**Practice**

# Shrinking Segments

## Practice Target

In this practice, you will create a testing table, make it fragmented, then shrink it.

## Practice Assumptions

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

Ahmed Baraka
Oracle Database Administrator

## Shrinking Segments

In the following steps, you will create a testing table and indexes, make the table fragmented, shrink the table, and examine the table statistics before and after shrinking it.

**1.** Start a Putty session to `srv1` as `oracle`.

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

The script displays the space statistics on the `CUST` table and its indexes.

```
cat > display_cust_stats.sql <<EOL

ANALYZE TABLE CUST COMPUTE STATISTICS;

SELECT BLOCKS, BLOCKS*8192/1024 TOTAL_SIZE_KB, NUM_ROWS,
round(BLOCKS*AVG_SPACE/1024,2) FREE_SPACE_KB
FROM USER_TABLES WHERE TABLE_NAME='CUST';

col INDEX_NAME for a20
ANALYZE INDEX CUST_FNAME_IDX COMPUTE STATISTICS;
ANALYZE INDEX CUST_NO_IDX COMPUTE STATISTICS;
SELECT SEGMENT_NAME INDEX_NAME, BLOCKS FROM USER_SEGMENTS WHERE SEGMENT_NAME IN
('CUST_NO_IDX','CUST_FNAME_IDX');
SELECT INDEX_NAME, STATUS, LEAF_BLOCKS, NUM_ROWS FROM USER_INDEXES WHERE
INDEX_NAME IN ('CUST_NO_IDX','CUST_FNAME_IDX');

EOL
```

**3.** Invoke SQL*Plus and connect to `pdb1` as `SYSTEM`.

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

**4.** Verify that the `USERS` tablespace is locally managed and with automatic segment space management (ASSM).

We can shrink objects only in locally managed tablespaces with ASSM.

```
SELECT EXTENT_MANAGEMENT, SEGMENT_SPACE_MANAGEMENT FROM dba_tablespaces WHERE
TABLESPACE_NAME='USERS';
```

**5.** As `HR`, run the following code to create a testing table named as `CUST` and two indexes on it.

```
conn hr/ABcd##1234@//srv1/pdb1.localdomain
CREATE TABLE CUST
( CUSTOMER_NO  NUMBER(6),
 FIRST_NAME VARCHAR2(20),
 LAST_NAME  VARCHAR2(20),
 NOTE1 VARCHAR2(100),
 NOTE2 VARCHAR2(100)
) PCTFREE 0 TABLESPACE USERS;
```

```
INSERT INTO CUST
SELECT LEVEL AS CUSTOMER_NO,
       DBMS_RANDOM.STRING('A', 15) FIRST_NAME,
       DBMS_RANDOM.STRING('A', 12) 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;

CREATE INDEX CUST_NO_IDX ON CUST (CUSTOMER_NO) NOLOGGING TABLESPACE USERS;
CREATE INDEX CUST_FNAME_IDX ON CUST(FIRST_NAME) NOLOGGING TABLESPACE USERS;
```

**6.** Run the script `display_cust_stats.sql` to display statistics on the `CUST` table. Take a note of the output. Those are the space statistics of the objects after initializing them.

The free space represents a small percentage of the total table size.

```
@ display_cust_stats.sql
```

**7.** Run the following code to make the `CUST` table fragmented. The code performs updates and deletes on a high percentage of the table rows.

```
BEGIN
 FOR I IN 1..100000 LOOP
  IF MOD(I,2)=0 THEN
   UPDATE CUST SET NOTE1=NULL, NOTE2=DBMS_RANDOM.STRING('A',
TRUNC(DBMS_RANDOM.value(0,10))) WHERE CUSTOMER_NO=I;
   IF MOD(I,100)=0 THEN
     COMMIT;
   END IF;
  ELSIF MOD(I,3)=0 THEN
   DELETE CUST WHERE CUSTOMER_NO=I;
  END IF;
 END LOOP;
 COMMIT;
END;
/
```

**8.** Run the script `display_cust_stats.sql` to display the size of the deleted space in the table. Take a note of the output. This is the space statistics of the table after fragmentation.

Nearly 20% of the table space is a deleted space. As expected, the total number of rows reduced in the table and in the indexes.

The total number of blocks did not change because we only performed `UPDATE` and `DELETE` statements.

The fragmentation did not change the space statistics on the indexes.

```
@ display_cust_stats.sql
```

**9.** Issue the following statement to enable row movement in the CUST table, shrink the table, and disable the row movement.

The enable row movement statement fails if there is an open transaction lock on the table.

CASCADE option is used to shrink the indexes as well.

```
ALTER TABLE CUST ENABLE ROW MOVEMENT;
ALTER TABLE CUST SHRINK SPACE CASCADE;
ALTER TABLE CUST DISABLE ROW MOVEMENT;
```

**10.** Run the script display_cust_stats.sql to verify that the table was shrunk. Compare the output with the statistics taken before the defragmentation.

For the table, the total number of blocks reduced to nearly half of the previous number of blocks. The free space represents a small percentage of the total table size.

For indexes, there is negligible impact on the index sizes.

```
@ display_cust_stats.sql
```

**11.** Rebuild the indexes and check out the changes on their space statistics.

Rebuilding the indexes has better impact on saving index space than shrinking.

```
ALTER INDEX CUST_FNAME_IDX REBUILD;
ALTER INDEX CUST_NO_IDX REBUILD;

@ display_cust_stats.sql
```

## Extra Suggested Practice

Consider studying the defragmenting the CUST table by moving it. Repeat the same steps in the beginning of this practice to make the table fragmented. Move the table using the statement ALTER TABLE MOVE TABLESPACE (you can move it to the same tablespace). Then display the table statistics to check on the statement defragmentation quality.

## Cleanup

**12.** Drop the CUST table.

```
DROP TABLE CUST PURGE;
```

**13.** Delete the script file.

```
host rm display_cust_stats.sql
```

Ahmed Baraka Oracle Database Administrator

## Summary

- Shrinking tables performs defragmentation on tables. The operation can be cascaded to include the table indexes.

Ahmed Baraka
Oracle Database Administrator