Compressed Tables

Oracle 9i onward allows whole tables or individual table partitions to be compressed to reduce disk space requirements. The basic compression described here is a free feature in the Enterprise Edition of the database. It is designed for compressing static data, since it only works for direct path inserts, not single row insert, update and delete operations typical in OLTP systems.

- Create Table
- Alter Table
- Partitioned Tables
- Check the Level of Compression
- Considerations

Related articles.

- Table Compression Enhancements in Oracle Database 11g Release 1
- Index Key Compression

Create Table

Table compression is enabled by adding the `COMPRESS` keyword to the end of the table definition, as shown below.

```
DROP TABLE test_tab;

CREATE TABLE test_tab (
  id            NUMBER(10)    NOT NULL,
  description   VARCHAR2(100) NOT NULL,
  created_date  DATE          NOT NULL,
  created_by    VARCHAR2(50)  NOT NULL,
  updated_date  DATE,
  updated_by    VARCHAR2(50)
)
COMPRESS;
```

The default compression status is `NOCOMPRESS`.

The compression status of the table can be displayed using `COMPRESSION` column in the `[DBA|ALL|USER]_TABLES` views. The introduction of alternative modes of compression in Oracle 11g (described here) came with an additional column called `COMPRESS_FOR` to indicate the type of compression. If you are using a version prior to 11g, reference to the `COMPRESS_FOR` column will cause the queries in this article to fail, so just remove them.

```
SELECT compression, compress_for
FROM   user_tables
WHERE  table_name = 'TEST_TAB';

COMPRESS COMPRESS_FOR
-------- ------------------------------
ENABLED  BASIC

SQL>
```

Alter Table

The compression status of an existing table can be changed using the `ALTER TABLE` statement.

```
ALTER TABLE test_tab NOCOMPRESS;

SELECT compression, compress_for
FROM   user_tables
WHERE  table_name = 'TEST_TAB';

COMPRESS COMPRESS_FOR
-------- ------------------------------
DISABLED

SQL>


ALTER TABLE test_tab COMPRESS;

SELECT compression, compress_for
FROM   user_tables
WHERE  table_name = 'TEST_TAB';


COMPRESS COMPRESS_FOR
-------- ------------------------------
ENABLED  BASIC
```

```
SQL>
```

This doesn't affect the compression of existing data, but will affect the compression of new data that is loaded by direct path loads. If you want to compress existing data, you must perform a move operation, so the data gets compressed during the copy.

```
ALTER TABLE test_tab MOVE NOCOMPRESS;
ALTER TABLE test_tab MOVE COMPRESS;
```

Remember, when you perform a move operation you will have two copies of the table for a period of time. Make sure you have enough storage to cope with this.

A similar action could be performed for an individual partition of a partitioned table.

```
ALTER TABLE test_tab MOVE PARTITION test_tab_q2 COMPRESS;
```

Partitioned Tables

Using the `COMPRESS` keyword at the table level makes this the default for all partitions.

```
DROP TABLE test_tab;

CREATE TABLE test_tab (
  id            NUMBER(10)    NOT NULL,
  description   VARCHAR2(100) NOT NULL,
  created_date  DATE          NOT NULL,
  created_by    VARCHAR2(50)  NOT NULL,
  updated_date  DATE,
  updated_by    VARCHAR2(50)
)
COMPRESS
PARTITION BY RANGE (created_date) (
  PARTITION test_tab_q1 VALUES LESS THAN (TO_DATE('01/04/2003', 'DD/MM/YYYY')),
  PARTITION test_tab_q2 VALUES LESS THAN (MAXVALUE)
);
```

The `[DBA|ALL|USER]_TAB_PARTITIONS` views display the compression status of the individual partitions.

```
COLUMN partition_name FORMAT A30

SELECT partition_name, compression, compress_for
FROM   user_tab_partitions
```

```
WHERE  table_name = 'TEST_TAB'
ORDER BY 1;

PARTITION_NAME                 COMPRESS COMPRESS_FOR
------------------------------ -------- ------------------------------
TEST_TAB_Q1                    ENABLED  BASIC
TEST_TAB_Q2                    ENABLED  BASIC

SQL>
```

It is possible to control the compression status of individual partitions in a partitioned table. The compression status can be specified for all partitions explicitly, or a table default can be used, with only exceptions to this default specified at partition level. In the following example the default `NOCOMPRESS` option is set explicitly at table level (this is not necessary), with the first partition using the `COMPRESS` keyword to override the default.

```
DROP TABLE test_tab;

CREATE TABLE test_tab (
  id            NUMBER(10)    NOT NULL,
  description   VARCHAR2(100) NOT NULL,
  created_date  DATE          NOT NULL,
  created_by    VARCHAR2(50)  NOT NULL,
  updated_date  DATE,
  updated_by    VARCHAR2(50)
)
NOCOMPRESS
PARTITION BY RANGE (created_date) (
  PARTITION test_tab_q1 VALUES LESS THAN (TO_DATE('01/04/2003', 'DD/MM/YYYY')) COMPRESS,
  PARTITION test_tab_q2 VALUES LESS THAN (MAXVALUE)
);


COLUMN partition_name FORMAT A30

SELECT partition_name, compression, compress_for
FROM   user_tab_partitions
WHERE  table_name = 'TEST_TAB'
ORDER BY 1;

PARTITION_NAME                 COMPRESS COMPRESS_FOR
------------------------------ -------- ------------------------------
TEST_TAB_Q1                    ENABLED  BASIC
TEST_TAB_Q2                    DISABLED
```

```
SQL>
```

As mentioned previously, we can alter the compression status of the partition and compress its contents using the ALTER TABLE command.

```
ALTER TABLE test_tab MOVE PARTITION test_tab_q2 COMPRESS;

COLUMN partition_name FORMAT A30

SELECT partition_name, compression, compress_for
FROM   user_tab_partitions
WHERE  table_name = 'TEST_TAB'
ORDER BY 1;

PARTITION_NAME                 COMPRESS COMPRESS_FOR
------------------------------ -------- ------------------------------
TEST_TAB_Q1                    ENABLED  BASIC
TEST_TAB_Q2                    ENABLED  BASIC

SQL>
```

Check the Level of Compression

Create the following table with one compressed partition and one non-compressed partititon.

```
DROP TABLE test_tab;

CREATE TABLE test_tab (
  id            NUMBER(10)    NOT NULL,
  description   VARCHAR2(100) NOT NULL,
  created_date  DATE          NOT NULL,
  created_by    VARCHAR2(50)  NOT NULL,
  updated_date  DATE,
  updated_by    VARCHAR2(50)
)
NOCOMPRESS
PARTITION BY RANGE (created_date) (
  PARTITION test_tab_q1 VALUES LESS THAN (TO_DATE('01/04/2003', 'DD/MM/YYYY')) COMPRESS,
  PARTITION test_tab_q2 VALUES LESS THAN (MAXVALUE)
);
```

Insert some rows into each partition using a direct path insert. We will do this separately and capture the elapsed time and CPU time used by each operations.

```
-- TEST_TAB : Direct path insert into non-compressed partition.
SET SERVEROUTPUT ON
DECLARE
  v_date        test_tab.created_date%TYPE := SYSDATE;
  v_user        test_tab.created_by%TYPE   := USER;
  l_start_time  NUMBER;
  l_start_cpu   NUMBER;
BEGIN

  l_start_time := DBMS_UTILITY.get_time;
  l_start_cpu  := DBMS_UTILITY.get_cpu_time;

  INSERT /*+ APPEND */ INTO test_tab (id, description, created_date, created_by)
  SELECT level,
         'A very repetitive, and therefore very compressible column value',
         v_date,
         v_user
  FROM   dual
  CONNECT BY level <= 1000000;
  COMMIT;

  DBMS_OUTPUT.put_line('CPU Time (hsecs)    : ' || (DBMS_UTILITY.get_cpu_time - l_start_cpu)
  DBMS_OUTPUT.put_line('Elapsed Time (hsecs): ' || (DBMS_UTILITY.get_time - l_start_time));
END;
/
CPU Time (hsecs)    : 89
Elapsed Time (hsecs): 116

PL/SQL procedure successfully completed.

SQL>



-- TEST_TAB : Direct path insert into compressed partition.
SET SERVEROUTPUT ON
DECLARE
  v_date        test_tab.created_date%TYPE := TO_DATE('31/03/2003', 'DD/MM/YYYY');
  v_user        test_tab.created_by%TYPE   := USER;
  l_start_time  NUMBER;
  l_start_cpu   NUMBER;
BEGIN

  l_start_time := DBMS_UTILITY.get_time;
  l_start_cpu  := DBMS_UTILITY.get_cpu_time;

  INSERT /*+ APPEND */ INTO test_tab (id, description, created_date, created_by)
  SELECT level,
         'A very repetitive, and therefore very compressible column value',
         v_date,
         v_user
  FROM   dual
  CONNECT BY level <= 1000000;
  COMMIT;

  DBMS_OUTPUT.put_line('CPU Time (hsecs)    : ' || (DBMS_UTILITY.get_cpu_time - l_start_cpu)
  DBMS_OUTPUT.put_line('Elapsed Time (hsecs): ' || (DBMS_UTILITY.get_time - l_start_time));
END;
/
CPU Time (hsecs)    : 137
Elapsed Time (hsecs): 139

PL/SQL procedure successfully completed.

SQL>
```

From repeated runs we see direct path inserts into into a compressed table or partition requires more CPU and takes longer compared to an uncompressed table or partition.

Once the statistics are gathered we can check the compression using the `[DBA|ALL|USER]_TAB_PARTITIONS` views.

```
EXEC DBMS_STATS.gather_schema_stats(USER, cascade => TRUE);

COLUMN table_name FORMAT A20
COLUMN partition_name FORMAT A20

SELECT table_name,
       partition_name,
       compression,
       num_rows,
       blocks,
       empty_blocks
FROM   user_tab_partitions
WHERE  table_name = 'TEST_TAB'
ORDER BY 1;

TABLE_NAME           PARTITION_NAME       COMPRESS   NUM_ROWS     BLOCKS EMPTY_BLOCKS
-------------------- -------------------- -------- ---------- ---------- ------------
TEST_TAB             TEST_TAB_Q1          ENABLED     1000000       1421            0
TEST_TAB             TEST_TAB_Q2          DISABLED    1000000      12483            0
```

```
SQL>
```

What we see here is the compressed partition is almost an order of magnitude smaller for the same number of rows based on the number of blocks used to hold the data, so if storage size is an issue, the overhead in CPU and elapsed time for direct path inserts may be worth it.

Remember, the level of compression will vary depending on the data. In this case we cheated and used the same text data, username and created date for each row, making the data highly compressible in the block. A different data set may not have produced such a dramatic result.

Considerations

Some things to consider before using this functionality.

- Basic compression is a free option with the Enterprise Edition version of the database.
- Basic compression is not designed for OLTP operations. You will get no compression benefits from conventional path inserts, updates or deletes. It is designed for use with direct path loads only. If you want table compression suitable for OLTP operations you will need to use the Advanced Compression option available from 11g onwards (described here).
- Basic compression can be used with OLTP systems that use partitioning. For example, if you have a table that is partitioned by a date column, it's possible you will have partitions with older static data, which can be compressed, leaving the newer active data in uncompressed partitions.
- Depending on the nature of the queries, basic compression may improve query performance by reducing the number of blocks read from disk. This is heavily dependent on workload.
- As mentioned previously, the level of compression you can achieve will vary depending on the nature of the data being compressed. For basic compression, the compression is performed at block level, so an individual block must contain repetitive data for compression to have any impact.

For more information see:

- Table Compression Enhancements in Oracle Database 11g Release 1
- Index Key Compression

Hope this helps. Regards Tim...

Back to the Top.

Created: 2005-05-14 Updated: 2018-02-24