**1. Table access full scan**

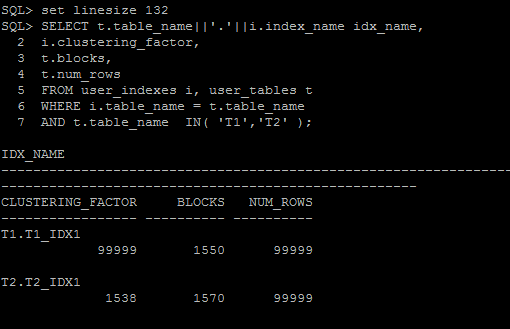
1.1. Task 1: Full Scans and the High-water Mark and Block reading

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1539 | We inserted 99999 rows into the table. For this required 1536 blocks, but oracle reserved 1664 block for the table, because the segment size increases discretely. For the query database refers to the buffer 1539 times (consistent gets) to read all blocks which contains the table. |
| 2 | 1664 | 0 | 0 | 1541 | After deleting data from the table the number of the reserved blocks is still 1664 because HWM remained at the same level and DB will read all of the blocks to find or not found data. |
| 3 | 1664 | 1 | 1 | 1541 | After inserting 1 row, HWM remained at the same level DB will read all of the blocks to find the row. |
| 4 | 8 | 0 | 0 | 3 | After truncating, the table HWM set to zero and DB knows that all required blocks are empty now and DB doesn’t need to read all empty block to find data. |

**2. Index Scan types**

**2.1. Task 2: Index Clustering factor parameter**

Step 5:



Clustering factor shows how rows are ordered in a table according to the values ​​of the index. If clustering factor value is closed to the total number of blocks (table t2) then the table is very well ordered. If clustering factor value is closed to the total number of rows (table 1) then the table is very disordered.

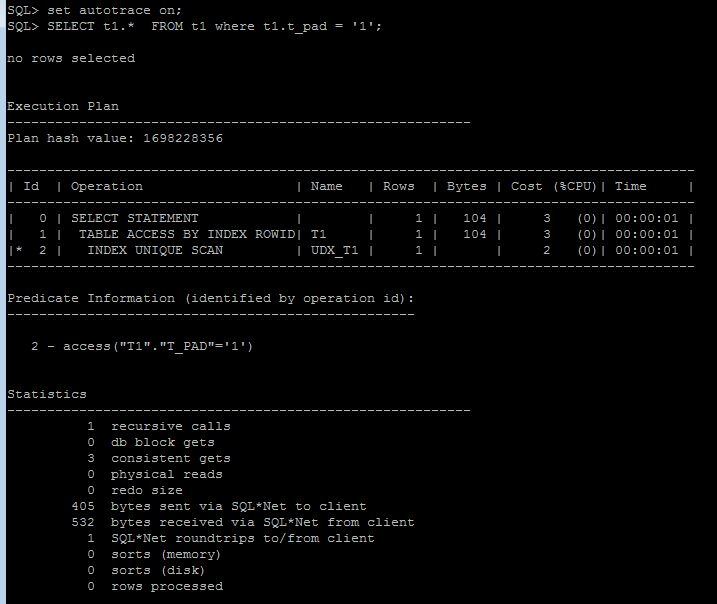
We have different values for indexes t1\_idx1 and t2\_idx1 because in the table T2 column ID was filled in ascending order and this column is used for creating index. So, all rows with the same values of ID is lying one behind the other in blocks.

In the table T1 rows with the same values of ID is repeated every 100 rows and this values are not sorteed. So, the rows with the same values of ID is lying in different blocks.

Index t2\_idx1 has the best selective performance in execution Select clause filtered by IN ( , list of values, ), because all rows in the table T2 is lying in the sorted order in the blocks.

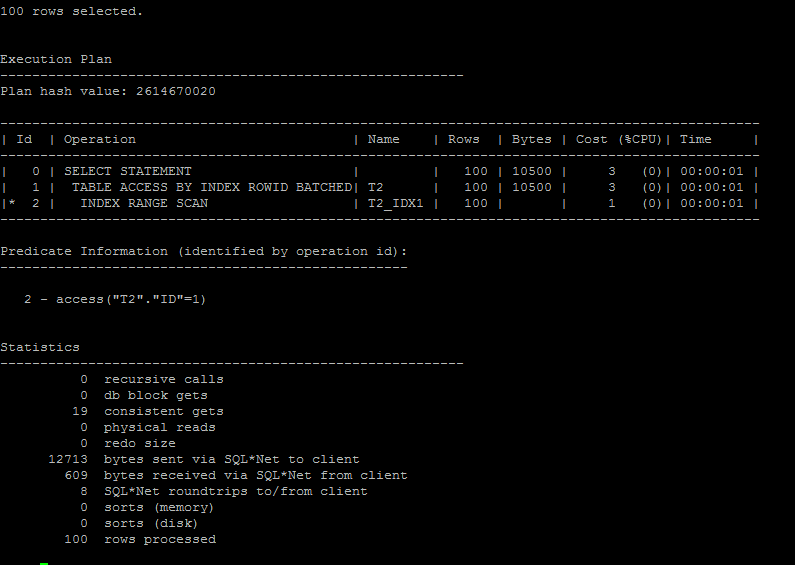
**2.2. Task 3: Index Unique Scan**

**Step 2**



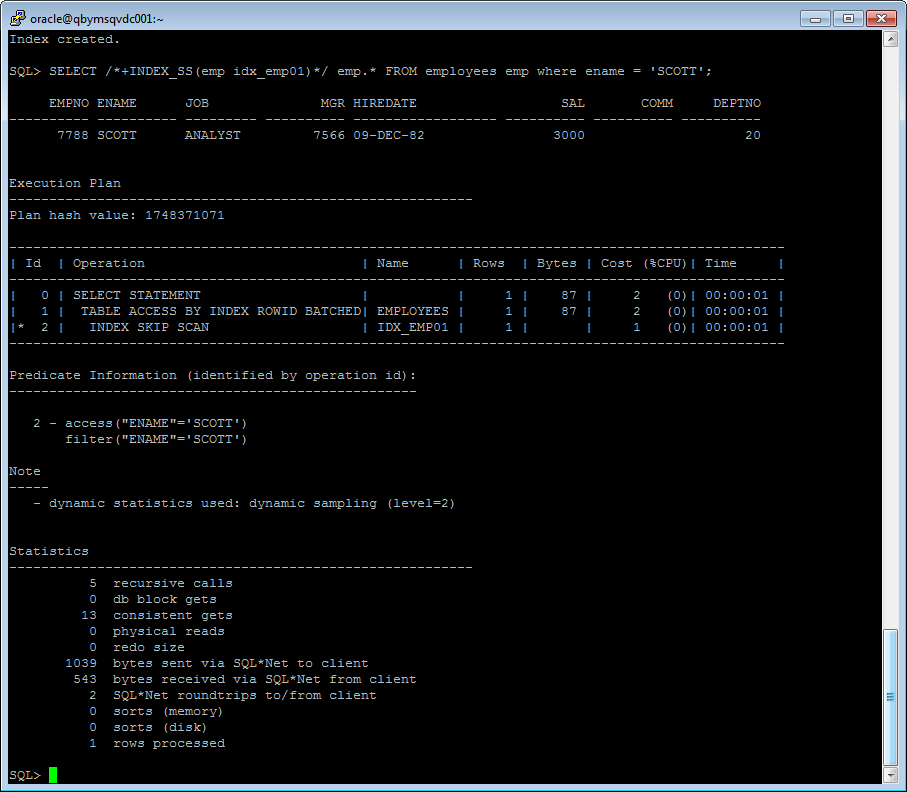
We have created a unique index on the ID column. Oracle used this index to find distinct value of the ID which is used in the ‘where’ clause. After the index is found oracle gets values of the rowid and read all needed rows from table, using this rowid.

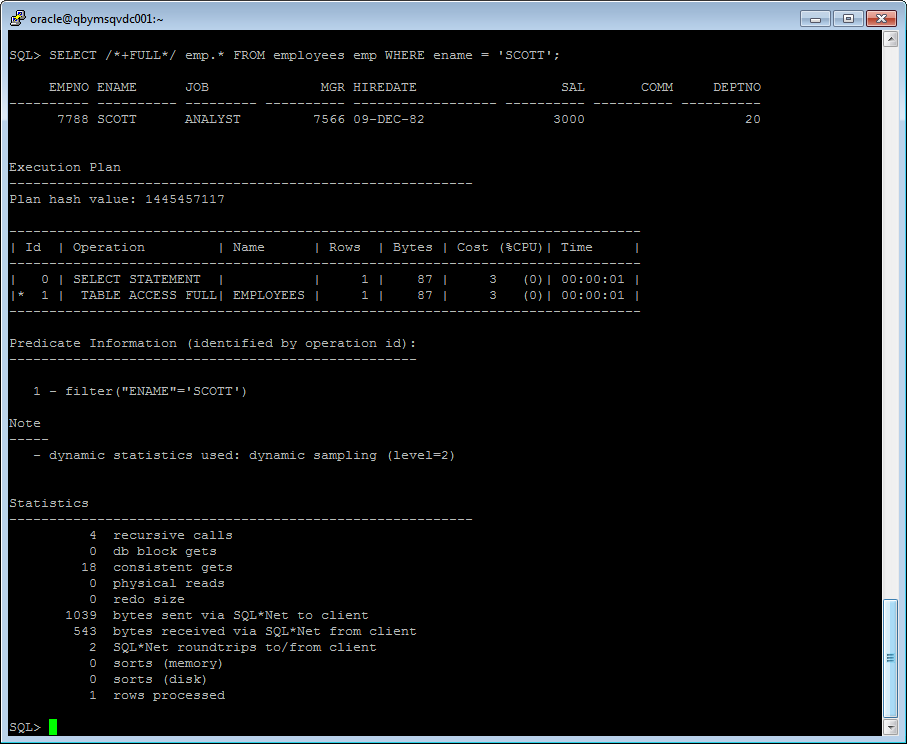
**2.3. Task 4: Index Range Scan**



For this table (T2) we haven’t created unique index, so oracle find first rowid for the required ID=1 and then make range index scan until it finds the last rowid for the ID=1(use in ‘where’ clause of the query). After that, oracle used this rowid to read blocks.

**2.4. Task 5: Index Skip Scan**





In the creation of composite index the order of columns on which the index is based is very important. For correct using of the composite index in the ‘where’ clause you should use columns in the order in which the columns are used to create the index.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scan type | Count of Rows | Consistent gets | Cost | Description |
| Index Skip Scan | 1 | 13 | 2 | The initial column of the composite index is not specified in the query and the index is divided into several smaller indexes and this indexes using to find data. |
| Full Scan | 1 | 18 | 3 | This type of scan is slower than Index Skip Scan one because of that Oracle reads the whole table. |