_SIZE` be set to 1000.

### **DBMS\_LOGSTDBY.SKIP**

For database objects that do not need to be applied to the standby database, the customer is advised to skip these objects using the `DBMS_LOGSTDBY.SKIP` procedure.

By skipping such objects, the SQL Apply engine will not need to perform the unnecessary operations of maintaining a table that is not required.

This is more likely to occur in a decision support environment.

## ROLE REVERSAL

Oracle Data Guard SQL Apply provides the ability to reverse the roles of the primary and logical standby database. Even if the primary reason for the creation of the SQL Apply database was to offload work from the primary database, the SQL Apply database can become the primary database either in the event of a failure or in the event of scheduled maintenance that affects the primary database.

Note: It is recommended that the Data Guard Broker be used for Role Reversal operations when both the primary and the logical standby databases consist of a single instance each.

### SQL Apply Switchover

SQL Apply Switchover is a planned operation where by the roles of the primary and logical standby database are reversed.

To complete a switchover or role reversal between the primary database and the logical standby database, five steps need to be completed in order.

#### Pre-switchover

- a) Prior to commencing a scheduled switchover, the user should first remove any delay from the SQL Apply engine<sup>3</sup>.

```
SQL> alter database stop logical standby apply;  
SQL> execute dbms_logstdby.apply_unset('APPLY_DELAY');  
SQL> alter database start logical standby apply;
```

Note: If you have previously setup the DELAY parameter on the Primary database using the LOG\_ARCHIVE\_DEST\_n parameter, then the DELAY should be changed to 0 on the Primary database, and a log switch issued.

- b) In addition, the user should register as much of the production redo with the SQL Apply database. This is recommended to reduce the length of time that the final switchover process takes, during which time, users are unable to access a primary database. On the primary database, force a log switch of the current redo log file.

```
SQL> alter system archive log current;
```

It might be useful to issue multiple log switches on the primary database in the run up to the execution of the switchover command, assuming a scheduled switchover.

#### Switchover

1. At the time that the switchover is to occur, the user should issue the first command in the switchover procedure on the primary database. The primary

---

<sup>3</sup> After switchover or failover has completed, the DELAY if present should be reset to the original value.

database will not be able to process any transactions once this command completes, until the switchover operation completes.

```
SQL> alter database commit to switchover to logical standby;
```

On an idle system, this command should take approximately 2 seconds to complete. On a system with a moderate load, the command should complete within approximately 20 seconds, and is dependent upon the number and size of transactions that need to be rolled back.

2. Once the command returns, the last of the online redo log files has been transferred and registered with the current logical standby database. The SQL Apply engine is currently applying these changes, but before continuing with the role reversal, the log transport service on the primary database should be disabled. If `log_archive_dest_4` were configured to transfer the redo log files from the primary database to the standby database, the command would be:

```
SQL> alter system set log_archive_dest_state_4=defer scope=both;
```

3. On the standby database, the last of the changes from the primary database are being applied. Once the last change has been applied, role reversal may continue. When the SQL Apply engine receives the last change, look for either of the following two messages before continuing.

Firstly, the alert log of the SQL Apply instance will record a message similar to the following

```
Tue Apr 15 11:29:15 2003
LOGSTDBY event: ORA-16128: User initiated shut down
successfully completed
```

Alternatively, the `DBA_LOGSTDBY_EVENT` object can be queried as follows;

```
SQL> select 'found'
2   from dba_logstdby_events
3  where event_time = ( select max(event_time)
4                      from dba_logstdby_events )
5     and status_code = 16128;
```

Once either the message is recorded in the alert log, or the message 'found' is returned from the query, the current logical standby database is in a position to complete the role reversal. The time for this status to occur is directly related to the number of transactions that are pending application to the SQL Apply database.

4. On the current standby host, issue the following commands.

```
SQL> alter system set log_archive_dest_state_4=enable scope=both;
SQL> alter database commit to switchover to primary;
```

The alter system command advises the still current SQL Apply instance to start shipping the archived redo log files to the new standby database, assuming that `log_archive_dest_4` is configured to transfer the files from the new primary to the new standby database.

The alter database command actually causes the role reversal to complete, and on completion of the command, users will be able to issue transactions against the new primary database.

5. Since role reversal has occurred at this point, the terms used to refer to the hosts should also reverse. Therefore, on the new standby host, start the SQL Apply engine as follows;

```
SQL> alter database start logical standby apply new primary
<location>;
```

Where `<location>` refers to a database link that is defined in the new standby database and connects to the new primary database, see “[Role Reversal Best Practices](#)” on page 15.

At this time, the role reversal has completed, and the SQL Apply engine has started to apply the transactions from the new primary database.

Following these best practices, a switchover operation involving an idle production database a logical standby database, maybe completed in less than 25 seconds. During this time, neither database is available for user access.

Note: If the SQL Apply Database contains a large number of database objects, the coordinator process maybe in the INITIALIZING state for a long period of time.

### SQL Apply Failover

SQL Apply Failover is an unplanned operation, that results in the current logical standby database being open as an independent database.

To complete a failover to the SQL Apply database, only one command is issued. However, prior to the issuing of the command, four steps may be completed to minimize the potential data loss.

#### Failover

1. The user should first remove any delay from the SQL Apply engine if full recovery or minimal data loss recovery is desired<sup>3</sup>.

```
SQL> alter database stop logical standby apply;
SQL> execute dbms_logstdby.apply_unset('APPLY_DELAY');
SQL> alter database start logical standby apply;
```

2. If the log transport service is using the LGWR process to transfer the redo information, then there might be information available in a partially created archive redo log file, residing on the standby host.

Assuming the primary database is shutdown, any logfile opened by the LGWR process will be closed, and the log transport service will advise that there is a partial log file available.

```
RFS: Possible network disconnect with primary database
Closing latent archivelog for thread 2 sequence 111
EOF located at block 918 low SCN 0:1536302 next SCN 0:1541253
Latent archivelog '/arch1/SALES/arch_2_0000000111.log'
```

To register the log file with the SQL Apply engine use the following command:

```
SQL> alter database register logical logfile '<log file name>';
```

3. If the log transport service is using the ARCH process to transfer the redo information, then there might be information available on the current primary host. If the administrator has decided on a failover because of an error affecting the primary database, but the host is still accessible, it might be possible to transfer the current online redo log files and any missing archived redo log files to the standby host. The files should first be copied to the standby host, using your preferred commands, and then as before, the files can be manually registered with the SQL Apply engine.

```
SQL> alter database register logical logfile '<log file name>';
```

4. Once the last of the available data has been applied to the logical standby database, the SQL Apply engine must be stopped. To verify that the Apply engine has applied all the possible work, execute the query as described in the section “newest\_scn” on page 26.

```
SQL> alter database stop logical standby apply;
```

5. At this time, database recovery has been successful to the latest possible transaction, and the failover may commence.

```
SQL> alter database activate logical standby database;
```

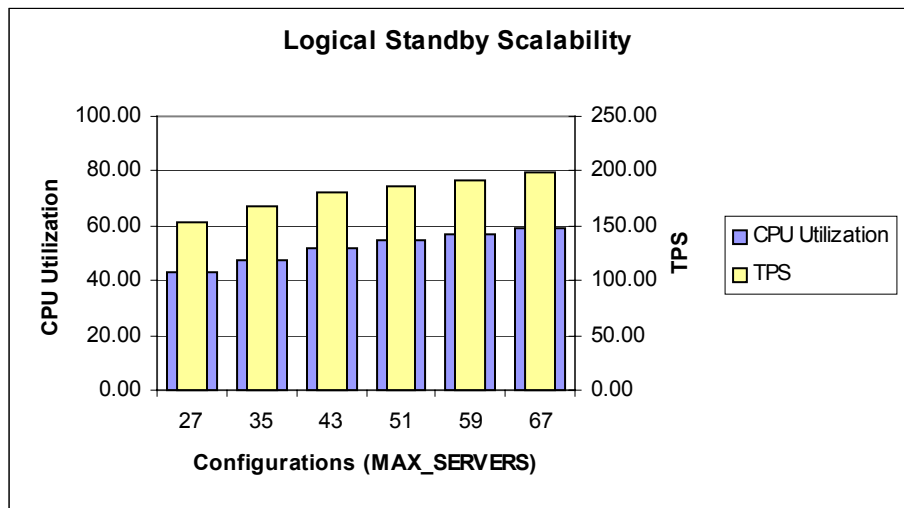
The database has now been opened as an independent database. In order to create a new standby database based upon this database, a new standby database must be instantiated. The procedure for instantiation of a SQL Apply database is documented in the Oracle Data Guard Concepts and Administrator manual.

## SQL APPLY ENGINE PERFORMANCE RESULTS

An OLTP workload was used to derive and test the best practices mentioned on page 14. In addition, a number of SQL\*Loader tests was also performed to simulate a batch transaction workload to confirm that the recommendations for an OLTP workload still applied to a batch workload.

### Results from the OLTP workload

For the OLTP workload, the production database was generating transactions at a rate of 143 transactions per second (TPS) over the course of 3 hours. The graph shows six different configurations for the SQL Apply parameter `MAX_SERVERS`, scaling from the default recommendation of 27 [ 3+(3 x CPU) for an 8-way server] up to 67. All six configurations utilized the `FULL` transactional consistency option of the SQL Apply engine, with a 500Mb “lcr cache” memory region.



It can be seen from the graph above that the recommended default is capable of applying transactions at a rate of approx 155 transactions, which indicates that transactions are being applied to the standby database faster than the production database was generating them.

The additional data points show that with the increase in the number of apply processes, the SQL Apply engine had a 29% improvement in the transactional apply rate at a cost of a 36% increase in CPU utilization.

## MONITORING AND TUNING THE SQL APPLY ENGINE

The “lcr cache” along with the number of processes assigned to the SQL Apply engine has the largest effect on the performance of the logical standby database.

Additionally, the consistency model that is being used also has a major impact on the transactional throughput. Please see “[SQL Apply Consistency Options](#)” on page 9 for additional information.

### SQL Apply progress

The SQL Apply engine reports three key metrics via the `DBA_LOGSTDBY_PROGRESS` view. This view reports the progress of the SQL Apply engine in terms of System Change Number (SCN) as recorded on the primary database. The three metrics are:

1. The **read\_scn** reports the earliest SCN that is required for the SQL Apply engine to start.
2. The **applied\_scn** reports the last SCN that has been applied to the standby database.
3. The **newest\_scn** reports the last SCN that has been registered with the standby database and that the SQL Apply engine is able to apply.

#### read\_scn

The **read\_scn** from the `DBA_LOGSTDBY_PROGRESS` view when compared to the `DBA_LOGSTDBY_LOGS` view, provides a list of redo log files that are no longer required by the SQL Apply engine, and can therefore be purged from the file system that holds them.

```
SQL> alter session set nls_date_format='YY/MM/DD HH24:MI:SS';
SQL> select thread# thr#
       2      , max(sequence#) seq#
       3      from dba_logstdby_log log
       4      , dba_logstdby_progress prog
       5      where log.next_change# < prog.read_scn
       6      group by thread#
       7      order by thread#;
```

```
THR#    SEQ#
----  -
1      291
```

This output indicates that the archived redo log files sequence# 291 and below from thread #1 are no longer required by the SQL Apply engine and can therefore be safely deleted from the standby host.

#### applied\_scn

The **applied\_scn** from the `DBA_LOGSTDBY_PROGRESS` view when compared to the `DBA_LOGSTDBY_LOGS` view, provides a list of the redo log files that the SQL Apply engine is currently processing.

```
SQL> alter session set nls_date_format='YY/MM/DD HH24:MI:SS';
```

```

SQL> select thread# thr#
       , sequence# seq#
       , first_time
       , first_change# first_chng
       , next_time
       , next_change# next_chng
       from dba_logstdby_log log
       , dba_logstdby_progress prog
       where prog.applied_scn between log.first_change# and
log.next_change#
       order by next_change#;

```

THR#	SEQ#	FIRST_TIME	FIRST_CHNG	NEXT_TIME	NEXT_CHNG
1	292	03/03/24 12:26:50	3615938	03/03/24 12:53:46	4109054

This output indicates that the reader process has thread #1, sequence #292 open, and is actively reading the redo from this archived redo log file.

Generally, this query will report one redo log sequence number for each thread that is active on the primary database.

### newest\_scn

The **newest\_scn** from the **DBA\_LOGSTDBY\_PROGRESS** view when compared to the **DBA\_LOGSTDBY\_LOGS** view, and in combination with the **applied\_scn** value provides a list of redo log files that have been registered with the logical standby database, and that are available for processing by the SQL Apply engine.

```

SQL> alter session set nls_date_format='YY/MM/DD HH24:MI:SS';
SQL> select thread# thr#
       , sequence# seq#
       , first_time
       , first_change# first_chng
       , next_time
       , next_change# next_chng
       from dba_logstdby_log log
       , dba_logstdby_progress prog
       where log.next_change# > prog.applied_scn
       and log.next_change# <= prog.newest_scn
       order by next_change#;

```

THR#	SEQ#	FIRST_TIME	FIRST_CHNG	NEXT_TIME	NEXT_CHNG
1	292	03/03/24 12:26:50	3615938	03/03/24 12:53:46	4109054
1	293	03/03/24 12:53:46	4109054	03/03/24 13:20:47	4602057
1	294	03/03/24 13:20:47	4602057	03/03/24 13:47:57	5094985
1	295	03/03/24 13:47:57	5094985	03/03/24 14:04:12	5214535
1	296	03/03/24 14:04:12	5214535	03/03/24 14:06:21	5214579

This output indicates that the redo log files for thread #1, sequences 292 to 296 contain redo information that can be applied to the logical standby database.

If this final query reports 1 or maybe 2 rows per active thread, then this would indicate that the SQL Apply engine is capable of applying the changes to the standby database at least as fast as the transactions are generated on the primary database.

If the final query reports zero rows, then this means that the SQL Apply engine has applied all the redo that has been registered, and is therefore up to date.

If the final query reports a large number of rows, then it could mean the SQL Apply engine was stopped for an extended period of time, or that the SQL Apply

engine is not able to apply the redo to the standby database at an appropriate rate. In the case of the results obtained above, the SQL Apply engine had been stopped previously, and is now in the process of “catching-up”.

### “lcr cache”

The [applied\\_scn](#) query on page 25 reports the redo log files that are currently being read by the SQL Apply engine. When the redo has been processed, the completed transactions will be stored in the “lcr cache” region of the shared pool. To determine how many transactions are stored in the “lcr cache” that still need to be applied to the standby database, the following query can be issued.

```
SQL> select value
      2   from v$logstdby_stats
      3  where name = 'committed txns being applied';

VALUE
-----
16214
```

In the above example, 16214 transactions are to be applied.

If the Apply processes are not applying the changes fast enough, which is to say the generation rate for the transactions is higher on the production database than the SQL apply engine’s apply rate<sup>4</sup>, then additional processes may be assigned to the SQL Apply engine by increasing the `MAX_SERVERS` logical standby parameter. This will require the SQL Apply engine to be stopped, and possibly the instance to be restarted, if the `PARALLEL_MAX_SERVERS` parameter needs to be increased.

If the Apply processes are applying the changes fast enough, then there may be periods of time when both the Apply processes and the Log Mining processes have no work to perform. If this is the case, which can be verified by executing the “[newest\\_scn](#)” query on page 26, then it might be possible to reduce the number of processes the SQL Apply engine has access to, by decreasing the `MAX_SERVERS` parameter. However, before reducing the number of processes available to the SQL Apply engine, you should ensure that there are no periods during the processing window, whenby a backlog occurs, as this could potentially impact Service Level Agreements.

---

<sup>4</sup> To determine the transaction rate for both the production database and the standby database, use Oracle’s Statspack utility, or some other database monitoring utility.

## CAPTURING STATISTICS ON THE SQL APPLY INSTANCE

It is highly recommended that some form of database statistic capture is implemented on the primary database. If the customer is using Oracle's Statspack utility, to capture the statistics, then the data is stored in a schema in the local database called PERFSTAT. Rather than replicating the statistics from the production database to the standby database, it is probable that the customer would want to capture statistics about the SQL Apply database, particularly if the SQL Apply database is being used for offloading work from the production database.

### Configuring the standby database

In order to configure the SQL Apply engine to skip the replication of a particular schema, the following steps should be performed.

1. Changes to the SQL Apply engine configuration can only be performed when the SQL Apply engine has been stopped. To stop the SQL Apply engine issue the following command;

```
SQL> alter database stop logical standby apply;
```

2. Use the DBMS\_LOGSTDBY.SKIP procedure to specify that all DML and DDL operations should be skipped for the PERFSTAT schema.

```
SQL> execute dbms_logstdby.skip('SCHEMA_DDL','PERFSTAT','%');  
SQL> execute dbms_logstdby.skip('DML','PERFSTAT','%');
```

Note that the third parameter of the procedure refers to the particular object in the schema that is to be skipped, and the % means all objects in the schema should be skipped.

3. Ensure that the PERFSTAT schema has been granted logical standby administrative permissions

```
SQL> grant logstdby_administrator to perfstat;
```

4. Only after the SQL Apply engine has been started and the initialization process has completed will the changes requested by the skip procedure be activated on the standby database. To start the SQL Apply engine issue the following statement;

```
SQL> alter database start logical standby apply;
```

At this point, the SQL Apply engine will cease to apply any changes made to the PERFSTAT schema on the production database. Instead it will allow the STATSPACK routines to be run locally against the SQL Apply instance and to record the statistics in the logical standby database.

## Capturing statistics in the standby database

In order to capture the statistics from the Statspack utility in the SQL Apply database, the following steps should be performed.

1. Use the DBMS\_LOGSTDBY.GUARD\_BYPASS\_ON procedure to temporarily disable the guard for this session.

```
SQL> execute dbms_logstdby.guard_bypass_on;
```

2. Use the STATSPACK.SNAP procedure to record the instance statistics in the standby database

```
SQL> execute statspack.snap;
```

3. Use the DBMS\_LOGSTDBY.GUARD\_BYPASS\_OFF procedure to re-enable the guard for this session.

```
SQL> execute dbms_logstdby.guard_bypass_off;
```

## TROUBLESHOOTING THE SQL APPLY ENGINE

### Troubleshooting SQL\*Loader sessions

Oracle SQL\*Loader provides a method of loading data from different sources into the Oracle Database. This section analyzes some of the features of the SQL\*Loader utility as it pertains to the SQL Apply engine.

#### Replace or Append

Regardless of the method of data load chosen, the SQL\*Loader control files contain an instruction on what to do to the current contents of the Oracle table into which the new data is to be loaded, via the keywords of APPEND and REPLACE.

```
LOAD DATA
INTO TABLE LOAD_STOK APPEND
```

```
LOAD DATA
INTO TABLE LOAD_STOK REPLACE
```

When the keyword 'APPEND' is used, the new data to be loaded is appended to the contents of the LOAD\_STOK table.

When the keyword 'REPLACE' is used, the contents of the LOAD\_STOK table are deleted prior to commencing the load of the new data. Oracle SQL\*Loader uses the DELETE statement to purge the contents of the table, in a single transaction.

Rather than using the REPLACE keyword in the SQL\*Loader script, it is recommended that prior to the load of the data that a SQL\*Plus TRUNCATE TABLE command should be issued against the table on the primary database. This will have the same effect of purging both the primary and standby databases copy of the table in a manner that is both fast and efficient since the TRUNCATE TABLE command is recorded in the online redo logs and issued by the SQL Apply engine against the standby database. The SQL\*Loader script may continue to contain the REPLACE keyword, but will now attempt to DELETE zero rows from the object on the primary database. Since no rows were deleted from the primary database, there will be no redo recorded in the redo log files. Therefore, no delete statement will be issued against the standby database.

The use of the REPLACE keyword without the DDL command TRUNCATE TABLE, provides four potential problems for the SQL Apply engine, when the transaction needs to be applied to the standby database.

1. If the table currently contains a significant number of rows, then these rows need to be deleted from the standby database. Since the SQL Apply engine is not able to determine the original syntax of the statement the SQL Apply engine must issue a DELETE statement for each row purged from the primary database.

For instance, if the table on the primary database originally had 10,000 rows, then Oracle SQL\*Loader will issue a single DELETE statement to purge the

10,000 rows. On the standby database though, the SQL Apply engine does not know that all rows are to be purged, and instead must issue 10,000 individual DELETE statements, each statement purging a single row.

2. If the table on the standby database does not contain an index that can be used by the SQL Apply engine, then the DELETE statement will issue a Full Table Scan to purge the information.

Continuing with the previous example, since the SQL Apply engine has issued 10,000 individual DELETE statements, this could result in 10,000 Full Table Scans being issued against the standby database. See [“Troubleshooting Long Running Transactions”](#) on page 33.

3. If the transaction to be replicated contains a significant number of rows, then it may fall under the definition of an [“eager transaction”](#) see [“SQL Apply Database Configurations”](#) on page 14. However, if the transaction does not contain a sufficient number of rows, then the SQL Apply engine may begin to pageout - see [“Troubleshooting Pageouts”](#) on page 32.
4. Additionally, if the transaction does not contain a sufficient number of rows for it to be considered an [“eager transaction”](#), then the transaction will need to be built completely in the “lcr cache” and only when the transaction has been built will the SQL Apply engine assign it to an apply slave to be executed. This could result in an extended length of time to replicate the data.

### Troubleshooting inconsistent data

If the data is manually changed on the standby database, using the DBMS\_LOGSTDBY.GUARD\_BYPASS\_ON procedure, then this problem will occur if the primary database attempts to delete or change the same data. In the alert log of the SQL Apply instance, an error message similar to the following will be reported.

```
Wed May 7 13:19:20 2003
LOGSTDBY event: ORA-01403: no data found
LOGSTDBY stmt: LOAD.LOAD_STOK (Oper=DELETE)
```

In this example, the primary database issued a DELETE statement against the LOAD\_STOK object of the LOAD schema and the SQL Apply engine could not find the corresponding row, resulting in the SQL Apply engine aborting.

Before the SQL Apply engine can be restarted, the transaction that is attempting to delete the record from the LOAD\_STOK object must first be skipped. When the SQL Apply engine aborted, it recorded the transaction ID of this statement in the DBA\_LOGSTDBY\_EVENTS view. The transaction ID can be obtained as follows;

```
SQL> select xidusn,xidslt,xidsqn,event
2      from dba_logstdby_events
3      where current_scn =
4            ( select max(current_scn)
5*           from dba_logstdby_events )
```

XIDUSN	XIDSLT	XIDSQN	EVENT
5	7	6113	LOAD.LOAD_STOK (Oper=DELETE)

Using the XIDUSN, XIDSLT, XIDSQN values from the query above, the transaction can be skipped as follows;

```
SQL> execute dbms_logstdby.skip_transaction(5,7,6113);
```

The SQL Apply engine can now be started again as follows;

```
SQL> alter database start logical standby apply;
```

When the SQL Apply engine is initializing, it reads the list of transactions that are to be skipped, and when the transaction is read by the Log Mining processes, it will be skipped.

In order to determine the tables that are affected by the transaction, it might be advisable to analyze the transaction using the DBMS\_LOGMNR database package.

Note: An alternative to skip\_transaction, would be to skip all DML operations on the table, or to define a skip\_error for the table. The skip\_error can be defined to stop the SQL Apply engine or to ignore the error and to continue.

### Troubleshooting Pageouts

In the event that the SQL Apply engine is attempting to build a large transaction that has not met the criteria of an [eager transaction](#), the “lcr cache” may fill up. Rather than immediately reporting an error, similar to an ORA-4031, the SQL Apply engine attempts to pageout portions of the “lcr cache” in a way similar to that of the OS paging portions of real memory.

To determine if SQL Apply pageouts are occurring, the following SQL statement should be issued.

```
SQL> select value
2   from v$logstdby_stats
3  where name = 'pageouts';
```

```
VALUE
-----
7517
```

In this example, a total of 7517 pageouts have occurred since the SQL Apply engine was started.

If pageouts are occurring on a regular basis during normal operations, then you may want to consider increasing the size of the MAX\_SGA logical standby parameter, or reduce the number of rows that constitute an eager transaction, by explicitly setting the \_EAGER\_SIZE parameter.

Note: It is for this reason that the best practices state the `_EAGER_SIZE` parameter should be set to 1000 when the standby database is instantiated. A value of 1000 should also prevent pageouts from occurring.

As with operating system paging, the SQL Apply engine pages portions of the “lcr cache” to disk. The page file for the SQL Apply engine is created in a database object that resides in the tablespace designated by the Logical Standby procedure `DBMS_LOGSTDBY_D.SET_TABLESPACE`. The SQL Apply engine creates a `LOBSEGMENT` segment in the designated tablespace with a system-generated name. In the event that the database object exhausts the tablespace of free space, then an ORA-1691 will be reported in the alert log of the SQL Apply instance as follows;

```
Wed Apr 23 10:25:16 2003
ORA-1691: unable to extend lobsegment SYSTEM.SYS_LOB0000006304C00008$$
by 4096 in tablespace LOGMNR
Wed Apr 23 10:25:16 2003
Errors in file /u01/app/oracle/admin/ins/bdump/ins1_lsp0_3365.trc:
ORA-12801: error signaled in parallel query server P001
ORA-01691: unable to extend lob segment
SYSTEM.SYS_LOB0000006304C00008$$ by 4096 in tablespace LOGMNR
ORA-06512: at "SYS.DBMS_LOB", line 767
ORA-06512: at line 1
Wed Apr 23 10:25:16 2003
logminer process death detected, exiting logical standby
```

If the ORA-1691 error is reported in the alert log, then you may want to consider one or all of the following

- allocating more space to the specified tablespace
- increase the size of the `MAX_SGA` logical standby parameter
- reduce the number of rows that constitute an eager transaction, by explicitly setting the `_EAGER_SIZE` parameter.

## Troubleshooting Long Running Transactions

One of the primary causes for long running transactions in a SQL Apply environment is because of Full Table Scans. Additionally though, long running transactions could be the result of DDL operations being replicated to the standby database, such as the creation or rebuild of an index.

### Identifying Long Running Transactions

If the SQL Apply engine is executing a single SQL statement for an extended period, then a warning message is reported in the alert log of the SQL Apply instance as follows;

```
Mon Feb 17 14:40:15 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0016.007.000017b6 cscn = 1550349, message# = 28, slavid = 1
knacrb: no offending session found (not ITL pressure)
```

Note the following:

- This warning is similar to the warning message seen with ITL pressure with the exception of the last line beginning “knacrb”. This final line indicates a potential for a Full Table Scan occurring, and the fact that this issue has nothing to do with ITL pressure.
- This warning message is only reported if a single statement takes in excess of 30 seconds to execute.

It may not be possible to determine the SQL statement being executed by the long running statement, but the following SQL may help in identifying the database objects that the SQL Apply slave is operating on.

```
SQL> SELECT SAS.SERVER_ID
2      , SS.OWNER
3      , SS.OBJECT_NAME
4      , SS.STATISTIC_NAME
5      , SS.VALUE
6 FROM V$SEGMENT_STATISTICS SS
7      , V$LOCK L
8      , V$STREAMS_APPLY_SERVER SAS
9 WHERE SAS.SERVER_ID = &SLAVE_ID
10      AND L.SID = SAS.SID
11      AND L.TYPE = 'TM'
12      AND SS.OBJ# = L.ID1;
```

Additionally, the following SQL can be executed to identify SQL statement that have resulted in a large number of disk reads being issued per execution.

```
SQL> SELECT SUBSTR(SQL_TEXT,1,40)
2      , DISK_READS
3      , EXECUTIONS
4      , DISK_READS/EXECUTIONS
5      , HASH_VALUE
6      , ADDRESS
7 FROM V$SQLAREA
8 WHERE DISK_READS/GREATEST(EXECUTIONS,1) > 1
9      AND ROWNUM < 10
10 ORDER BY DISK_READS/GREATEST(EXECUTIONS,1) DESC
```

It is recommended that all tables have primary key constraints defined upon them, which automatically means that the column is defined as NOT NULL. For any table where a Primary Key constraint cannot be defined, then an index should be defined on an appropriate column that is defined as NOT NULL. If a suitable column does not exist on the table, then the table should be reviewed and if possible skipped by the SQL Apply engine.

To skip all DML issued against the FTS table, assuming a schema name of SCOTT, issue the following commands:

1. Stop the SQL Apply engine

```
SQL> alter database stop logical standby apply;
Database altered.
```
2. Configure the skip operation for the SCOTT.FTS table for all DML operations.

```
SQL> execute dbms_logstdby.skip('DML','SCOTT','FTS');
PL/SQL procedure successfully completed.
```

### 3. Restart the SQL Apply engine

```
SQL> alter database start logical standby apply;
Database altered
```

## Troubleshooting ITL pressure

Previously in the “[SQL Apply Best Practices](#)” section on page 14, we talked about a schema best practice that relates to interested transaction list (ITL) pressure being exerted on a database object.

ITL pressure is reported in the alert log of the SQL Apply instance as follows;

```
Tue Apr 22 15:50:42 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0006.005.000029fa cscn = 2152982, message# = 2, slaveid = 17
```

## Real Time Analysis

The message above is reporting that the SQL Apply process (slaveid) #17 has not made any progress in the last 30 seconds. To determine the SQL statement being issued by the Apply process, issue the following statement:

```
SQL> select sa.sql_text
2   from v$sqlarea sa
3      , v$session s
4      , v$streams_apply_server sas
5  where sas.server_id = &slaveid
6      and s.sid = sas.sid
7*   and sa.address = s.sql_address
```

```
SQL_TEXT
-----
insert into "APP"."LOAD_TAB_1" p("PK","TEXT") values (:1,:2)
```

As alternative method to identifying ITL pressure, is to execute the following query. Any session that has a request value of 4 on a TX lock, is waiting for an ITL to become available.

```
SQL> select sid,type,id1,id2,lmode,request
2   from v$lock
3  where type = 'TX'
```

SID	TY	ID1	ID2	LMODE	REQUEST
8	TX	327688	48	6	0
10	TX	327688	48	0	4

In this example, SID 10 is waiting for the TX lock held by SID 8.

## Post Incident Review

Pressure for a segment’s interested transaction list (ITL) is unlikely to last for any extended period. In addition, pressure that lasts for less than 30 seconds will not be

reported in the standby databases alert log. Therefore, to be able determine which objects have been subjected to ITL pressure the following statement can be issued.

```
SQL> select segment_owner, segment_name, segment_type
2   from v$segment_statistics
3   where statistic_name = 'ITL waits'
4         and value > 0
5*  order by value
```

This statement will report all database segments that have had ITL pressure at some time since the instance was last started.

Note: This SQL statement is not limited to a Logical Standby environment, and is applicable to any Oracle database.

### Resolving ITL Pressure

In order to increase the INITRANS for the particular database object, it is necessary to first stop the SQL Apply engine. The following example shows the necessary steps to increase the INITRANS for table "load\_tab\_1" in the schema "app"

1. Stop the SQL Apply engine

```
SQL> alter database stop logical standby apply;
Database altered.
```
2. Temporarily bypass the database guard

```
SQL> execute dbms_logstdby.guard_bypass_on;
PL/SQL procedure successfully completed.
```
3. Execute the SQL statement on the standby database

```
SQL> alter table app.load_tab_1 initrans <#initrans>;
Table altered
```
4. Reenable the guard

```
SQL> execute dbms_logstdby.guard_bypass_off;
PL/SQL procedure successfully completed
```
5. Restart the SQL Apply engine

```
SQL> alter database start logical standby apply;
Database altered
```

It is also advisable to modify the database object on the primary database, so in the event of a switchover occurring, the error should not occur on the new standby database.

## **CONCLUSION**

Oracle Data Guard SQL Apply addresses the requirements of the business community for an online disaster recovery solution that also allows the database to be simultaneously used for reporting and decision support operations.

This paper has detailed the best practices that will support the successful implementation of a SQL Apply database, including:

- Providing an initial set of parameters for the SQL Apply engine
- Understanding the characteristics of the SQL Apply database
- Choosing the correct consistency model
- Monitoring and Tuning the SQL Apply database to proactively keep the database running at its peak throughput
- The procedures for successful switchover and failover operations.

## APPENDIX A – POPULATING A NEW SCHEMA

If there is a requirement to add a new schema to an existing database, then the schema could be created on the primary database, and the schema will be replicated to the logical standby database. However, if it is anticipated that the new schema will take a significant amount of time to be created, and will also involve the population of tables with millions of rows possibly from text files, then it is probably desirable for this schema to be populated in a more economical manner.

1. The following steps will allow a new schema to be created and populated in the shortest amount of time. However, the steps should be reviewed in advance, and appropriate security measures established to not allow the data to diverge. On primary - create new user  

```
CREATE USER NEW_USER IDENTIFIED BY PASSWORD;
```

2. On primary - archive current logs

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

3. On standby - make sure logs generated in Step #2 are applied. Additionally, the alert log of the logical standby database will report the creation of the new\_user schema.

```
Tue Jun 24 13:49:06 2003  
LOGSTDBY event: ORA-16204: DDL successfully applied  
LOGSTDBY stmt: create user new_user identified by VALUES  
'D940C9DA9EEF7828'
```

4. On standby - stop apply services

```
ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

5. On standby – configure the SQL Apply engine to skip all DDL & DML operations issued against the “new\_user” schema.

```
DBMS_LOGSTDBY.SKIP('SCHEMA_DDL', 'NEW_USER', '%');  
DBMS_LOGSTDBY.SKIP('DML', 'NEW_USER', '%');
```

6. On standby - ensure that the default transaction consistency (FULL) is chosen.

```
DBMS_LOGSTDBY.APPLY_SET('TRANSACTION_CONSISTENCY', 'FULL');
```

7. On standby – Reduce the level of the SQL Apply GUARD to STANDBY.

```
ALTER DATABASE GUARD STANDBY;
```

8. On standby - start the SQL Apply services

```
ALTER DATABASE START LOGICAL STANDBY APPLY;
```

9. Create the new schema on both the Primary & Standby

- a. On primary - connect as the new user created in Step #1
- b. On primary - create tables, do SQL-Load, create indexes
- c. On standby - connect as the new user created in Step #1
- d. On standby - create the same tables, do the same SQL-Load, create the same indexes (as in Step #10)

At this point, make sure that the tables and data on the primary and standby are identical - i.e. the data created in Step #10 should not be independently modified on the primary

- On primary - archive the current and next online redo logs

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
SELECT THREAD#, SEQUENCE#
FROM V$LOG
WHERE STATUS = 'CURRENT'
```

```

  THREAD#  SEQUENCE#
  -----  -
         1           68
         2           72
```

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

- On standby - make sure logs generated in Step #10 are applied. Use the SQL statement “applied\_scn” on page 25 to ensure that the archived redo logs currently being applied are those reported above, or are greater than those reported above.

```

THR#   SEQ#  FIRST_TIME          FIRST_CHNG NEXT_TIME          NEXT_CHNG
-----  -
  1     68  03/06/24 14:16:39    278354 03/06/24 14:17:13    278371
  2     72  03/06/24 14:16:05    278357 03/06/24 14:16:39    278374
```

Do not proceed until the SQL Apply engine has applied the correct log files.

- On standby - stop apply services

```
ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

- On standby – remove the SQL Apply engine skip commands.

```
DBMS_LOGSTDBY.UNSKIP ('SCHEMA_DDL', 'NEW_USER', '%');
DBMS_LOGSTDBY.UNSKIP ('DML', 'NEW_USER', '%');
```

- On standby – reset the transaction consistency to the original settings.

```
DBMS_LOGSTDBY.APPLY_SET ('TRANSACTION_CONSISTENCY', <LEVEL>);
```

- On standby – reset the level of the SQL Apply GUARD to ALL.

```
ALTER DATABASE GUARD ALL;
```

- On standby - start the SQL Apply services

```
ALTER DATABASE START LOGICAL STANDBY APPLY;
```

- On primary - grant access rights to the table, and open up tables to users.

A similar set of steps could be used to add a new tablespace to the database. The notable differences would be:

- There is no "create new user" step
- The skip statement to be used on the standby is `dbms_logstdby.skip('TABLESPACE')`
- In step #9, do a create tablespace, besides creating the table, doing the SQL-Load and creating the indexes. The tablespace created on the

standby must have the same name as that of the tablespace created on the primary.

The above steps (for new users or new tablespaces) have to be done under highly controlled environments to make sure that the data in the logical standby database is fully synchronized with that in the primary. For example, if the data that is imported in the primary database is somehow modified by an application on the primary database before the skip statements are removed on the logical standby database, the data will not be consistent between the two databases, and may lead to "ORA-01403 no data found" errors in the future on the logical standby, which will require you to reinstantiate the offending database object(s).

## **APPENDIX B – TEST ENVIRONMENT**

### **Database Server Hardware and Software**

The production system consists of a 2-node Oracle Real Application Clusters (RAC) with two HP N-class servers. Each node has 8 CPU's, 16GB RAM with the database files striped across HP va7400 storage with 45 spindles over two controllers using HP's Logical Volume manager (LVM). The HP va7400 storage arrays were configured with RAID 1+0.

The standby system was configured identically to the primary system; the logical standby process (LSP0) was run on a single node of the standby system in the Real Application Clusters system.

#### **Hardware**

2-node Primary RAC cluster, 2-node Standby RAC cluster. Each node in the RAC Primary and the RAC standby cluster has the following configuration:

8 x 440Mhz CPU's per node

16GB memory per node

HP StorageWorks Virtual Array va7100 for file systems and archive destinations

HP StorageWorks Virtual Array va7400 for the database files

HP HyperFabric cluster interconnect

#### **Software**

HP-UX v11.11 64-bit.

HP ServiceGuard eRAC edition v11.13

Oracle Database

Enterprise Edition Release 9.2.0.4.0 – Production,

with the Partitioning and Real Application Clusters options



Oracle9i Data Guard: SQL Apply Best Practices

September 2003

Authors: Andrew Babb

Contributing Authors: Lawrence To, Raymond Dutcher, Ashish Ray, Joydip Kundu, Ray Guzman, Nimar Arora, Lik Wong

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

[www.oracle.com](http://www.oracle.com)

Oracle is a registered trademark of Oracle Corporation. Various product and service names referenced herein may be trademarks of Oracle Corporation. All other product and service names mentioned may be trademarks of their respective owners.

Copyright © 2003 Oracle Corporation

All rights reserved.