Do large deletes or updates the safe way in Oracle

Performing a large amount of DML in a single transaction can be disastrous in an Oracle database. It may run for a long time, creating large amounts of redo and undo, overwhelming a database tuned for smaller transactions. Also, if interrupted, all the work will be rolled back, creating locking and performance issues, sometimes for hours. Even restarting the DB will not fix this, and this has been the cause of much downtime in Oracle shops.

The obvious solutions is to break up the DML into a series of smaller transactions, committing in between each statement. While this works, it takes time to setup, is prone to errors that commit as part of the script, and even if done well, can cause blocks to be changed multiple times, creating much more total redo than would be needed if done as as single statement.

There is a way to get the best of both worlds, using an Oracle package called DBMS_PARALLEL_EXECUTE. This package is fairly new and somewhat confusing in it's documentation, but it works very well. I have written a helper procedure (ent_common.purge_utils.purge_large_table) to hide the complexity and provide a tested, easy implementation for doing DML on large tables.

Theory of Operation

The goal is to perform all the needed DML to a set of blocks rather than searching for a subset of records and changing all the blocks with those records, resulting in some blocks being affected by many changes. The package will perform the subsetting by blocks and will perform commits and checkpoints at appropriate intervals. Commit frequency is determined by the number of records changed. This allows the process to traverse large areas with few or no affected records without adding commit load to the database, but will commit frequently enough to ensure that blocking locks are kept to a minimum. After each commit, the process will sleep to allow the redo stream to catch up and other database operations to use the resources. Tuning the sleep time, chunk size, and commit/checkpoint intervals can have a dramatic impact on the run time of the process and it's load on the database. The defaults are meant to be conservative and favor low impact on the database over fast completion time.

Procedure Usage

```
ent_common.purge_utils.purge_large_table(p_owner VARCHAR2,
                              p_table_name      VARCHAR2,
                              p_sql             VARCHAR2,
                              p_keep_days       PLS_INTEGER DEFAULT 365,
                              p_sleep_time      PLS_INTEGER DEFAULT 2,
                              p_chunk_size      PLS_INTEGER DEFAULT 500,
                              p_commit_interval PLS_INTEGER DEFAULT 1000,
                              p_chkpnt_interval PLS_INTEGER DEFAULT 100,
                              p_max_run_time    PLS_INTEGER DEFAULT 0,
                              p_exit_time       VARCHAR2 DEFAULT NULL,
                              p_log_always      VARCHAR2 DEFAULT 'N');
```

The only mandatory parameters are the first 3, table owner and name and the SQL that is to be run for the DML. This SQL needs to be specially crafted to look like this:

'DELETE /*+ ROWID (RESIDUAL_STAGE) */ FROM RE.RESIDUAL_STAGE WHERE rowid BETWEEN :start_id AND :end_id AND processing_period < sysdate - :keep_days'

Notes about the SQL parameter:

1. Any single quotes in the statement must be escaped by adding an extra single quote before it.
2. The ROWID hint is needed and must have the name of the table in quotes
3. The BETWEEN clause must remain exactly as shown, it is what allows for range operations.
4. The :keep_days var needs to be included to prevent errors when executing, but if you wish to not use it, add it as a throwaway clause like " :keep_days > 0 " (it defaults to 365, so this will always be true)

Notes about the tuning parameters:

1. p_sleep_time - This is the number of seconds to sleep after each commit. This is to allow other database operations to use resources for a time and prevent the DML process from hogging all the resources and impacting DB operations.
2. p_chunk_size - The number of blocks in each processing unit. The entire table will be processed, but only this many blocks at a time. Increasing this speeds the DML process but increases the time for each operation and the chance of other processes being blocked.
3. p_commit_interval - Once this many (or more) records are affected by the DML (regardless of the number of actual blocks processed), a commit will be issued and the sleep will take place. Increasing this has the same risks/benefits as p_chunk_size
4. p_chkpnt_interval - Once this many commits have been made, a checkpoint is performed to avoid any problems with online redo logs not archiving fast enough.

Example:

This will delete all the rows from the RE.RESIDUAL_STAGE table where the processing period is at least 60 days ago.

```
BEGIN
  ent_common.purge_large_table(p_owner => 'RE', p_table_name => 'RESIDUAL_STAGE',
    p_sql => 'DELETE /*+ ROWID (RESIDUAL_STAGE) */ FROM RE.RESIDUAL_STAGE WHERE rowid BETWEEN :start_id AND
:end_id AND processing_period < sysdate - :keep_days',
    p_keep_days => 60);
END;
```


Do large deletes or updates the safe way in Oracle


Redacting selected data columns in the Oracle databases


Install Oracle 11g Standalone