

# **Advanced master-to-master replication in Oracle**

**A Technical White Paper**

**By**

**David Gornshtein**

**Boris Tamarkin**

**[WisdomForce Technologies, Inc](http://www.wisdomforce.com)**

**<http://www.wisdomforce.com>**

**Revision 1.1**

**Note\*:** The latest copy of this document is available at [http://www.wisdomforce.com/dweb/resources/docs/advanced\\_replication.pdf](http://www.wisdomforce.com/dweb/resources/docs/advanced_replication.pdf)

*This topic contains hands-on for intermediate to advanced level Oracle DBA.*

|   |    |
|---|----|
| Oracle advanced replication limitation .....                        | 2  |
| Approaches in advanced replication .....                            | 3  |
| Hands on session.....   | 4  |
| Monitoring and troubleshooting of Oracle advanced replication. .... | 11 |
| Final .....   | 12 |

## Oracle advanced replication limitation

When Oracle advanced replication is used, it always good idea to remember its limitations. Oracle advanced replication

- has error prone administration.
- has problems with high transaction rates more then 50 - 100 transactions per second, depending on platform or machine .
- not resistant to schema changes.
- has serious performance overhead on both sides. .
- significant part of replicated schema on non-primary master sites might bee required to rebuild in case of failure.

Another well-known issue is that sequences could not be replicated. Although problem with sequences has no solution in Oracle advanced replication, you still can, for instance, create sequences with the same name on both instances, but with large gap for "START WITH" and "MINVALUE" between these sequences. For example if you have sequence NonReplicatedSequence defined as

```
CREATE SEQUENCE NonReplicatedSequence MINVALUE 10000 START WITH 10000  
INCREMENT BY 1 CACHE 20 NOCYCLE;
```

On the first master instance, you may define it on the second master instance as

```
CREATE SEQUENCE NonReplicatedSequence MINVALUE 50000000 START WITH  
50000000 INCREMENT BY 1 CACHE 20 NOCYCLE;
```

*\*Note: We received a lot of feedback on first draft of this article from many readers around the world. We will discuss an additional available strategy of using Oracle sequences in the Oracle master-to-master replication environment. This strategy was originally proposed by Thomas Kyte from Oracle in his AskTom forum on Oracle Technet.*

The idea is in having several sequences, created as in example below for three master sites:

```
create sequence s start with 1 increment by 3; <=== at site 1
create sequence s start with 2 increment by 3; <=== at site 2
create sequence s start with 3 increment by 3; <=== at site 3
```

In this case key values will be non intersected and almost sequential.

In our opinion this particular idea has a lot of disadvantages comparing to creating several different ranges with large gap because of following reasons:

1. The sequence definitions are depended on number of master sites. If you are going to add a new site, then sequences on all other master sites needed to be recreated. Otherwise, sequences should be always created with bigger gap than actually need for it. For instance, if you have three master sites, use “increment by 6”, with option to add 3 additional sites without recreate sequences. However, in this case there will be gaps between each triple of keys, until three additional sites will be added.
2. Gaps can not be eliminated by simple update similar to if two dense sequences are used with large gap between starting values.
3. It is almost never happens that master-to-master replication is really symmetric. For instance there is a rare case when same or even similar amount of inserts is performed on each site. Let’s say, you have 3 master sites. Most of insert operations performed on site 1 and sometimes on site 3. However inserts almost never performed on site 2.

The example of the following sequence of key values is 1, 4, 7, 9, 10, etc.

I will give simple example, where will be only one replication administrator, and the environment is build for TRUSTED replication. There is no reason to divide responsibilities of replication administrator.

## **Approaches in advanced replication**

There are two approaches in using master-to-master advanced replication. These options are namely SYNCHRONOUS and ASYNCHRONOUS. Each one has its pros and cons.

### **Pros of SYNCHRONOUS:**

- While SYNCHRONOUS approach guarantees full synchronization between two sides with each point in time, it may be a good solution for distributed applications. In distributed environment the validity of data is critical and there is relatively small amount of DML (insert/update/delete) operations performed on both sides. The DML response time is not an issue.
- There is no possibility for collisions (conflicts) happening. Due to the fact that SYNCHRONOUS approach uses two phase commit, it is impossible to change the same data on both sides at the time between each change will be propagated to the target.

### **Cons of SYNCHRONOUS:**

- Due to the fact that SYNCHRONOUS approach uses two-phase commit if single side is not available the data on other side may not be updated, so this method could not be used for high availability.
- Have serious overhead on DML response time on both sides.

### **Pros of ASYNCHRONOUS:**

- Using change queues and materialized queues snapshot logs to store changes and therefore even if single side is not available during some period of time the remained side may be used. Moreover, after detached side will be available again, data will be synchronized [last statement depends on purge period, the purge period constraint depends active master database via to be filled by not propagated changes.]

### **Cons of ASYNCHRONOUS:**

- The main disadvantage is a possibility of DML conflicts. When conflict happened, then same data has be changed in two (or more) database in the interval between change propagation to other peer databases. There are several application and database specific techniques to conflict resolution. In the next topics we will look at these techniques, but anyway there is no technique that may resolve all kinds of conflicts to 100%.
- The second disadvantage of ASYNCHRONOUS approach is that replication works via dbms\_job and dbms\_job facility is well known as error prone and not so simple for administration. For example, after 16 unsuccessful attempts, job becomes broken and then you should restart it manually or via OEM if available and if you have XWindows connection to production site and if etc...

### **Hands on session**

The first thing to know prior to start with advanced replication is that you should have primary key (or at least the alternative) for all tables you would like to replicate. For instance, you have two instances test1 and test2. On both instances you have created user REPTTEST00 with default tablespace pool\_data and temporary tablespace temp.

The following statement describes database names and tnsnames aliases of the involved instances:

```
TEST1 = Global Database name of the Master Definition Site
TEST2 = Global Database name of secondary Master Site

test1 = Net alias to the Master Definition Site
test2 = Net alias to the secondary Master Site
```

You may check your global database name by running:

```
select * from global_name
```

Net alias is an alias from \$ORACLE\_HOME/network/admin/tnsnames.ora

**\*Note:**

*Even all oracle documentation that describes advanced replication features, explicitly suggesting that you are using oracle domains eg.test1.world. However it is not really required. So if you have well designed set of database names, you do not need this feature.*

During this session we will avoid to use oracle domains and our Net aliases will be named just test1 and test2 respectively.

In order to work with the oracle advanced replication we should add GLOBAL\_NAMES = true to our pfile (or spfile).

Test1 creates user repadmin in both instances with appropriate permissions

```
CONNECT system/manager@test1
CREATE USER repadmin IDENTIFIED BY repadmin;
ALTER USER repadmin DEFAULT TABLESPACE POOL_DATA;
ALTER USER repadmin TEMPORARY TABLESPACE TEMP;

GRANT connect, resource TO repadmin;
EXECUTE dbms_repcat_admin.grant_admin_any_schema('repadmin');
GRANT comment any table TO repadmin;
GRANT lock any table TO repadmin;

EXECUTE dbms_defer_sys.register_propagator('repadmin');
GRANT execute any procedure TO repadmin;

CREATE PUBLIC DATABASE LINK TEST2 USING 'test2';

CONNECT repadmin/repadmin@test1
CREATE DATABASE LINK TEST2 CONNECT TO repadmin IDENTIFIED BY repadmin;
```

Test2.

```
CONNECT system/manager@test2
CREATE USER repadmin IDENTIFIED BY repadmin;
ALTER USER repadmin DEFAULT TABLESPACE POOL_DATA;
ALTER USER repadmin TEMPORARY TABLESPACE TEMP;

GRANT connect, resource TO repadmin;
EXECUTE dbms_repcat_admin.grant_admin_any_schema('repadmin');
GRANT comment any table TO repadmin;
GRANT lock any table TO repadmin;

EXECUTE dbms_defer_sys.register_propagator('repadmin');
GRANT execute any procedure TO repadmin;

CREATE PUBLIC DATABASE LINK TEST1 USING 'test1';

CONNECT repadmin/repadmin@test2
CREATE DATABASE LINK TEST1 CONNECT TO repadmin IDENTIFIED BY repadmin;
```

Test1 uses ASYNCHRONOUS replication. It tries pushing changes every 10 second, purge changes that not succeed to be propagated during 3 days

```
BEGIN
  dbms_defer_sys.schedule_push(
    destination => 'test2',
    interval => '/*10:Sec*/ sysdate + 10 / (3600*24)',
    next_date => sysdate,
    stop_on_error => FALSE,
    delay_seconds => 0,
    parallelism => 1);
END;
/

BEGIN
  dbms_defer_sys.schedule_purge(
    next_date => sysdate,
    interval => '/*3 : days*/ sysdate + 3',
    delay_seconds => 0,
    rollback_segment => '');
END;
/
```

Test2 uses ASYNCHRONOUS replication. It tries pushing changes each 10 second, purge changes that not succeed to be propagated during 3 days

```
BEGIN
  dbms_defer_sys.schedule_push(
    destination => 'test1',
    interval => '/*10:Sec*/ sysdate + 10 / (3600*24)',
    next_date => sysdate,
    stop_on_error => FALSE,
    delay_seconds => 0,
    parallelism => 1);
END;
/

BEGIN
  dbms_defer_sys.schedule_purge(
    next_date => sysdate,
    interval => '/*3 : days*/ sysdate + 3',
    delay_seconds => 0,
    rollback_segment => '');
END;
/
```

Create master replication group on test1 which is primary master:

```
CONNECT repadmin/repadmin@test1

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPGROUP(
    gname => '"REPTEST"',
    qualifier => '',
    group_comment => '');
END;
/
```

Add secondary master and suspend master activity. If this replication needed to be SYNCHRONOUS, then just change propagation\_mode from ASYNCHRONOUS to SYNCHRONOUS but be aware that SYNCHRONOUS configuration may not be used for High Availability

```
BEGIN
  DBMS_REPCAT.ADD_MASTER_DATABASE (
    gname => '"REPTEST"',
    master => 'TEST2',
    use_existing_objects => TRUE,
    copy_rows => TRUE,
    propagation_mode => 'ASYNCHRONOUS');
END;
/

BEGIN
  DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
    gname => '"REPTEST"');
END;
/
```

Let's assume you have tables XXXX with primary key XXXX\_PK YYYY without primary key but with unique constraint YYYY\_UX contains two columns YYYY\_UX\_COL1, YYYY\_UX\_COL2 and additional index YYYY\_IX. Also assume there is a trigger on table XXXX namely XXXX\_TRIGGER that will be replicated as well.

On test1 which is primary master, create master replication objects:

```
CONNECT repadmin/repadmin@test1
```

Now creating master replication object for all tables

```
BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'TABLE',
    oname => '"XXXX"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'TABLE',
    oname => '"YYYY"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/
```

Create master replication object for trigger:

```

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'TRIGGER',
    oname => '"XXXX_TRIGGER"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/

```

Create master replication object for all indexes:

```

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'INDEX',
    oname => '"YYYY_IX"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/

```

```

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'INDEX',
    oname => '"YYYY_UX"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/

```

```

BEGIN
  DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
    gname => '"REPTEST"',
    type => 'INDEX',
    oname => '"XXXX_PK"',
    sname => '"REPTEST00"',
    copy_rows => TRUE,
    use_existing_object => TRUE);
END;
/

```

Define alternative key for table that have no primary key:

```

BEGIN
  DBMS_REPCAT.SET_COLUMNS (
    sname => '"REPTEST00"',
    oname => '"YYYY"',
    column_list => '"YYYY_UX_COL1, YYYY_UX_COL2"');
END;
/

```

Generate replication support for all the tables to be replicated:

```

BEGIN

```

```

DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
  sname => '"REPTEST00"',
  oname => '"XXXX"',
  type => 'TABLE',
  min_communication => TRUE,
  generate_80_compatible => FALSE);
END;
/

BEGIN
  DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
    sname => '"REPTEST00"',
    oname => '"YYYY"',
    type => 'TABLE',
    min_communication => TRUE,
    generate_80_compatible => FALSE);
END;
/

```

Now wait until all objects from primary master will be copied to secondary master. Afterwards replication can be started by issuing running following statement:

```

BEGIN
  DBMS_REPCAT.RESUME_MASTER_ACTIVITY (gname => '"REPTEST"');
END;
/

```

**Warning!**

*Oracle Advanced replication is not resistant to schema changes since very likely that schema change replication will fail with ORA-23474 afterwards. This is one of the worst possible cases. Replication may not continue to work until you drop all non-primary masters and recreate all replication support from the scratch.*

```

23474, 0000, "definition of \"%s\".\"%s\" has changed since generation
of replication support"
*Cause: The current columns in the specified table and their column
types
do not match the columns and column types when replication
support
was last generated.
*Action: Regenerate replication support for the affected table.
All flavors that include the specified table should be
checked
for validity. Types for any UDT columns should also be
checked
for validity.

```

You can try to resolving ORA-23474 by regenerating replication support for invalidated objects on primary master:

```

CONNECT repadmin/repadmin@test1

BEGIN
  DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
    gname => '"REPTEST"');
END;
/

```

Next is regenerate replication support for all affected objects, for instance:

```
BEGIN
  DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
    sname => '"REPTEST00"',
    oname => '"YYYY"',
    type => 'TABLE',
    min_communication => TRUE,
    generate_80_compatible => FALSE);
END;
/
```

Then you may resume replication by issuing the following statement:

```
BEGIN
  DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
    gname => '"REPTEST"');
END;
/
```

The right way to change schema would be using DBMS\_REPCAT.EXECUTE\_DDL.  
Example:

```
BEGIN
  DBMS_REPCAT.EXECUTE_DDL (
    gname => 'REPTEST',
    master_list => 'test1, test2',
    DDL_TEXT => 'alter table REPTTEST00.XXXX add (add_test
number(8,0))');
END;
/
```

If one of the master databases to be removed:

```
BEGIN
  DBMS_REPCAT.REMOVE_MASTER_DATABASES (
    gname => '"REPTEST"',
    master_list => 'test2');
END;
/
```

If one of master objects to be removed:

```
BEGIN
  DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
    gname => '"REPTEST"');
END;
/

BEGIN
  DBMS_REPCAT.DROP_MASTER_REPOBJECT (
    sname => '"REPTEST00"',
    oname => '"YYYY"',
    type => 'TABLE',
    drop_objects => FALSE);
END;
/
```

```

BEGIN
    DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
        gname => 'REPTEST');
END;
/

```

## Monitoring and troubleshooting of Oracle advanced replication.

Oracle advanced replication uses tables with names such as def\$\_xxxx in the schema “system”. User schema “sys” holds several useful views on those tables named defxxxx respectively.

For example, if one of the alternative keys that you set was not unique, then you may receive ORA-01422 error.

All errors received during advanced replication activity and not yet purged can be identified from DEFERROR view.

You may use the following script:

```

set linesize 132
col DESTINATION          for a10
col ERROR_MSG            for a45
col START_TIME           for a21
col DEFERRED_TRAN_ID    for a15
col CALLNO               for 9999

```

```

SELECT deferred_tran_id, callno, SUBSTR (destination, 1, 10)
destination,
       TO_CHAR (start_time, 'HH24:MI:SS DD/MM/YYYY') start_time,
       SUBSTR (error_msg, 1, 45) error_msg
FROM deferror;

```

Example of inappropriate surrogate key usage result:

| DEFERRED_TRAN_I | CALLNO | DESTINATIO | START_TIME          | ERROR_MSG  |
|-----------------|--------|------------|---------------------|--|
| 9.3.865         | 0      | TEST1      | 08:10:09 14/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 2.27.874        | 0      | TEST1      | 08:10:10 14/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 9.33.864        | 0      | TEST1      | 08:10:16 14/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 9.39.1073       | 0      | TEST1      | 22:02:36 18/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 1.15.1107       | 0      | TEST1      | 22:02:48 18/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 5.36.1089       | 0      | TEST1      | 22:02:51 18/03/2004 | ORA-01422:<br>exact fetch returns more than requ |
| 9.30.1074       | 0      | TEST1      | 22:10:58 18/03/2004 | ORA-01422:<br>exact fetch returns more than requ |

The other useful view of this kind is **DEFTRAN**.

Via this view you may see the currently queued transaction. Please note, transaction that are not propagated due to the error such as in the previous example will remain in this queue forever or until purged.

```
select DEFERRED_TRAN_ID, DELIVERY_ORDER, DESTINATION_LIST, START_TIME
from DEFTRAN;
```

```
DEFERRED_TRAN_I DELIVERY_ORDER D START_TIME
-----
9.3.865          4183230 D 14-MAR-04
2.27.874        4183237 D 14-MAR-04
9.33.864        4183406 D 14-MAR-04
9.39.1073       5196085 D 18-MAR-04
1.15.1107       5196094 D 18-MAR-04
5.36.1089       5196100 D 18-MAR-04
9.30.1074       5198042 D 18-MAR-04
```

## Final

You have read the first article in series about Oracle master-to-master advanced replication, with hands on usage example.

Later on we will add more topics about replication. These series will give you pretty complete picture of replication solutions available today on the market.

You will be able to identify all pros and cons of modern log based replication solutions compared to conservative "trigger / per table change log" replication solutions. Please stay tuned.