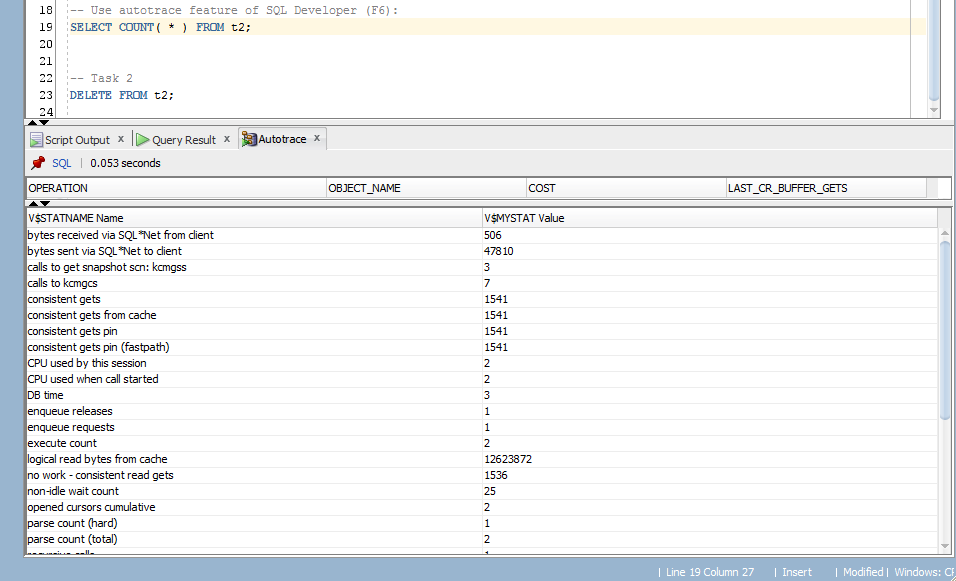
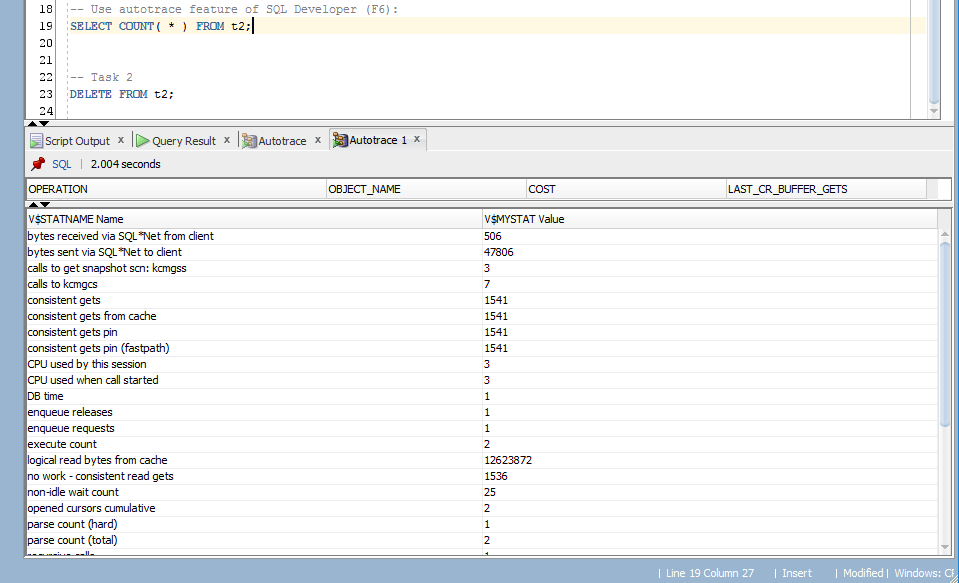
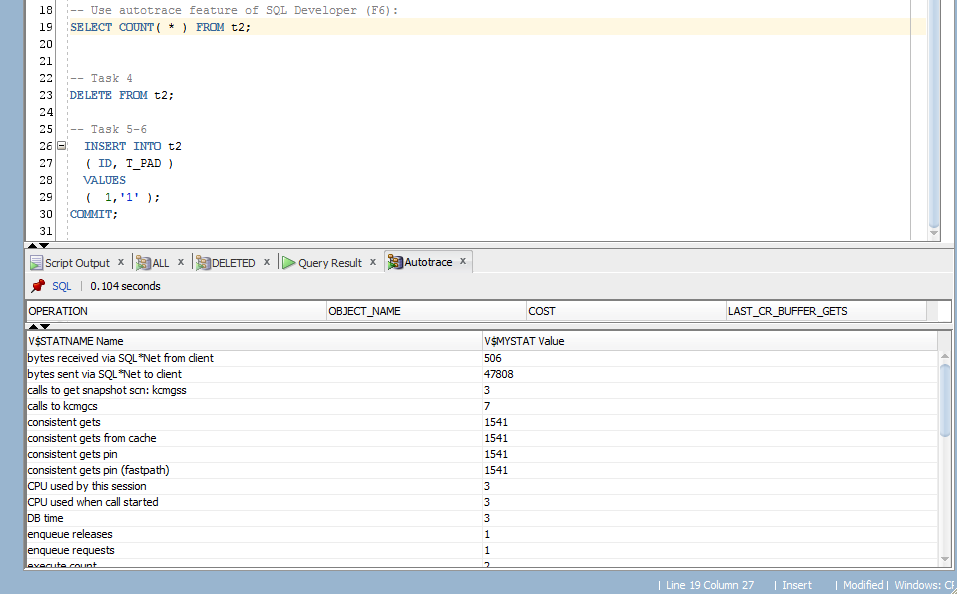
**Task 1 - Autotrace**



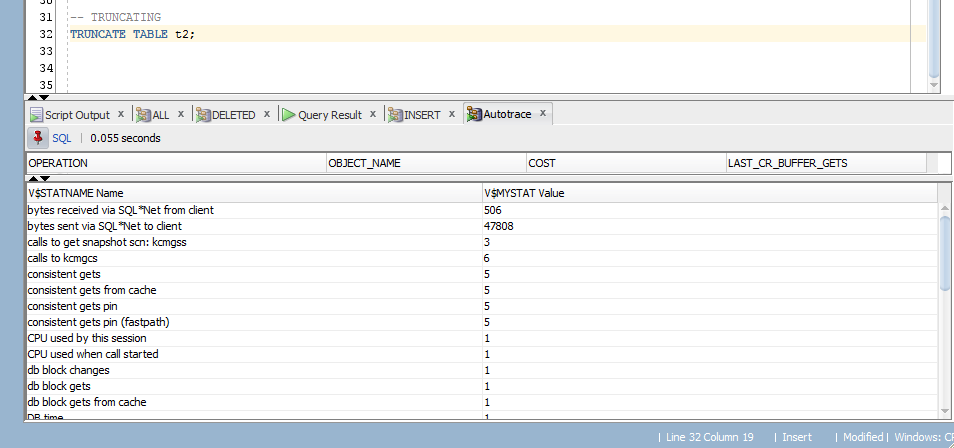
After DELETE



After INSERT



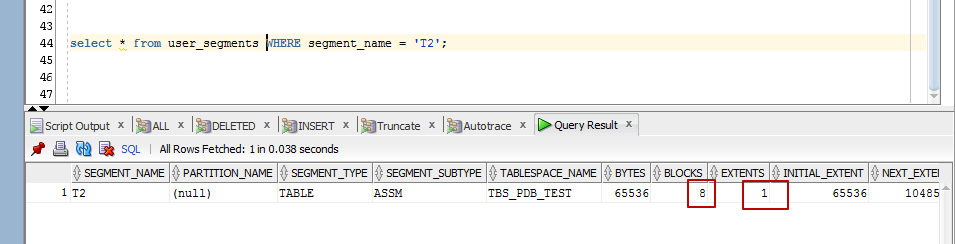
After TRUNCATE



Statistics

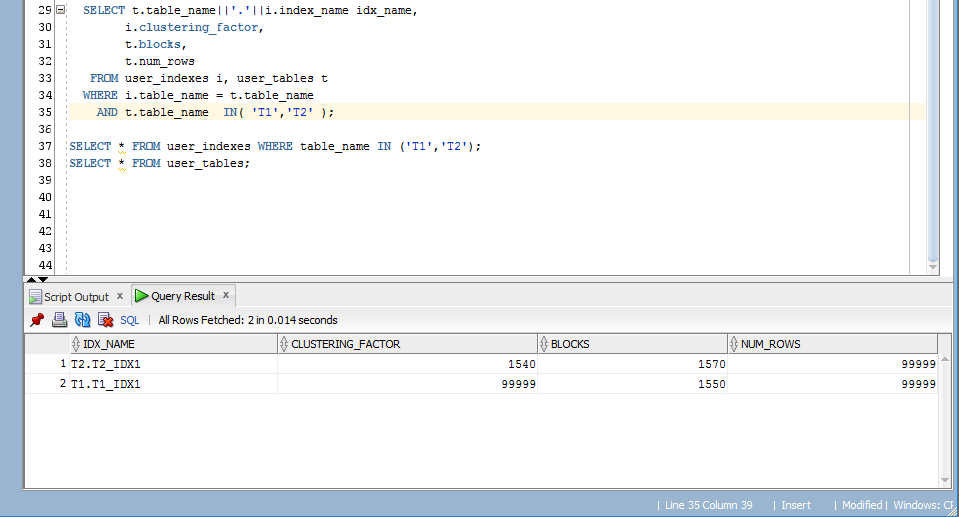
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1539 | ALL |
| 2 | 1664 | 0 | 0 | 1541 | DELETE |
| 3 | 1664 | 1 | 1 | 1541 | INSERT |
| 4 | 8 | 0 | 0 | 3 | TRUNCATE |

Statistics shows that in DELETE step all used blocs have decreased to 0 with the number of rows. The . Increase in Consistent gets could be caused by Oracle metadata usage. TRUNCATE had decreased the number of blocks to 8. This is the number of blocs in one extent:



**Index Scan Types**

**Task 2: Index Clustering Factor**



Index clustering factor shows the index stats. These stats helps to indicate the amount of order of the rows in the tables, which are based on the values of the index. The less the Clustering factor is, the better performance it will provide.

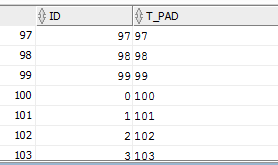
In our Example we got

* clustering factor = 1540 for “T2”

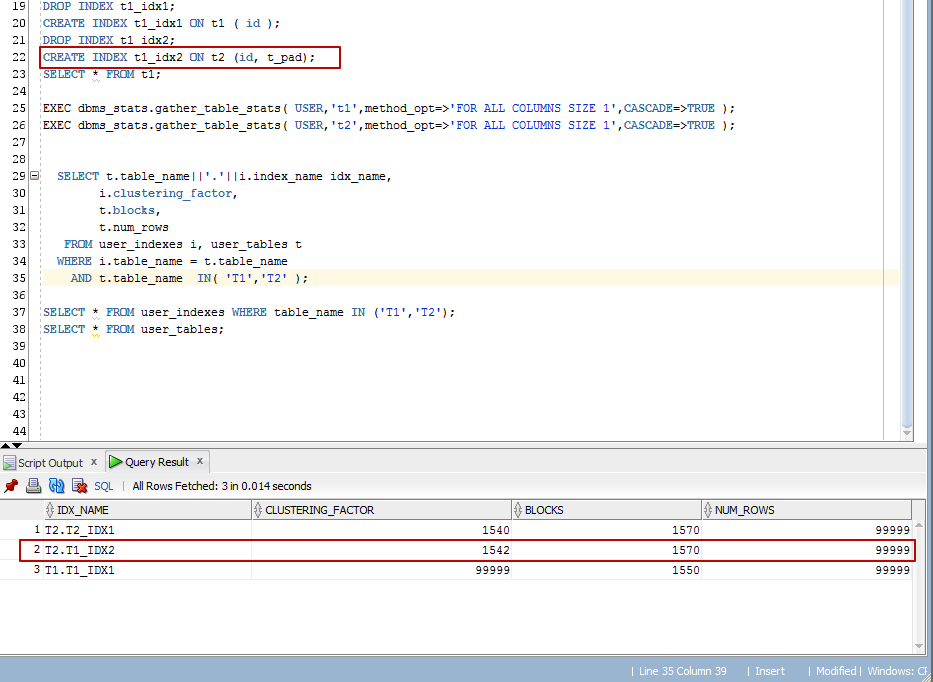
small clustering factor that is pretty good for our table

* clustering factor = 99999 for “T1”

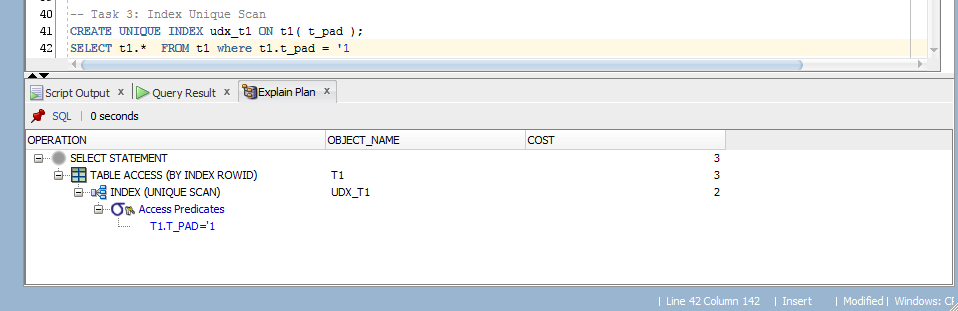
Very high rate that had appeared because of non-distinct Index values in “T1”. “T1” includes repeating ids from 1 to 99 till all 99999 rows, so ids are repeated 1 thousand times.



So as to provide better performance in this example we can use include two rows into index and that what we get:

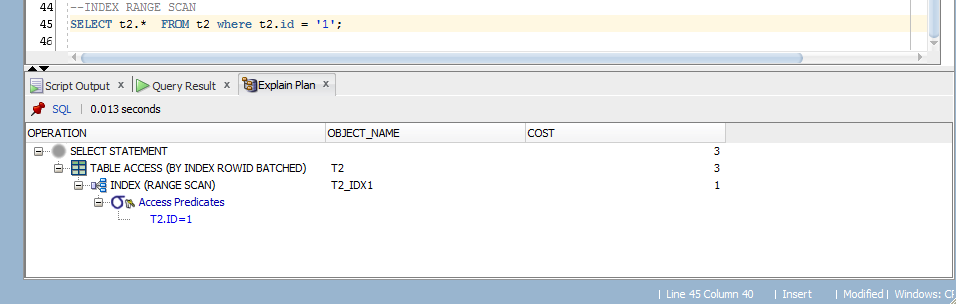


**Task 2: Index Unique Scan**



Oracle reads the block in order to find a specified key. It stops processing when the first record is found in the table, no matter there are any other values in the table with the same key. The database obtains the rowid from the index entry, and then retrieves the row specified by the rowid.

**Task 4: INDEX RANGE SCAN**



Index Range Scan moves backward or forward through the leaf blocks of the tree. Starting with the first leaf, provides index the info about rowed and data table block and moves futher to the next leaf until all index entries will be read.

**Task 5: Index Skip Scan**

Task 5 starts with correcting a mistake in statement 2:

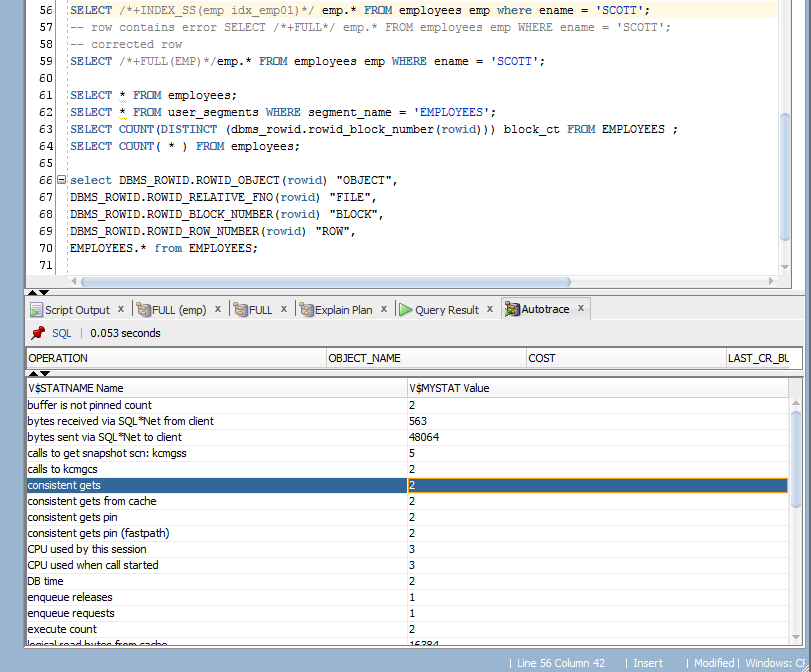
* *SELECT /\*+FULL\*/ emp.\* FROM employees emp WHERE ename = 'SCOTT'; (v1)*

Where hint /\*+FULL\*/ is not fully correct because of missing table alias, that should be added to the FULL in the way like this:

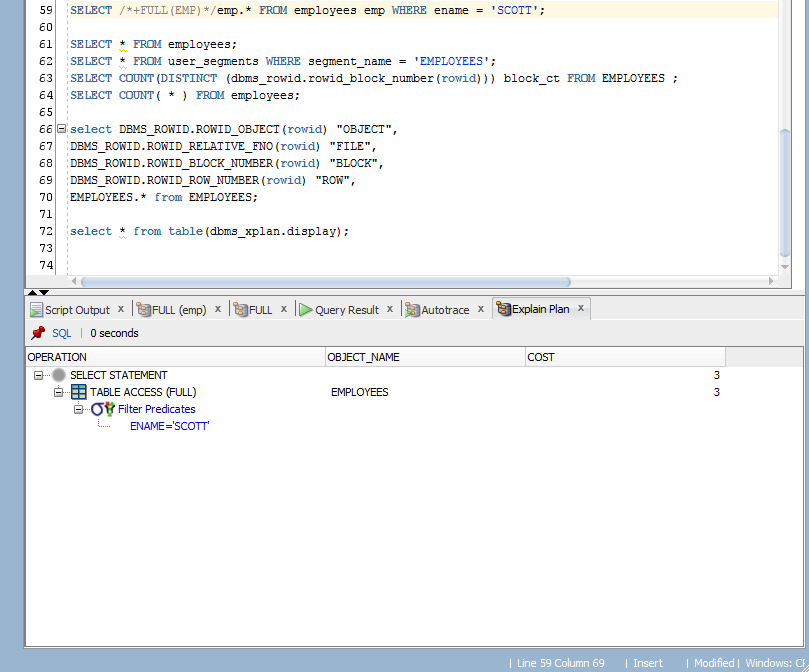
* *SELECT /\*+FULL(EMP)\*/emp.\* FROM employees emp WHERE ename = 'SCOTT' (v2);*

It makes Oracle use Full Table scan instead of Skip Scan that activates in the v1 of the statement.

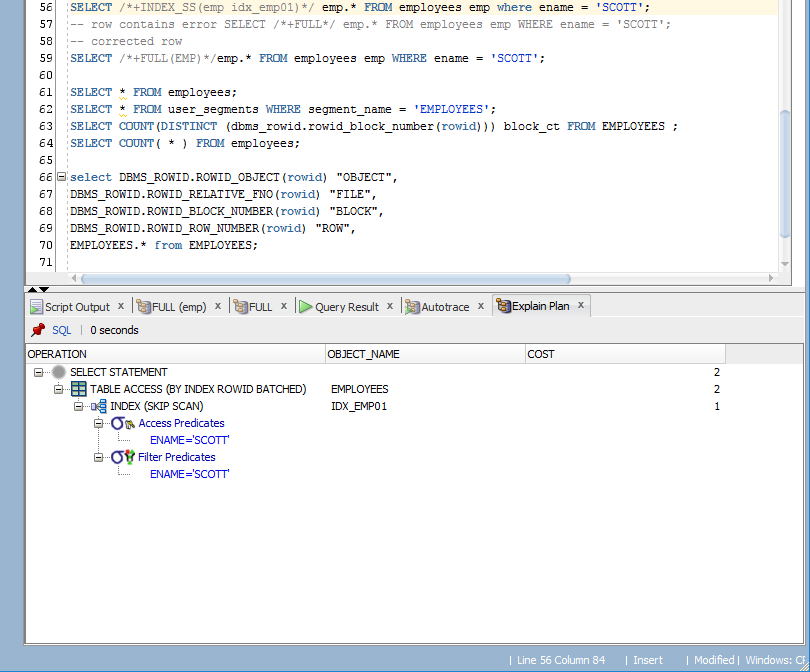
Tracing and execution **Skip Scan**

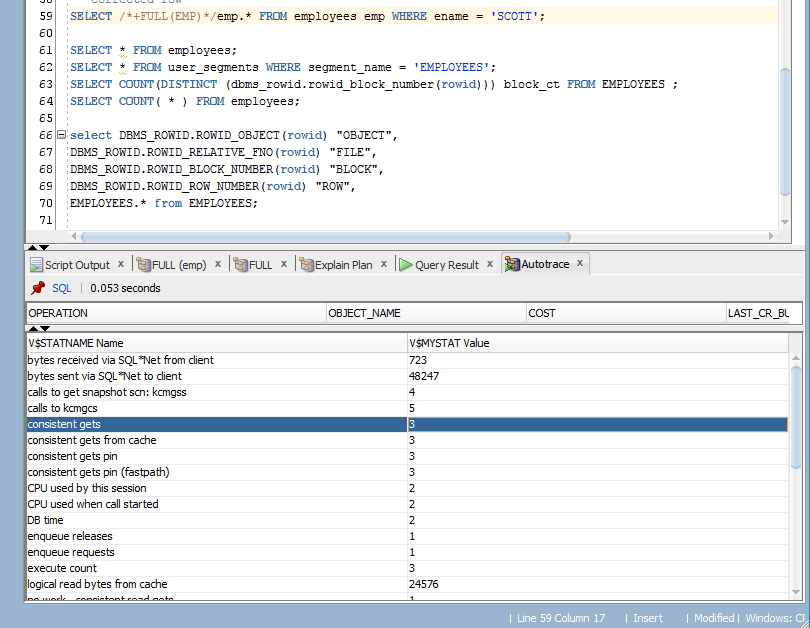


Tracing and execution **Full Table Scan:**



As a result we get better performance using **index skip scan** than **full table scan**.



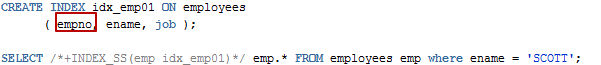


**Index skip scan** is used in 2 cases:

1. Composite key leading column nonspecified in predicate;
2. Few distinct values exist in the leading column and many distinct values exists in nonleading key of index.

The way of Index scan to perform he’s job is to split a composite index into smaller parts – subindexes – number of distinct values in the leading columns.

So we have exactly the same situation as described in the 1st case, where there is no empno in the where clause:



Talking about Full Table Scan, it should be said that it reads every formatted block under the high water mark and gets worse results in costs compared to **Index Skip Scan**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 8 | 1 | 1 | 2 | Skip Scan |
| 2 | 8 | 1 | 1 | 3 | Full Table Scan |
|  |  |  |  |  |  |
|  |  |  |  |  |  |