1. ORACLE STREAMS INTRODUCTION

Oracle streams enables the sharing of data and events in a data stream either within a database, or from one database to another. This feature provides a lot of functionality and flexibility for capturing and managing events, and sharing the events with other databases and applications.

Oracle streams system consists of three main processes:

- Capture changes to database objects from redo log. These changes are placed in a queue.
- Propagate changes from queue at source database to queue at destination databases.
- Apply to retrieve changes from the destination queue area and apply to the database.

REPLICATION using ORACLE STREAMS

Oracle Streams provides the elements to capture, staging, and consumption of events within the Oracle database which implement flexible replication systems. The capture mechanism extracts data and structure changes from the redo log and publishes these updates to the staging area. These changes can be propagated to one or more remote staging areas where they can be applied at the destination site. Both the source and destination databases are fully available to end-users for reading and writing during any replication activity.

Before setting up the streams replication environment, set the following parameters to the values indicated:

global_names=true

It must be set to true at each database that is participating in the replication environment.

job_queue_processes=20

It should be set to the same value as the maximum number of jobs that can run simultaneously plus one.

aq_tm_processes=1

Setting it to 1 or more starts the specified number of queue monitor processes.

logmnr_max_persistent_sessions=10

This parameter specifies the maximum number of pesistent LOGMINER mining sessions.

parallel_max_servers=10

parallel_min_servers=2

Specify a value for this parameter to ensure that there are enough parallel execution servers.

shared_pool_size=150M

It has to be set to at least 100MB.

open_links=4

Specifies the maximum number of concurrent open connections to remote databases.

log_parallelism=1

This parameter must be set to 1 at each database that captures events.

Furthermore, any database where changes are captured must be running in ARCHIVELOG mode and make sure that the network is configured so that all databases in the Streams environment can communicate with each other (modify the tnsnames.ora file).

Performance tips:

1. Increase the shared_pool_size.

   Each capture process requires 10Mb of shared pool. In addition to the memory required for the capture process, there is an in-memory buffer queue which holds all of the logical change records (LCRs) for streams requires memory from the shared pool. The buffer queue memory requirement is limited to 10% of the memory allocated with the shared_pool_size parameter.

   When the buffer queue memory threshold is exceeded, LCRs will spill-over to disk and continue to spill to disk until all transactions have been consumed by all down-streams sites. When this spill-over occurs, streams performance is impacted.

   As of 9.2.0.5, the percentage of shared_pool_size can be modified with the hidden parameter _first_spare_parameter.

   In Oracle 10g, this parameter will have no effect as Streams memory can be explicitly specified with the streams_pool_size initialization parameter.

2. For bi-directional replication, configure two queues to minimize spill-over from buffer queue to disk.

   When configuring bi-directional replication, another technique for minimizing spill-over to disk is to configure two queues at each site: one queue for capturing local changes, and a second queue to hold the changes from other sites.

3. To reduce the propagation lag, set the hidden initialization parameter _job_queue_interval to 1.

2. STREAMS SETUP

Replication using Streams overview:

The first step is setting up the user and creating queues and databases links for the streams replication environment.

The capture process uses Log Miner to capture changes that are recorded in the redo log. LogMiner tables include data dictionary tables and temporary tables. By default, all LogMiner tables are created to use the SYSTEM tablespace, but it may not have enough space to accommodate the LogMiner tables. Therefore, creating an alternate tablespace for them is recommended.

To manage a streams environment, either create a new user with the appropriate privileges (these privileges enable him to manage queues, create rule sets, create rules and monitor the streams environment) or grant these privileges to an existing user.

The next step is to configure the streams environment to share information between databases.

To use one or more apply processes to apply LCRs captured by a captured process, it is necessary to enable supplemental logging at the source database. Supplemental logging places additional data into a redo log whenever an update operation is performed. The capture process captures this additional information and places it in LCRs.
Once the capture process is created, the sites must be configured to apply those changes.
Also, the propagation job must be configured and scheduled to propagate events (DML and DDL changes) in the schema from the source queue to the destination queue.

Note: *If you share a sequence at multiple databases, sequence values used for individual rows at these databases may vary. Also, changes to actual sequence values are not captured. For example, if a user references a NEXTVAL or sets the sequence, the capture process does not capture changes resulting from these operations.*

2.1 ADDING NEW DATABASE TO A MULTIPLE SOURCE ENVIRONMENT

Note: *Make sure the "strmuser" schema is created before starting with the script.*

```
-- at site "new site"
-- create alternate tablespace for the LogMiner tables

-- connect as sysdba
connect /as sysdba
CREATE TABLESPACE logmntrs DATAFILE '<location>/logmnrts01.dbf'
SIZE 1M AUTOEXTEND ON NEXT 2M MAXSIZE 500m;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmntrs');

-- create the streams administrator user
CREATE USER strmadmin IDENTIFIED BY strmadmin;

-- grant privileges
GRANT DBA, AQ_ADMINISTRATOR_ROLE to STRMADMIN;
GRANT SELECT ANY DICTIONARY TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_aq TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_aqadm TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_flashback TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_streams_adm TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_capture_adm TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_apply_adm TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_rule_adm TO STRMADMIN;
GRANT EXECUTE ON sys.dbms_propagation_adm TO STRMADMIN;
GRANT SELECT_CATALOG_ROLE TO STRMADMIN;


BEGIN
 DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege     => 'ENQUEUE_ANY',
```

```
   grantee       => 'STRMADMIN',
   admin_option  => FALSE);
END;
/
BEGIN
 DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege     => 'DEQUEUE_ANY',
   grantee       => 'STRMADMIN',
   admin_option  => FALSE);
END;
/
BEGIN
 DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE(
   privilege     => 'MANAGE_ANY',
   grantee       => 'STRMADMIN',
   admin_option  => TRUE);
END;
/
BEGIN
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
END;
/
BEGIN
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.CREATE_ANY_RULE_SET,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.ALTER_ANY_RULE_SET,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.EXECUTE_ANY_RULE_SET,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.CREATE_ANY_RULE,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.ALTER_ANY_RULE,
    grantee       => 'STRMADMIN',
    grant_option  => TRUE);
 DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege     => DBMS_RULE_ADM.EXECUTE_ANY_RULE,
    grantee       => 'STRMADMIN',
```

```
    grant_option  => TRUE);
END;
/

-- create the streams queues at site "new site"
-- connect to site "new site" as user strmadmin
CONNECT strmadmin/strmadmin@<new_site>
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
     queue_table => 'STREAMS_QUEUE_TABLE_CA',
     queue_name => 'STREAMS_QUEUE_CA',
     queue_user => 'STRMADMIN');
END;
/
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
     queue_table => 'STREAMS_QUEUE_TABLE_AP',
     queue_name => 'STREAMS_QUEUE_AP',
     queue_user => 'STRMADMIN');
END;
/


-- create a database link to the source database in the Streams
environment
CREATE DATABASE LINK rls1.cern.ch CONNECT TO strmadmin IDENTIFIED
BY strmadmin USING 'rls1.cern.ch';
```

```
-- at site rls1.cern.ch
-- create a database link to the new destination
database
-- connect to site rls1.cern.ch as user strmadmin
connect strmadmin/strmadmin@rls1.cern.ch
CREATE DATABASE LINK <link_name_new_site> connect to
STRMADMIN    identified    by    strmadmin    using
<service_name_new_site>;
```

```
-- at site "new site"
-- create the capture process
-- connect to site "new site" as user strmadmin
connect strmadmin/strmadmin@<new_site>
-- add capture rules for the schema (the new database is a source
database)
BEGIN
 DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name => 'STRMUSER',
   streams_type => 'CAPTURE',
   streams_name => 'STRMADMIN_CAPTURE',
   queue_name => 'STRMADMIN.STREAMS_QUEUE_CA',
   include_dml => true,
   include_ddl => true,
   source_database => <new_site(database global name)>);
END;
```

```
/

DECLARE
  iscn  NUMBER;   -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@RLS1.CERN.CH(
    source_schema_name     => 'STRMUSER',
    source_database_name   => <new_site(database global name)>,
    instantiation_scn      => &iscn);
END;
/


-- create the apply process for each source database
-- add apply rules for the schema at the destination database
BEGIN
 DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name => 'STRMUSER',
   streams_type => 'APPLY',
   streams_name => 'STRMADMIN_APPLY',
   queue_name => 'STRMADMIN.STREAMS_QUEUE_AP',
   include_dml => true,
   include_ddl => true,
   source_database => <new_site(database global name)>);
END;
/

-- specify an apply user
BEGIN
 DBMS_APPLY_ADM.ALTER_APPLY(
   apply_name => 'STRMADMIN_APPLY',
   apply_user => 'STRMUSER');
END;
/

-- grant the user execute privilege on the apply process rule set
DECLARE
   rs_name  VARCHAR2(64);   -- variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='STRMADMIN_APPLY';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'STRMUSER');
END;
/

-- grant the appropriate privileges to perform DDL changes to the
apply user
```

```
-- at site rls1.cern.ch
```

```
-- add apply rules to apply the changes of new
database
-- connect to site rls1.cern.ch as user strmadmin
connect strmadmin/strmadmin@rls1.cern.ch
BEGIN
 DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
   schema_name => 'STRMUSER',
   streams_type => 'APPLY',
   streams_name => 'STRMADMIN_APPLY_<NEW>',
   queue_name => 'STRMADMIN.STREAMS_QUEUE_AP',
   include_dml => true,
   include_ddl => true,
   source_database =>
              <new_site(database global name)>);
END;
/
```

```
-- at site "new site"
-- add propagation rules for the schema at "new site"
-- connect to site "new site" as user strmadmin
connect strmadmin/strmadmin@<new site>
BEGIN
 DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
   schema_name => 'STRMUSER',
   streams_name => 'STRMADMIN_PROPAGATE',
   source_queue_name => 'STRMADMIN.STREAMS_QUEUE_CA',
   destination_queue_name =>

'STRMADMIN.STREAMS_QUEUE_AP@RLS1.CERN.CH',
   include_dml => true,
   include_ddl => true,
   source_database => <new_site(database global name)>);
END;
/
```

```
-- at site rls1.cern.ch
--  configure propagation from source to new
destination database
-- connect to site rls1.cern.ch as user strmadmin
connect strmadmin/strmadmin@rls1.cern.ch
BEGIN
 DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
   schema_name => 'STRMUSER',
   streams_name => 'STRMADMIN_PROPAGATE_<NEW>',
   source_queue_name =>
          'STRMADMIN.STREAMS_QUEUE_CA',
   destination_queue_name =>
          'STRMADMIN.STREAMS_QUEUE_AP@<new_site>,
include_dml => true,
   include_ddl => true,
   source_database => 'RLS1.CERN.CH');
END;
/
```

In different window, export the schema at rls1.cern.ch that will be instantiated at testadd.cern.ch.

```
exp strmuser/strmuser@RLS1.CERN.CH FILE=schema_strmuser.dmp
GRANTS=y ROWS=n LOG=exportSchema.log  OBJECT_CONSISTENT=y
INDEXES=y STATISTICS=none
```

Then, transfer the export dump file schemas.dmp to the destination database and import it to instantiate the tables at the destination database.

```
imp strmuser/strmuser@<new_site> FILE=schema_strmuser.dmp IGNORE=y
ROWS=n COMMIT=y LOG=importSchema.log STREAMS_INSTANTIATION=y
```

Before starting the capture and apply processes, check that the source site (rls1.cern.ch) complete the appropriate steps (shadow code).

```
-- at site "new site"
-- connect to "new site" as user strmadmin
connect strmadmin/strmadmin@<new_site>

-- start the apply process
BEGIN
 DBMS_APPLY_ADM.SET_PARAMETER(
   apply_name => 'STRMADMIN_APPLY',
   parameter => 'DISABLE_ON_ERROR',
   value => 'N');
END;
/
BEGIN
 DBMS_APPLY_ADM.START_APPLY(
   apply_name => 'STRMADMIN_APPLY');
END;
/
```

```
-- at site rls1.cern.ch
-- connect to site rls1.cern.ch as user strmadmin
connect strmadmin/strmadmin@rls1.cern.ch

-- start the apply process
BEGIN
 DBMS_APPLY_ADM.SET_PARAMETER(
   apply_name => 'STRMADMIN_APPLY_<NEW>',
   parameter => 'DISABLE_ON_ERROR',
   value => 'N');
END;
/
BEGIN
 DBMS_APPLY_ADM.START_APPLY(
   apply_name => 'STRMADMIN_APPLY_<NEW>');
END;
/
```

```
-- at site "new site"
-- connect to "new site" as user strmadmin
connect strmadmin/strmadmin@<new_site>

-- start capture process
BEGIN
 DBMS_CAPTURE_ADM.START_CAPTURE(
   capture_name => 'STRMADMIN_CAPTURE');
END;
/
```

2.2 DISABLING REPLICATION

To disable the replication between the two databases, the capture process must be stopped and the propagation must be disabled, both at source database and then the apply process must be stopped at destination database.

```
-- connect as strmadmin user
-- stop an existing capture process at source database
BEGIN
  DBMS_CAPTURE_ADM.STOP_CAPTURE(
    capture_name => 'STRMADMIN_CAPTURE_CA');
END;
/
-- stop a propagation job at source database
BEGIN
  DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'STRMADMIN.STREAMS_QUEUE',
    destination => <destination_database_link>);
END;
/
-- stop an existing apply process at destination database
BEGIN
  DBMS_APPLY_ADM.STOP_APPLY(
    apply_name => 'STRMADMIN_APPLY');
END;
/

-- stop queues
BEGIN
  DBMS_AQADM.STOP_QUEUE(queue_name => 'STREAMS_QUEUE_CA');
  DBMS_AQADM.STOP_QUEUE(queue_name => 'STREAMS_QUEUE_AP');
END;
/
```

2.3 RE-ENABLING REPLICATION

Following the next steps to enable the replication: restarting the apply process at destination database, enabling the propagation at source database and, finally, restarting the capture process at source database too.

```
-- connect as strmadmin user
-- start queues
BEGIN
  DBMS_AQADM.START_QUEUE(queue_name => 'STREAMS_QUEUE_CA');
  DBMS_AQADM.START_QUEUE(queue_name => 'STREAMS_QUEUE_AP');
END;
/

-- start the apply process
BEGIN
 DBMS_APPLY_ADM.START_APPLY(
   apply_name => 'STRMADMIN_APPLY');
END;
/
-- enable propagation at destination database
BEGIN
  DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'STRMADMIN.STREAMS_QUEUE',
    destination => <destination_database_link>);
END;
/
-- start the capture process
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'strmadmin_capture');
END;
/
```

2.4 RECOVERING FROM FAILURES

Streams replicas can be open to read/write operations at all times. If a primary database fails, services that were using the failed database can connect to Streams replicas, assuming all data is replicated.

It is important to ensure that propagation continues to function after a failure of a database. A propagation job will retry the database link after a failure until the connection is re-established.

If the capture process was running at the time of the failure, there is no need to restart the capture process. The same happens with the apply process, it will automatically return to the state it was in at the time of the failure.

For events to be propagated from a source queue, a propagation job must run on the instance owning the source queue. If the event cannot be propagated to a specific destination queue for some reason (such as a network problem), the event will remain in the source queue until the problem will be solved. Then the captured event will be propagated to the destination queue and the apply process will apply the change at destination database.

3. REFERENCES

[1] Oracle Streams Replication Administrator's Guide 10g Release 1 (10.1)
http://download-west.oracle.com/docs/cd/B13789_01/server.101/b10728/toc.htm

[2] Oracle Streams Concepts and Administration 10g Release 1 (10.1)
http://download-west.oracle.com/docs/cd/B13789_01/server.101/b10727/toc.htm

[3] Database Administrator's Guide
http://download-uk.oracle.com/docs/cd/A91202_01/901_doc/server.901/a90117/toc.htm

[4] PL/SQL User's Guide and Reference
http://download-uk.oracle.com/docs/cd/A91202_01/901_doc/appdev.901/a89856/toc.htm

[5] Oracle Streams--Simplifying Information Sharing in Oracle Database 10g - Patricia McElroy
http://download.oracle.com/owsf_2003/40208_McElroy.doc
http://download.oracle.com/owsf_2003/40208_McElroy.ppt