Practice

Using Table Compression

Practice Target

In this practice, you will examine in which scenarios basic compression and advanced compression take effect.

Practice Assumptions

- You have the `srv1` and its **CDB** database up and running.

Examining the Way Compression Types Work

In this section of the practice, you will create three tables of different compression types and examine their effect on saving disk space.

1. Open Putty, login as `oracle` to `srv1`.

2. Run the following code to create a script file named as `display_table_stats.sql`

The script displays the size statistics about the tables that will be created in this practice.

```
cat > display_table_stats.sql <<EOL
col TABLE_NAME format a17

SELECT TABLE_NAME, BLOCKS*8 SIZE_KB, COMPRESSION, PCT_FREE
FROM USER_TABLES
WHERE TABLE_NAME IN ('CUST_SOURCE','CUST_BCOMPRESSED','CUST_ACOMPRESSED');

EOL
```

3. Invoke SQL*Plus, and login as `HR` to `PDB1`

```
sqlplus HR/ABcd##1234@//srv1/pdb1.localdomain
```

4. Run the following code to create three tables. One is not compressed (`CUST_SOURCE`), another one is basic compressed (`CUST_BCOMPRESSED`), and the last one is advanced compressed (`CUST_ACOMPRESSED`).

The `PCTFREE` is set to zero in the advanced compressed table because by default it is set to 10. It is not set in the basic compressed table because it is by default set to 0 in the basic compressed tables.

Note: Using advanced compression in production systems requires a separate license.

```
CREATE TABLE CUST_SOURCE
( CUSTOMER_NO  NUMBER(6),
  FIRST_NAME VARCHAR2(20),
  LAST_NAME  VARCHAR2(20),
  NOTE1 VARCHAR2(100),
  NOTE2 VARCHAR2(100)
) NOLOGGING PCTFREE 0;

CREATE TABLE CUST_BCOMPRESSED
( CUSTOMER_NO  NUMBER(6),
  FIRST_NAME VARCHAR2(20),
  LAST_NAME  VARCHAR2(20),
  NOTE1 VARCHAR2(100),
  NOTE2 VARCHAR2(100)
) ROW STORE COMPRESS BASIC NOLOGGING;
```

```
CREATE TABLE CUST_ACOMPRESSED
( CUSTOMER_NO  NUMBER(6),
  FIRST_NAME VARCHAR2(20),
  LAST_NAME  VARCHAR2(20),
  NOTE1 VARCHAR2(100),
  NOTE2 VARCHAR2(100)
) ROW STORE COMPRESS ADVANCED NOLOGGING PCTFREE 0;
```

5. Run the following code to populate `CUST_SOURCE` with random data, copy the data from `CUST_SOURCE` to `CUST_BCOMPRESSED`, and then gather their optimizer statistics.

Because the data was populated randomly, there is hardly any chance of having repeated data in the columns.

You will learn about gathering segment statistics later in the course.

```
INSERT INTO CUST_SOURCE
SELECT LEVEL AS CUSTOMER_NO,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(10,20))) FIRST_NAME,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(10,20))) LAST_NAME,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(0,100))) NOTE1,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(0,100))) NOTE2
FROM DUAL
CONNECT BY LEVEL <= 100000;
COMMIT;

INSERT INTO CUST_BCOMPRESSED SELECT * FROM CUST_SOURCE;
COMMIT;

exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_SOURCE')
exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_BCOMPRESSED')
```

6. Display the table statistics.

Observe that the compressed table is of the same size as the non-compressed table.

Conclusion: Basic compression does not compress data inserted by normal `INSERT` statements.

```
@ display_table_stats.sql
```

7. Run the following code to repopulate the compressed table with the same data but using direct path loading this time.

When the `APPEND` hint is used with the INSERT statement, Oracle database uses direct-path loading to load the data into the table.

```
TRUNCATE TABLE CUST_BCOMPRESSED;

INSERT /*+ APPEND */ INTO CUST_BCOMPRESSED SELECT * FROM CUST_SOURCE;
COMMIT;

exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_BCOMPRESSED')
```

8. Display the table statistics.

Observe that the compressed table is slightly smaller than the non-compressed table. The compression has a slight effect because there is hardly any repeated data in the columns.

Conclusion: Basic compression does compress data inserted by direct path loading method.

```
@ display_table_stats.sql
```

9. Run the following code to repopulate the non-compressed and compressed tables.

This time the data has a lot of repeated data in its columns `NOTE1` and `NOTE2`.

```
TRUNCATE TABLE CUST_SOURCE;
TRUNCATE TABLE CUST_BCOMPRESSED;

INSERT INTO CUST_SOURCE
SELECT LEVEL AS CUSTOMER_NO,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(10,20))) FIRST_NAME,
       DBMS_RANDOM.STRING('A', TRUNC(DBMS_RANDOM.value(10,20))) LAST_NAME,
       'XXXXXX' NOTE1,
       'YYYYYY' NOTE2
FROM DUAL
CONNECT BY LEVEL <= 100000;
COMMIT;

INSERT /*+ APPEND */ INTO CUST_BCOMPRESSED SELECT * FROM CUST_SOURCE;
COMMIT;

exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_SOURCE')
exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_BCOMPRESSED')
```

10. Display the table statistics.

Observe that the compressed table is much smaller than the non-compressed table.

Conclusion: Basic compression ratio depends on the repeated data in the table columns.

```
@ display_table_stats.sql
```

11. Populate the advanced compressed table with the same data using the normal `INSERT` statement.

```
INSERT INTO CUST_ACOMPRESSED SELECT * FROM CUST_SOURCE;
COMMIT;
exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_ACOMPRESSED')
```

12. Display the table statistics.

Observe that the advanced compressed table is a little bit smaller than the non-compressed table but larger than the basic compressed table.

Conclusion: Advanced compression works with normal `INSERT` statements.

```
@ display_table_stats.sql
```

13. Repopulate the advanced compressed table with the same data using the direct path loading.

```
TRUNCATE TABLE CUST_ACOMPRESSED;

INSERT /*+ APPEND */ INTO CUST_ACOMPRESSED SELECT * FROM CUST_SOURCE;
COMMIT;

exec DBMS_STATS.GATHER_TABLE_STATS(USER,'CUST_ACOMPRESSED')
```

14. Display the table statistics.

Observe that the advanced compressed table is as small in size as the basic compressed table size.

Conclusion: With direct path loading, advanced compression has the same effect as the basic compression.

```
@ display_table_stats.sql
```

15. To clean up, drop the tables and the file that you created.

```
DROP TABLE CUST_SOURCE;
DROP TABLE CUST_BCOMPRESSED;
DROP TABLE CUST_ACOMPRESSED;
host rm display_table_stats.sql
```

Summary

- Basic compression effectivity depends on the data duplication in the table columns. The more duplicate data, the more efficient is the compression.
- Basic compression does not compress data inserted by normal `INSERT` statements. Basic compression compresses data inserted by direct path loading methods.
- Using direct path loading on a compressed table provides performance gain in loading the table.
- Advanced compression works with normal `INSERT` statements as well as with direct path loading methods.