PAINLESS MASTER TABLE ALTER IN REPLICATION

Arup Nanda, Proligence Inc.

INTRODUCTION

One of the biggest challenges in administration of a snapshot replication environment, also called materialized view replication, is the usual maintenance of the snapshot after a modification of the master table, for example adding columns or modifying the data type of a column. After a column is added to the master table, the only way the newly added column could be replicated to the replication site is by dropping the snapshot and recreating it. If the master table is large, the recreation process may take several hours as it brings all the data over the network. This also needs a large rollback segment both on the master and the replication sites and it may lead to the ORA-1555, *Snapshot Too Old* problem if the table access and rate of change is high. At a certain point, it may be impossible to even build the snapshot by recreating in this manner. At that point the only option would be to do a full export and import of the table and recreating the snapshot by using the PREBUILT table option.

This problem, however, is not present in a multi-master setup. While the snapshot replication site presents numerous advantages in setting up and administration, this lack of ability to alter a master table easily poses real challenges to the DBAs maintaining the environment in relatively trivial tasks like altering a table to add columns or change data types. This article presents a way to achieve these objectives without recreating the snapshot or do a full refresh.

BACKGROUND

This is best explained by an example. In our setup, we have two databases, PROD and REPL, denoting production and the replication databases respectively. All the activity happens on the production site whereas the replication site can be used as a reporting database only (read only snapshot) or as a separate data activity point and the changes are periodically pushed to master site (updatable snapshot). The technique described in this article applies to both situations. The snapshot replication is also called *materialized view replication*. In this example, the terms snapshot and materialized view are used interchangeably. To explain the concepts easily and to allow the reader to simulate these exercises, the setup has been kept simple. The reader is encouraged to try the recommendations mentioned here.

There are several tables under schema ANANDA. This example focuses on a table named TEST1, with about 2 million rows and 4 GB in total size. The table has three columns, COL1 NUMBER (9), COL2 CHAR(1000) and COL3 CHAR(900). The REPL database has a snapshot called TEST1 defined on the same table. The master site has a replication group called TEST1, which has only one object, TEST1. The replication group is owned by the schema REPADMIN. To setup the replication, the script given below can be run at the master site.

```
REM
REM This script is named gen_repl_support.sql
conn ananda/******@prod
create snapshot log on test1
tablespace user_data
with primary key
including new values
/
prompt MView Log Created...
connect repadmin/******@prod
begin
        dbms_repcat.create_master_repgroup(
                gname=>'TEST1',
                qualifier=>'',
                group_comment=>'Test Snapshot');
end;
/
prompt Master RepGroup Created...
```

```
begin
        dbms_repcat.create_master_repobject(
                gname=>'TEST1',
                type=>'TABLE',
                oname=>'TEST1',
                sname=>'ANANDA',
                copy_rows=>TRUE,
                use_existing_object=>TRUE);
end;
/
prompt Master RepObject Created...
begin
        dbms_repcat.generate_replication_support(
                sname=>'ANANDA',
                oname=>'TEST1',
                type=>'TABLE',
                min_communication=>TRUE);
end;
/
prompt Rep Support Generated...
begin
        dbms_repcat.resume_master_activity(
                gname=>'TEST1');
end;
/
prompt Master Activity Resumed
```

On the snapshot site, the snapshot TEST1 is included in a snapshot group called TEST1 owned by schema MVADMIN. It is assumed that there is a public database link from each database to the other with the same name and the global_names parameter in init.ora is set to true.

The goal is the alter the table TEST1 at master site adding a new column called COL4 with CHAR(1) and changing the column COL3 to CHAR(1000).

THE USUAL METHOD

For the purpose of demonstration, first we need to set up the replication environment. The following script contains the statements to create the snapshot site.

```
REM
REM This script is named cr_repl_site.sql
REM
spool cr_repl_site.log
/* Section 1 : Create the Snapshot Group */
connect mvadmin/******@REPL
begin
      dbms_refresh.make(
            name=>'MVADMIN.TEST1',
            list=>'',
            next_date=>sysdate+5/(24*60),
            interval=>'sysdate+5/(24*60)',
            implicit_destroy=>TRUE,
            lax=>FALSE,
            job=>0,
            /* rollback_seg=>'RBS6', Use this if you want to use a particular RBS */
            push_deferred_rpc=>FALSE,
            refresh_after_errors=>TRUE,
            purge_option=>NULL,
            parallelism=>NULL,
            heap_size=>NULL);
end;
```

```
/
prompt Refresh Group Created...
/* Section 2 : Create the Replication Group */
begin
      dbms_repcat.create_snapshot_repgroup(
            gname=>'TEST1',
            master=>'PROD',
            propagation_mode=>'ASYNCHRONOUS');
end;
/
prompt Replication group Created...

/* Section 3 : Create the Snapshot */
connect ananda/******@REPL
create snapshot TEST1
refresh fast
as
select * from ananda.TEST1@PROD
/
prompt Snapshot Created

/* Section 4 : Add the Snapshot to the Group */
begin
      dbms_refresh.add(
            name=>'MVADMIN.TEST1',
            list=>'ANANDA.TEST1',
            lax=>TRUE);
end;
/
prompt Snapshot added

/* Section 5 : Generate Replication Support for the Snapshot */
connect mvadmin/******@REPL
begin
      dbms_repcat.create_snapshot_repobject(
            gname=>'TEST1',
            sname=>'ANANDA',
            oname=>'TEST1',
            type=>'SNAPSHOT',
            min_communication=>TRUE);
end;
/
prompt Snapshot RepObject Created
spool off
```

For the convenience of discussion, the script is split into several sections. The most important part, relevant to this article, is *Section 3, Creating the Snapshot*. This section actually creates the snapshot by getting the data across from the master site over the network. In case of a small table or during periods of light load, this approach might work without filling up the rollback segments or choking the network. However, in large tables, this method of transferring data might fail. Therefore, we have to follow a slightly different approach – using *prebuilt tables*.

In this approach, you have to drop the snapshot TEST1, if it exists and create a table by the same name in the schema ANANDA with the same structure as that table at master site, but no data. This is easily done by running `CREATE TABLE TEST AS SELECT * FROM TEST1@PROD WHERE 1 = 2`. This will work if the table has no LONG columns. In case of LONG, you could create the DDL and run it. After running the script `gen_repl_supp.sql` at the master, you have to export the table from PROD and import into REPL. The table data can be brought over by other methods, too, e.g. by creating a delimited text file from the production database and loading it into replication database using SQL*Loader DIRECT path option. While the data is being moved from the source to the destination system, using export/import, unload/sql*loader, create table as select or any other method, the source table can be open for updates.

Once the table exists at the replication site, the snapshot can be created on the table simply by making a small change in the script cr_repl_site.sql. Change the *Section 3* part of the script as described below.

```
/* Section 3 : Creating the Snapshot */
connect ananda/******@REPL
create snapshot TEST1
on prebuilt table
refresh fast
as
select * from ananda.TEST1@PROD
/
```

Notice the clause ON PREBUILT TABLE. This instructs Oracle that there is a table called TEST1 and that should be used as the segment for the snapshot named TEST1 and it should not create a new segment. The rest of the script in cr_repl_site.sql simply make the snapshot ready for replication.

After running the setup for a while, the table TEST1 was altered as described above – the column COL3 was changed to CHAR(1000) and a new column COL4 CHAR(1) was added. These changes need to be reflected at the snapshot site, too. The Oracle recommended approach is to drop the snapshot at the snapshot site and run the process from beginning again, i.e. drop the table TEST1, create it, import rows, create snapshot on prebuilt table and finally generating replication support. With a large table, this may lead to several problems like running out of rollback segments, taking considerable time and slowing down performance. In our case it took more than 4 hours, including the table alters and of course, the time will vary depending on your exact environment.

THE ALTERNATIVE APPROACH

When the snapshot is created on a table using PREBUILT option, the snapshot simply takes over control of the segment defined for the table. When the snapshot is dropped, the segment is *not* dropped, rather is turns the control into the original table. Since the segment is the same, any data changes occurred during the snapshot operation remains in the segment even after the snapshot is dropped. For example, say, the value of COL2 in the original table for COL1 = 1 was 'A'. After the snapshot creation on the table, the snapshot operation changed the data to 'B' since that was changed to 'B' at the master site. Subsequently the snapshot was dropped and the segment reverted to a table called TEST1. At this point the value of COL2 for COL1 = 1 will *still* be 'B', not 'A'. Thus, data in the segment remains exactly same, as it was the moment before the snapshot is dropped. This fact is exploited in the alternative approach.

Since data remains the same, there is no need to drop and rebuild the table from master. We will use this trick to let the replication setup know that the snapshot was never dropped and so a fast refresh would work. Another little detail to take care of here is the use of the already present snapshot log entries. These entries are needed for the fast refresh. When a snapshot is recreated on a prebuilt table, these entries on the master table are deleted. We will have to store it prior to the deletion and insert it after the snapshot is ready for replication.

DETAILED STEPS

At the master site, the table can be just altered using sql. Logging in as user ANANDA, issue the statements

```
alter table test1 add (col4 char(1);
alter table test1 modify (col3 char(1000));
```

Since DML is still active on the table, you may have to wait until the table can be locked exclusively to add and modify the columns.

At the replication site, we need to stop the replication pull for a while by shutting down the job that does it. Logging in as user MVADMIN, issue

```
select job from user_refresh where rname = 'TEST1';
```

Note the job number. Again, as we assumed in the beginning, the refresh group is named TEST1. Shutdown the job by issuing

```
exec dbms_job.broken(<jobnumber>,TRUE);
commit;
```

It's very important to issue a commit here. You will also have to make sure no current sessions are currently active by this job. Check that by issuing

```
select sid from dba_jobs_running where job = <jobnumber>;
```

If you see any session, wait for it to finish or kill it before proceeding.

Then, logging in as user ANANDA and issue

```
DROP SNAPSHOT TEST1;
```

This drops the snapshot but leaves the table in place. Then issue

```
alter table test1 add (col4 char(1);
alter table test1 modify (col3 char(1000));
```

The rest of the operation has been placed in a script as shown in script `cr_repl_prebuilt.sql` below. Most of the script is the same as in the previous approach. The differences are explained here.

```
REM
REM This script is a called cr_repl_prebuilt.sql
REM
spool cr_repl_prebuilt.log

/* Section 1 : Create the Refresh Group */
connect mvadmin/******@REPL
begin
      dbms_refresh.make(
            name=>'MVADMIN.TEST1',
            list=>'',
            next_date=>sysdate+5/(24*60),
            interval=>'sysdate+5/(24*60)',
            implicit_destroy=>TRUE,
            lax=>FALSE,
            job=>0,
            /* rollback_seg=>'RBS6', Use this if you want to use a particular RBS */
            push_deferred_rpc=>FALSE,
            refresh_after_errors=>TRUE,
            purge_option=>NULL,
            parallelism=>NULL,
            heap_size=>NULL);
end;
/
prompt Refresh Group Created...

/* Section 2 : Create Snapshot Replication Group */
begin
      dbms_repcat.create_snapshot_repgroup(
            gname=>'TEST1',
            master=>'PROD',
            propagation_mode=>'ASYNCHRONOUS');
end;
/
prompt Master RepObject Created...

/* Section 3 : Store the snaphot log entries */
connect ananda/******@REPL
col part_tab_name noprint new_value part_tab_val
select substr('TEST1',1,20) part_tab_name
from dual
/
drop table mlog_bak
```

```
/
create table mlog_bak
as
select * from mlog$_&&part_tab_val.@PROD
/

/* Section 4 : Built the snapshot */
create snapshot ananda.TEST1
on prebuilt table
refresh fast
as
select * from ananda.TEST1@PROD
/
prompt Snapshot Created

/* Section 5 : Add the snapshot to the refresh group */
begin
      dbms_refresh.add(
            name=>'MVADMIN.TEST1',
            list=>'ANANDA.TEST1',
            lax=>TRUE);
end;
/
prompt Snapshot added

/* Section 6 : Build replication support for the group */
connect mvadmin/******@REPL
begin
      dbms_repcat.create_snapshot_repobject(
            gname=>'TEST1',
            sname=>'ANANDA',
            oname=>'TEST1',
            type=>'SNAPSHOT',
            min_communication=>TRUE);
end;
/
prompt Snapshot RepObject Created

/* Section 7 : Restore the snapshot log entries */
connect ananda/******@REPL
insert into mlog$_&&part_tab_val.@PROD
select * from mlog_bak
/
commit
/
prompt Interim Mlog entries Inserted
spool off
```

ANALYSIS

Section 3 is new. When a snapshot is built on an existing table and replication support is enabled on that, Oracle assumes that the snapshot has gone through a complete refresh; so it deletes the snapshot log entries at the master site. This is not acceptable in our situation since the master table has undergone some DML activity and has generated snapshot log entries. Even if there is no DML, there could still be some unapplied snapshot log entries that need to be preserved. Therefore, we move the entries in master table's snapshot log to a temporary table called *mlog_bak*. If the table name is too long, Oracle uses only the first 20 characters of the name to create the name of the snapshot log table. Although in this case the table name is only five letters, to make it generalized, we have used only the first 20 characters and prefixed it to MLOG$_ to get the name of the snapshot log table.

Sections 4 through 6 are the same as in the previous approach.

After the replication support is enabled on the table, the snapshot log entries preserved earlier need to be restored back so that they can aid in fast refresh. This is achieved in Section 7.

After all these steps are executed issue a fast refresh of the snapshot just to make sure that the fast refresh works. Logging in as ANANDA issue

```
execute dbms_snapshot.refresh('TEST1','F')
```

This will do a fast refresh of the snapshot using the snapshot log entries we just restored back. The snapshot is now all set for fast refresh and with the modified table structure.

Finally, you have to re-enable the job broken earlier. Logging in as MVADMIN, issue the following statement. Use the job number obtained before.

```
execute dbms_job.run(<jobnumber>)
```

Total elapsed time, including the table alters, was 10 minutes.

CONCLUSION

As you can see the time to alter the master table was reduced from 4 hours to 10 minutes, a 98% savings in time and more substantial in terms of effort of the DBA. Of course, in your situation it might vary depending on the table size and the network transfer speed. Nevertheless, no matter how fast the network speed is, unless the table is tiny, the time and effort reduction will always be quite substantial in nature.

ABOUT THE AUTHOR

Arup Nanda (arup@proligence.com) has been an Oracle DBA for more than ten years and is the founder of Proligence, a New York area based advanced and specialized Oracle solutions company providing assistance in formulating strategy and direction in Oracle database implementation and management. Arup specializes in performance tuning, replication and backup solutions. He has written several papers for Oracle related journals like SELECT, Oracle Scene, SQLUpdate and presented at several Oracle technical conferences like Atlantic Oracle Training Conference at Washington, DC; Virginia Oracle Technology Conference at Richmond, VA; Oracle Technology Symposium at Stamford, CT and others. An updated version of this article can be found on www.proligence.com after the audience feedback.