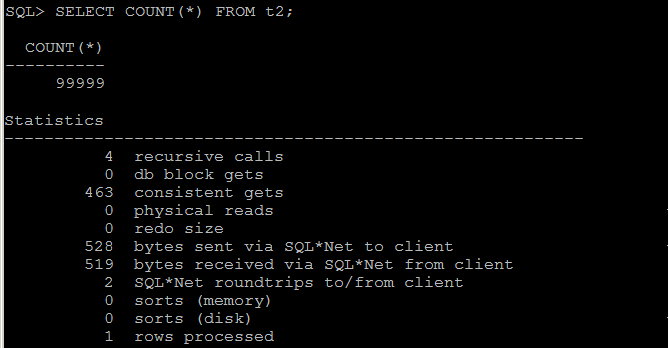
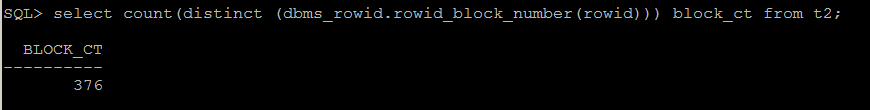
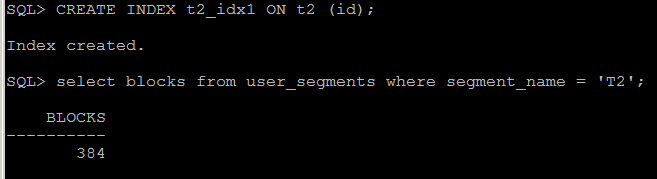
# Table access full scan

## Task 1: Full Scans and the High-water Mark and Block reading

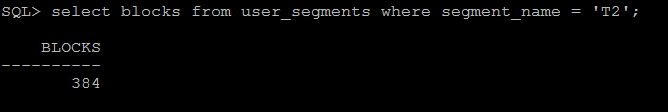
Step 1: CREATE TABLE t2 AS SELECT TRUNC( rownum / 100 ) id, rpad( rownum,100 ) t\_pad FROM dual CONNECT BY rownum < 100000;

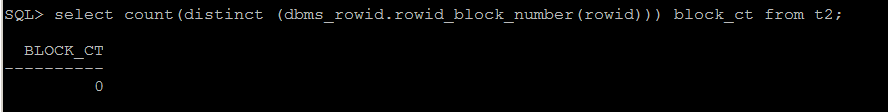
Step 2: CREATE INDEX t2\_idx1 ON t2 ( id );

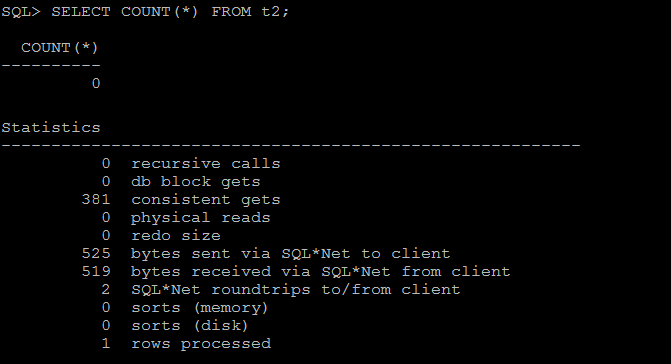
**Step 3:**

Step 4: DELETE FROM t2;

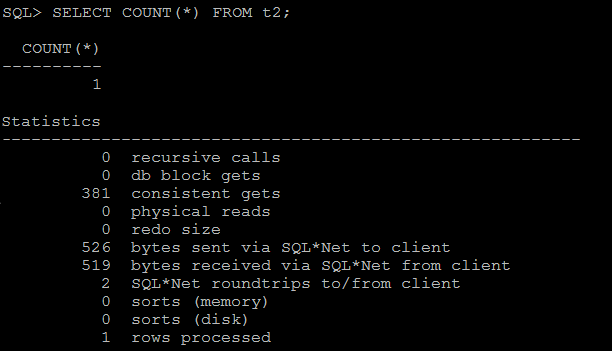
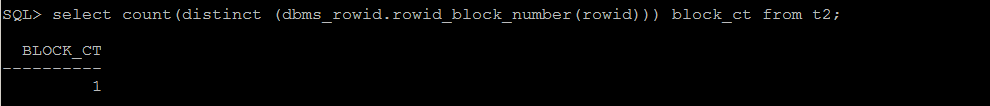
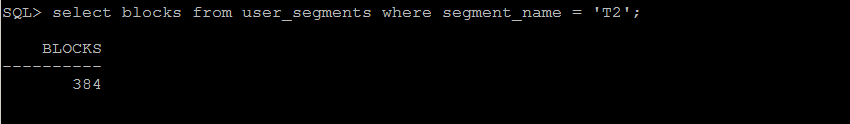
**Step 5:**



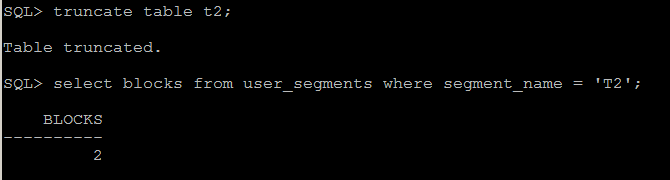


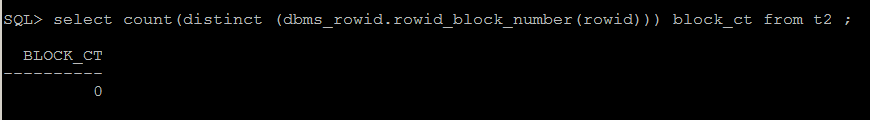


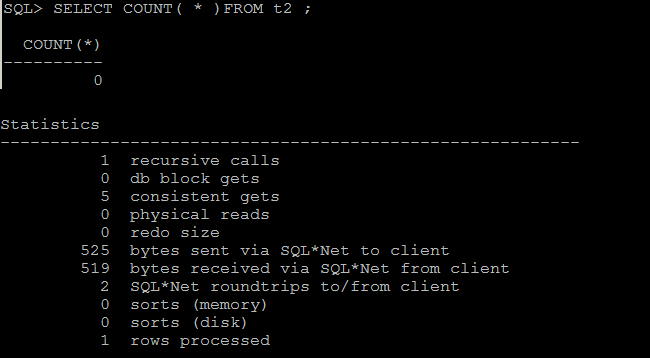
Step 6: INSERT INTO t2 ( ID, T\_PAD ) VALUES ( 1,'1' ); COMMIT;

**Step 7:** 

Step 8: TRUNCATE TABLE t2;

**Step 9:**  



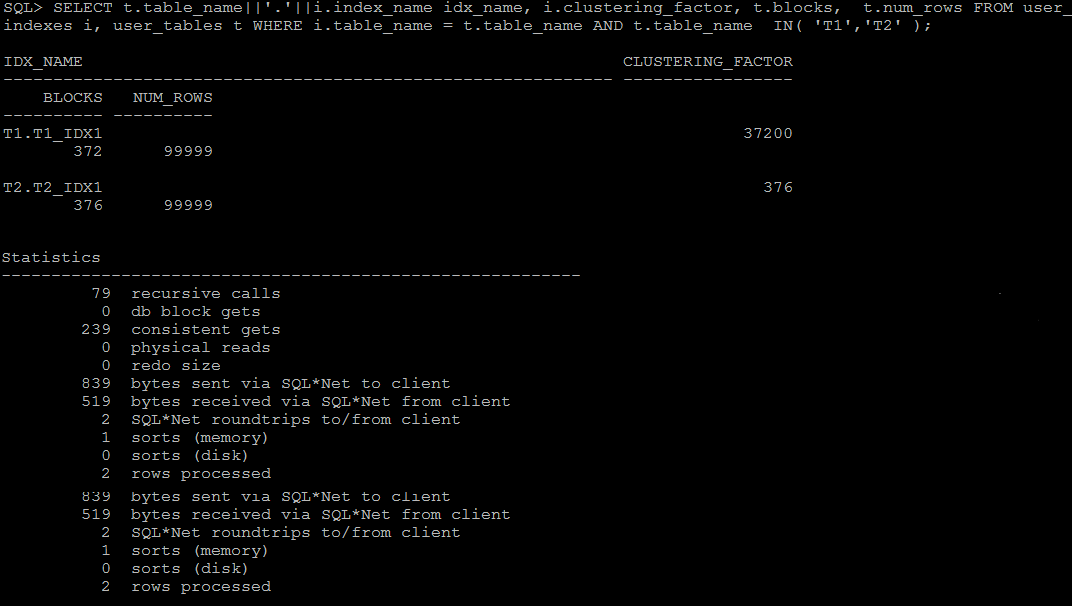


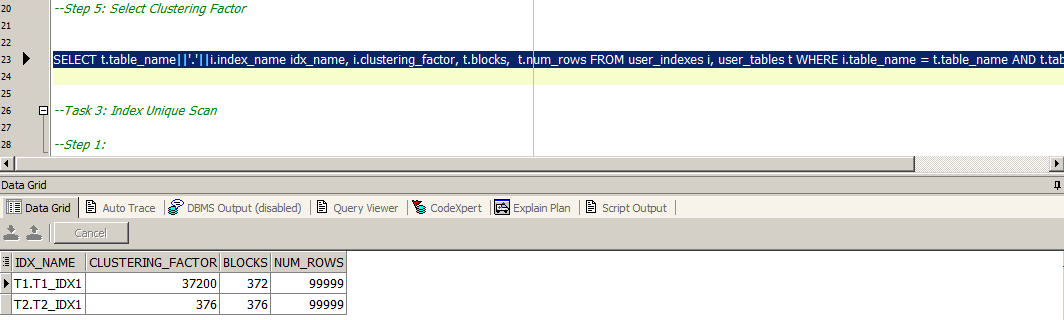
**Task Results:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| Step3 | 384 | 376 | 99999 | 463 | Consistent gets shows number of times a consistent read was requested for a block. |
| Step5 | 384 | 0 | 0 | 381 | After the command “delete from t2;”, We have cleaned all rows, but HWM in each blocks are stayed in the same place. So blocks were read |
| Step7 | 384 | 1 | 1 | 381 | was added one row, but each blocks were read again |
| Step9 | 2 | 0 | 0 | 5 | Command TRUNCATE cleans all rows and moves HWM of each blocks to ‘zero’ |

# Index Scan types

## Task 2: Index Clustering factor parameter

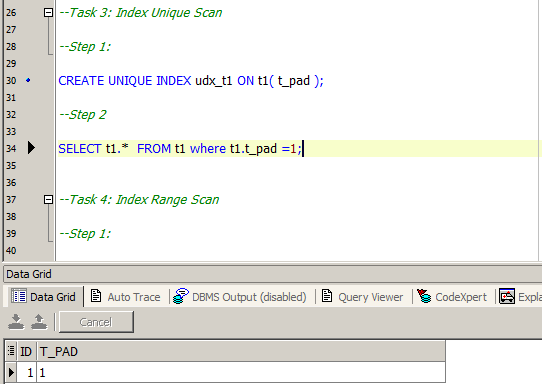
**Task Results:** 

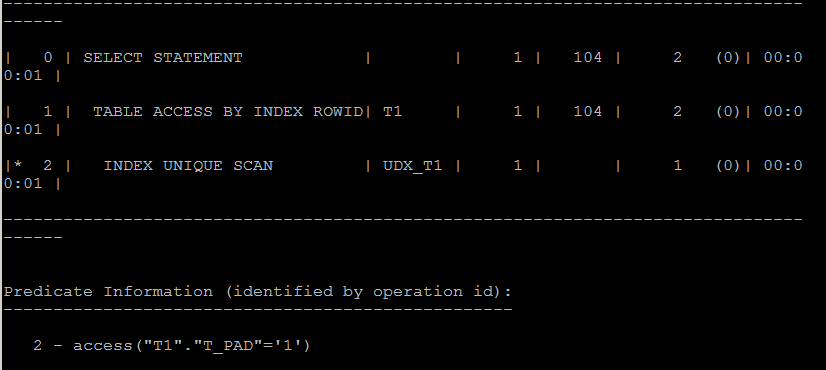


**The Clustering Factor** is the most important index related statistic. If the data in the table is relatively well clustered in relation to the index, then an index range scan can visit relatively few table blocks to obtain the necessary data. If the data is effectively randomised and not well clustered in relation to the index, then an index range scan has to visit many more table blocks and not be as effective as a result.

We have different values for t1\_idx1 and t2\_inx1 because values of a certain number for table t2 were write into the same block.

## Task 3: Index Unique Scan

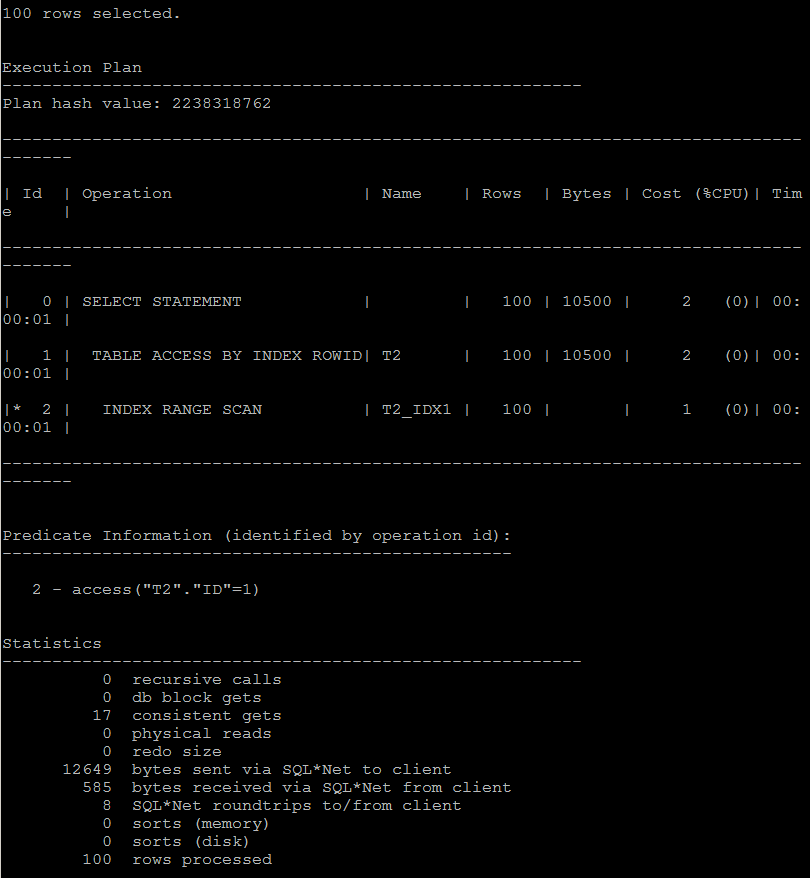
**Task Results:** 



You have a unique index on udx\_t1, and an equality predicate against t\_pad, so the optimizer chose to use the unique index to get the rowid for the row with t1.t\_pad= '1'.

All indexes store the rowid of the row indexed as part of the index. That is why an index can speed up access to the table, since typically the index is much smaller than the table, and is stored sorted by the value of the indexed column.

## Task 4: Index Range Scan

**Task Results:**

So as we have no unique index and we use filter by particular value, optimizer chooses index range scan. For an index based on B - tree in the search, we can get down to the required initial leaf blocks and continue to move in leaf blocks only, not the entire view every time an index structure from the top down

## Task 5: Index Skip Scan Task Results: C:\Users\Aliaksandr_Ahushevic\Desktop\task5_1(+index_ss).pngC:\Users\Aliaksandr_Ahushevic\Desktop\task5_2(+full).png

During a skip scan, the composite index is accessed once for each distinct value of the leading column(s). For each distinct value, the index is searched to find the query's target values.

Skip scanning is advantageous if there are few distinct values in the leading column of the composite index and many distinct values in the nonleading key of the index.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
|  |  |  | 1 | 3 | Skip Scan |
|  |  |  | 1 | 4 | Full |