Partitioning an Existing Table using DBMS_REDEFINITION

This article presents a simple method for partitioning an existing table using the `DBMS_REDEFINITION` package, introduced in Oracle 9i. The contents of the article should not be used as an indication of when and how to partition objects, it simply shows the method of getting from A to B. Remember, in many cases incorrect partitioning is worse than no partitioning!

- Create a Sample Schema
- Create a Partitioned Interim Table
- Start the Redefinition Process
- Create Constraints and Indexes (Dependencies)
- Complete the Redefinition Process

Related articles.

- Online Conversion of a Non-Partitioned Table to a Partitioned Table in Oracle Database 12c Release 2 (12.2) (/articles/12c/online-conversion-of-a-non-partitioned-table-to-a-partitioned-table-12cr2)
- Partitioning an Existing Table using EXCHANGE PARTITION (/articles/misc/partitioning-an-existing-table-using-exchange-partition)
- All Partitioning Articles (/articles/misc/articles-misc#partitioning)

Create a Sample Schema

First we create a sample schema as our starting point.

```
-- Create and populate a small lookup table.
CREATE TABLE lookup (
  id            NUMBER(10),
  description   VARCHAR2(50)
);

ALTER TABLE lookup ADD (
  CONSTRAINT lookup_pk PRIMARY KEY (id)
);

INSERT INTO lookup (id, description) VALUES (1, 'ONE');
INSERT INTO lookup (id, description) VALUES (2, 'TWO');
INSERT INTO lookup (id, description) VALUES (3, 'THREE');
COMMIT;

-- Create and populate a larger table that we will later partition.
CREATE TABLE big_table (
  id            NUMBER(10),
  created_date  DATE,
  lookup_id     NUMBER(10),
  data          VARCHAR2(50)
);

DECLARE
  l_lookup_id    lookup.id%TYPE;
  l_create_date  DATE;
BEGIN
  FOR i IN 1 .. 1000000 LOOP
    IF MOD(i, 3) = 0 THEN
      l_create_date := ADD_MONTHS(SYSDATE, -24);
      l_lookup_id   := 2;
    ELSIF MOD(i, 2) = 0 THEN
      l_create_date := ADD_MONTHS(SYSDATE, -12);
      l_lookup_id   := 1;
    ELSE
      l_create_date := SYSDATE;
      l_lookup_id   := 3;
    END IF;

    INSERT INTO big_table (id, created_date, lookup_id, data)
    VALUES (i, l_create_date, l_lookup_id, 'This is some data for ' || i);
  END LOOP;
  COMMIT;
END;
/

-- Apply some constraints to the table.
ALTER TABLE big_table ADD (
  CONSTRAINT big_table_pk PRIMARY KEY (id)
```

```
);

CREATE INDEX bita_created_date_i ON big_table(created_date);

CREATE INDEX bita_look_fk_i ON big_table(lookup_id);

ALTER TABLE big_table ADD (
  CONSTRAINT bita_look_fk
  FOREIGN KEY (lookup_id)
  REFERENCES lookup(id)
);

-- Gather statistics on the schema objects
EXEC DBMS_STATS.gather_table_stats(USER, 'LOOKUP', cascade => TRUE);
EXEC DBMS_STATS.gather_table_stats(USER, 'BIG_TABLE', cascade => TRUE);
```

Create a Partitioned Interim Table

Next we create a new table with the appropriate partition structure to act as an interim table.

```
-- Create partitioned table.
CREATE TABLE big_table2 (
  id            NUMBER(10),
  created_date  DATE,
  lookup_id     NUMBER(10),
  data          VARCHAR2(50)
)
PARTITION BY RANGE (created_date)
(PARTITION big_table_2003 VALUES LESS THAN (TO_DATE('01/01/2004', 'DD/MM/YYYY')),
 PARTITION big_table_2004 VALUES LESS THAN (TO_DATE('01/01/2005', 'DD/MM/YYYY')),
 PARTITION big_table_2005 VALUES LESS THAN (MAXVALUE));
```

With this interim table in place we can start the online redefinition.

Start the Redefinition Process

First we check the redefinition is possible using the following command.

```
EXEC DBMS_REDEFINITION.can_redef_table(USER, 'BIG_TABLE');
```

If no errors are reported it is safe to start the redefinition using the following command.

```
-- Alter parallelism to desired level for large tables.
--ALTER SESSION FORCE PARALLEL DML PARALLEL 8;
--ALTER SESSION FORCE PARALLEL QUERY PARALLEL 8;

BEGIN
  DBMS_REDEFINITION.start_redef_table(
    uname      => USER,
    orig_table => 'BIG_TABLE',
    int_table  => 'BIG_TABLE2');
END;
/
```

Depending on the size of the table, this operation can take quite some time to complete.

Create Constraints and Indexes (Dependencies)

If there is delay between the completion of the previous operation and moving on to finish the redefinition, it may be sensible to resynchronize the interim table before building any constraints and indexes. The resynchronization of the interim table is initiated using the following command.

```
-- Optionally synchronize new table with interim data before index creation
BEGIN
  dbms_redefinition.sync_interim_table(
    uname      => USER,
    orig_table => 'BIG_TABLE',
    int_table  => 'BIG_TABLE2');
END;
/
```

The dependent objects will need to be created against the new table. This is done using the `COPY_TABLE_DEPENDENTS` procedure. You can decide which dependencies should be copied.

```
SET SERVEROUTPUT ON
DECLARE
  l_errors  NUMBER;
BEGIN
  DBMS_REDEFINITION.copy_table_dependents(
    uname            => USER,
    orig_table       => 'BIG_TABLE',
    int_table        => 'BIG_TABLE2',
    copy_indexes     => DBMS_REDEFINITION.cons_orig_params,
    copy_triggers    => TRUE,
    copy_constraints => TRUE,
    copy_privileges  => TRUE,
    ignore_errors    => FALSE,
    num_errors       => l_errors,
    copy_statistics  => FALSE,
    copy_mvlog       => FALSE);

  DBMS_OUTPUT.put_line('Errors=' || l_errors);
END;
/
```

The fact you are partitioning the table means you should probably consider the way you are indexing the table. You may want to manually create the constraints and indexes against the interim table using alternate names to prevent errors. The indexes should be created with the appropriate partitioning scheme to suit their purpose.

```
-- Add new keys, FKs and triggers.
ALTER TABLE big_table2 ADD (
  CONSTRAINT big_table_pk2 PRIMARY KEY (id)
);

CREATE INDEX bita_created_date_i2 ON big_table2(created_date) LOCAL;

CREATE INDEX bita_look_fk_i2 ON big_table2(lookup_id) LOCAL;

ALTER TABLE big_table2 ADD (
  CONSTRAINT bita_look_fk2
  FOREIGN KEY (lookup_id)
  REFERENCES lookup(id)
);

-- Gather statistics on the new table.
EXEC DBMS_STATS.gather_table_stats(USER, 'BIG_TABLE2', cascade => TRUE);
```

Complete the Redefinition Process

Once the constraints and indexes have been created the redefinition can be completed using the following command.

```
BEGIN
  dbms_redefinition.finish_redef_table(
    uname      => USER,
    orig_table => 'BIG_TABLE',
    int_table  => 'BIG_TABLE2');
END;
/
```

At this point the interim table has become the "real" table and their names have been switched in the data dictionary. All that remains is to perform some cleanup operations.

```
-- Remove original table which now has the name of the interim table.
DROP TABLE big_table2;

-- Rename all the constraints and indexes to match the original names.
ALTER TABLE big_table RENAME CONSTRAINT big_table_pk2 TO big_table_pk;
ALTER TABLE big_table RENAME CONSTRAINT bita_look_fk2 TO bita_look_fk;
ALTER INDEX big_table_pk2 RENAME TO big_table_pk;
ALTER INDEX bita_look_fk_i2 RENAME TO bita_look_fk_i;
ALTER INDEX bita_created_date_i2 RENAME TO bita_created_date_i;
```

The following queries show that the partitioning was successful.

```
SELECT partitioned
FROM   user_tables
WHERE  table_name = 'BIG_TABLE';

PAR
---
YES

1 row selected.

SELECT partition_name
FROM   user_tab_partitions
WHERE  table_name = 'BIG_TABLE';

PARTITION_NAME
------------------------------
BIG_TABLE_2003
BIG_TABLE_2004
BIG_TABLE_2005

3 rows selected.
```

For more information see:

- Online Conversion of a Non-Partitioned Table to a Partitioned Table in Oracle Database 12c Release 2 (12.2) (/articles/12c/online-conversion-of-a-non-partitioned-table-to-a-partitioned-table-12cr2)

- Partitioning an Existing Table using EXCHANGE PARTITION (/articles/misc/partitioning-an-existing-table-using-exchange-partition)
- All Partitioning Articles (/articles/misc/articles-misc#partitioning)
- DBMS_REDEFINITION (http://docs.oracle.com/cd/B10501_01/appdev.920/a96612/d_redefi.htm)

Hope this helps. Regards Tim...

Back to the Top.

22 comments, read/add them... (/misc/comments?page_id=334)