



Cloning a Schema Using Materialized Views

By Michael S. Abbey

This paper will look at the marriage of a time-consuming concept and a fundamental piece in Oracle's replication technology — materialized views.

Author's Notes: This article was originally published in the UKOUG publication *Scene* and is reprinted with their permission.

The title of this article mentions materialized views. At the start of the cloning exercise discussed herein, we use export and import. After the NOROWS export and following import are complete, the routines shift to using only materialized views—hence the title.

Suppose you are tasked with cloning a schema containing a large number of objects. There are many ways to do it, including the ever-favourite export/import approach; but this article will approach the task using traditional snapshots. The advantage offered by this approach is zero-downtime; yes, you heard that right—zero! There is one step in this whole process that will have problems with tables whose row counts are in excess of 5,000,000 or so rows. I will indicate where that is, and offer some suggestions on how to deal with these prohibitively large tables.

An ideal scenario for using this method of cloning is when you are moving a production schema to another server and, when the move is complete, the one on the original server will be taken out of service. Throughout this article there are a number of technical assumptions made regarding the specific scenario which will be prefixed as such. These assumptions set the stage for some of the code and routines that I use. If they do not work as planned it is because your environment differs from the one that I used.

Assumption: All tables in the *pythian* schema have a primary key constraint in place. We will be building Oracle8/9i type snapshots that require this constraint. These are true primary key constraints, not unique indexes.

If you are unsure whether or not your source schema has the appropriate constraints, try the following queries—if the counts are the same, OK; if they are different, use the third query to find which tables need a primary key definition.

```
SQL> select count(*) from user_tables;
COUNT(*)
-----
109
SQL> select count(*) from user_constraints
2> where constraint_type = 'P';
COUNT(*)
-----
109
SQL> - Suppose the counts are not the same . . .
SQL> select table_name from user_tables
2> minus
3> select table_name from user_constraints
4> where constraint_type = 'P';
TABLE_NAME
-----
BORDER
MAPPING
VECTOR
3 rows selected.
```

The Infrastructure to Support the Exercise

Suppose the schema being moved is called **pythian**, the source server is **masii** and the target server is **beto**. The following SQL statements will provide the appropriate system privileges to build snapshots on masii and beto.

Assumption: An Oracle Net alias is defined called **beto.world**.

```
SQL> grant create materialized view to pythian;
Grant succeeded.
SQL> conn poweruser/yyhau87@beto.world
SQL> grant create materialized view to pythian;
Grant succeeded.
```

The pythian schema on beto needs a database link that will be used to pull data from masii. This is shown next.

Assumption: The *pythian* user has the “create database link” system privilege.

```
SQL> create database link to_masii connect to pythian identified
2 by hihje09 using 'beto.world';
Database link created.
```

Getting Table Definitions from Source to Target

It's now time to build empty tables on **beto**. The easiest way to do this is using export, passing it parameters similar to those shown in the next parameter file.

```
userid=/ owner=pythian indexes=n rows=n file=stub
buffer=5000000 log=stub triggers=n
```

Assumption: The UNIX account running the export/import has an O/S authenticated Oracle account with the appropriate database privileges to export/import any user.

The export file can be used to create the empty tables. Upon the import, primary key constraints will be created as their SQL definitions are in the export file. The import into the pythian schema on beto is run with the following parameter file, provided that you are logged into the target server.

```
userid=/ fromuser=pythian touser=pythian file=stub
buffer=5000000 log=stub_in
```

Preparing the Source Environment for the Move

This is where we begin to implement the snapshot technology. The first step is to create snapshot logs of pythian's tables on masii. While logged in as pythian, the following SQL script will accomplish this task:

```

set echo off feed off ver off pages 0 lines 999 trimsp on
spool logs_create.sql
select 'create snapshot log on '||table_name||';'
  from user_tables;
spool off
set echo on lines 70 feed on
spool logs_create
@logs_create
spool off

```

The snapshot logs are nothing more than traditional Oracle tables. Unlike most tables we work with, these are not manipulated directly. Oracle's replication mechanism works with them in the background. From this point on, transactions interacting with pythian's tables on masii are tracked in the snapshot log for each object, and ready to be forwarded to any dependent snapshots when they are created.

The second step on the master site is to ensure that the instance's job queue is running. This is accomplished by setting one parameter in an Oracle9i's initialization parameter file (affectionately called INIT.ora) or two parameters in Oracle8i or lower as shown here:

```

# Oracle9i and Oracl8x
job_queue_processes = 8
# Just Oracle8i, measured in seconds
job_queue_interval = 60

```

If you are running Oracle8i or later, the first of these two parameters can be modified dynamically via the appropriate *alter system* command. The interval in Oracle8x can only be changed by shutting down and restarting the database.

Building Snapshots in the Target Environment

This is the start of something that is nothing short of magic. Oracle8i and above have a "prebuilt" option that can be utilized when building snapshots. This allows you to create a snapshot using an existing table and, at any future time, drop the snapshot leaving the original table in place. This is the heart of this cloning exercise. The following is a script to build the snapshots on the prebuilt tables on beto:

```

set lines 999 pages 0 trimsp on feed off
spool sn_cre.sql
select 'create snapshot '||table_name||' on prebuilt table '||
chr(10)||'refresh fast start with sysdate next '||chr(10)||
' sysdate+10/1440 '||chr(10)||' as select * from '||
table_name||'@masii.world; '
  from user_tables;
spool off
rem * Sample output . . .
rem * create snapshot MAST_CTS on prebuilt table
rem * refresh fast start with sysdate next
rem * sysdate+10/1440
rem * as select * from MAST_CTS@masii.world;

```

If you have snapshot experience, you may remember placing storage parameters in the SQL you used to create them without the "on prebuilt table" feature. With this feature, the storage parameters have already been defined for the prebuilt tables brought in with import. The output from this code, *sn_cre.sql* is now run as pythian on beto to create the snapshots.

Kick-Starting the Snapshot Refreshes

Normally when snapshots are built, the "start with sysdate" wording triggers the first refresh. When they are constructed with prebuilt tables, you must perform the first refresh manually, using a script similar to the following:

```

set lines 999 pages 0 trimsp on feed off
spool sn_kick.sql
select 'exec dbms_snapshot.refresh ('||''''||
name||''''||','||''''||'C'||''''||') '
  from user_snapshots;
spool off
rem * Sample output . . .
rem * exec dbms_snapshot.refresh ('MAST_CTS', 'C')

```

Next, run *sn_kick.sql* to start the refresh process. Notice that in the call to *dbms_snapshot.refresh*, you are asking for a complete refresh. This is mandatory even though subsequent refreshes will be fast.

"The Envelope Please"

I always find that those three words conjure up visions of excitement and suspense and, in the case of this article, nothing could be closer to the truth. We now have a set of tables on the master (masii) server replicating every 10 minutes to the remote site (beto). Transaction information is stored in the master's snapshot logs and forwarded to the remote. This process goes on and on and on (hopefully unattended) forever. When you decide it is time to make the move, use the following steps:

1. Assemble a script, which eventually can be used to drop all of the snapshot logs on masii.

```

rem * Run on master (masii)
set echo off feed off ver off pages 0 lines 999 trimsp on
spool logs_drop.sql
select 'drop snapshot log on'||name||';'
  from table_name;
spool off
set echo on lines 70 feed on
rem * Sample output . . .
rem * drop snapshot log on MAST_CTS;
rem * drop snapshot log on ADDRESS;

```

2. Assemble a script that can be run to get row counts in the snapshot log tables on the master using a query similar to the following:

```

rem * Run on master (masii)
set feed off pages 0 line s999 trimsp on echo off
spool sl_counts.sql
select 'select count(*) from MLOG$_'||table_name
  from user_tables;
spool off
rem * Sample output . . .
rem * select count(*) from MLOG$_MAST_CTS;
rem * select count(*) from MLOG$_ADDRESS;

```

continued on page 16

- Assemble a script which can be used to perform fast refresh of all the snapshots on the remote site using something like the following:

```
rem * Run on remote (beto)
set feed off pages 0 line s999 trimsp on echo off
spool sr_run.sql
select 'exec dbms_job.run ('||to_char(job)|| ') '
  from user_jobs
  where lower(what) like ('%dbms_refresh.refresh%');
spool off
rem * Sample output . . .
rem * exec dbms_job.run (123)
rem * exec dbms_job.run (124)
rem * exec dbms_job.run (125)
```

- Assemble a script on the remote to drop the snapshots which, when run later, leaves the prebuilt tables intact (the magic of this approach and syntax).

```
rem * Run on remote (beto)
set echo off feed off ver off pages 0 lines 999 trimsp on
spool sns_drop.sql
select 'drop snapshot '||name||';'
  from user_snapshots;
spool off
set echo on lines 70 feed on
rem * Sample output . . .
rem * drop snapshot MAST_CTS;
rem * drop snapshot ADDRESS;
```

- Find yourself a quiet time in the business day, if one exists, and one after the other, run *sr_run.sql* on the remote, followed by *sl_counts.sql* on the master. What this does is to refresh the snapshots manually, then go back to the appropriate snapshot logs on the master and check the row counts.

When these row counts are 0, it's time to run *sns_drop.sql*. Assuming that *sl_counts.sql*, *sr_run.sql*, *logs_drop.sql* and *sns_drop.sql* have been created, Figure 1 illustrates the logic used to ensure the snapshot log(s) are empty, after which the snapshots they feed and the snapshot logs themselves can be dropped.

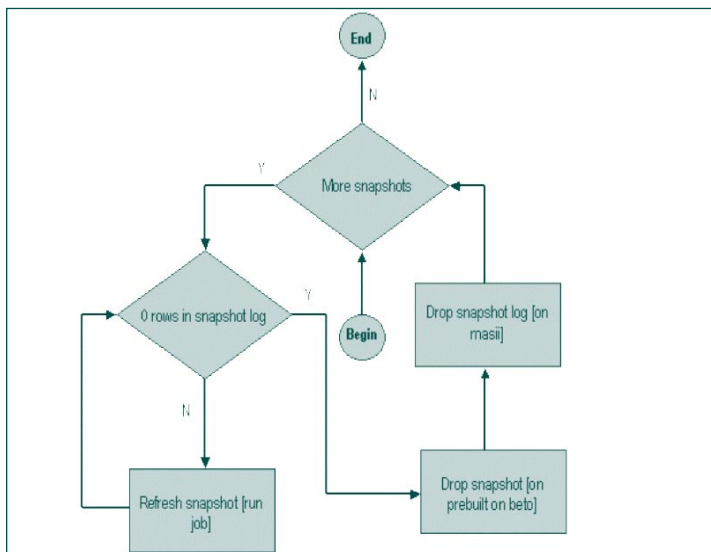


Figure 1: Snapshot Log Logic

This is an iterative process. Rather than do all of the snapshots at once, select a line out of *sr_run.sql* and the corresponding lines out of *sl_counts.sql*, *sns_drop.sql*, *logs_drop.sql* and create one snapshot at a time. When you are finished, you will have a 100% up-to-date copy of the pythian schema on beto rather than masii. It is your responsibility to ensure that the applications using these objects now point to the correct location.

Closing Time

The description above explains the theory behind this trickery (actually simply leveraging the power of snapshots). You could very easily be sitting there saying “If only it were that simple in real life.” We both know it may not be, but you now know the procedures to follow and have some code to try it yourself. Many thanks to Rob Hamel, a fellow techie at the Pythian Group for lighting the spark that led to the birth of this version of this process.



About the Author

Michael S. Abbey is affiliated with The Pythian Group (TPG at www.pythian.com) in Ottawa Canada, which vends remote DBA services throughout the world. Michael is the lead DBA, and presents extensively on the TPG's behalf all over North America and elsewhere when the opportunity presents itself. Michael has co-authored many works in the Oracle Press series, the most popular being a series of the *Beginner's Guides*, written for versions 7, 8, 8i, and most recently 9i. Michael can be reached at abbey@pythian.com and welcomes email on his second most favorite item—Oracle—his most favorite being the works of Gilmour and Waters (aka Pink Floyd).