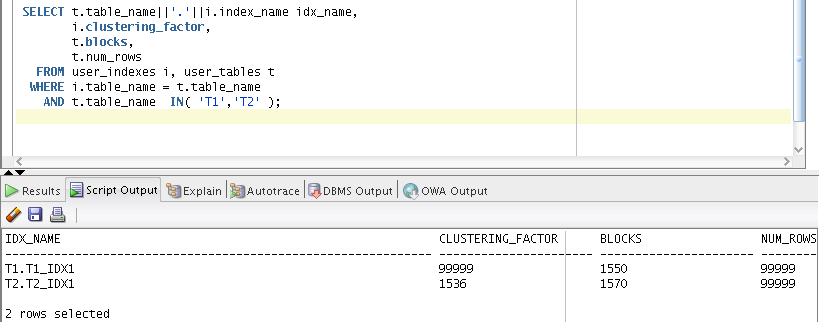
# Table access full scan

**Task Results:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 9999 | 1618 | After inserting of data, we can see number of rows and consistent gets, required to read data |
| 2 | 1664 | 0 | 0 | 1618 | After deleting all data from table we should steel read all blocks. And in block we read information up to HWM(it is not changed, when we delete data) |
| 3 | 1664 | 1 | 1 | 1618 | We insert 1 row, but HWM are not updated, so we read all data blocks to find one row as well as in case #3☹ |
| 4 | 8 | 0 | 0 | 5 | After truncate situation changed. We delete all(one) rows from table, update HWM, delete all unused blocks from segment where we store table(except 8, for reserve as I think) |

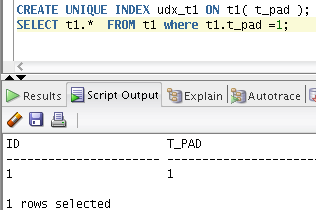
# Index Scan types

****

Clustering factor column is a measure of how organized the data is compared to the indexed column. It depends of the range of values, which used in index. For all rows in table 1, ID column have unique values, but for table 2 100 rows have repeated ID. It allows organizing index (B\*tree) better. Index in table 2 is smaller and on each petal of index we have information about 100 rows.

It is obvious that in execution Select clause filtered by IN we will see best performance with table2.

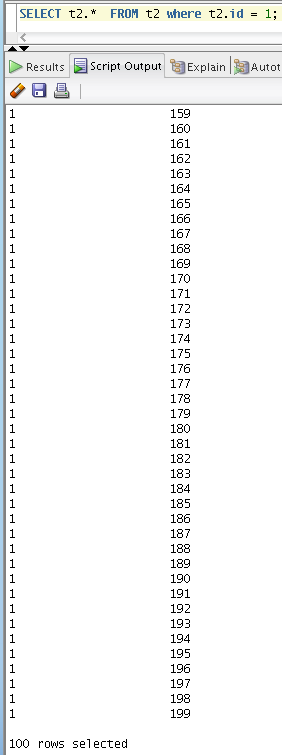
## Task 3: Index Unique Scan

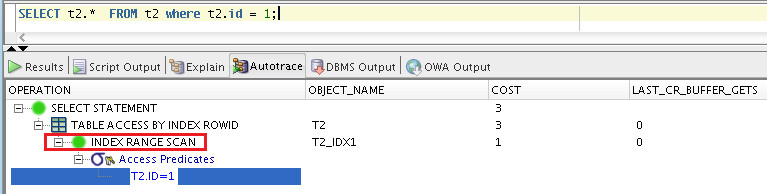


Oracle can use the unique index, knowing perfectly well there will be a maximum of one possible value or row. As we can see we need to find 1 row in where clause. When we found this unique row we don't need more to continue search. In other words an index unique scan stops processing as soon as it finds the first record because no second record is possible.

## 

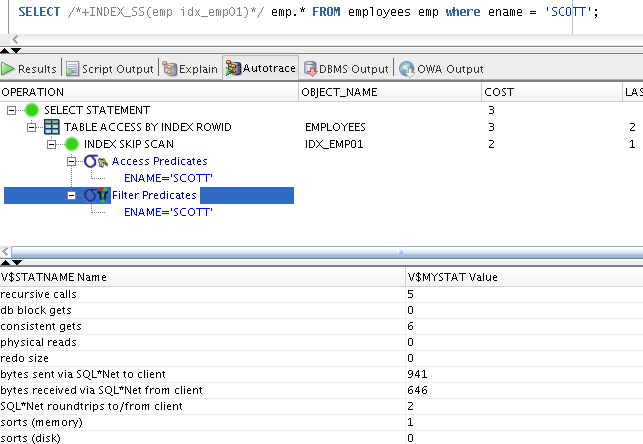
## Task 4: Index Range Scan

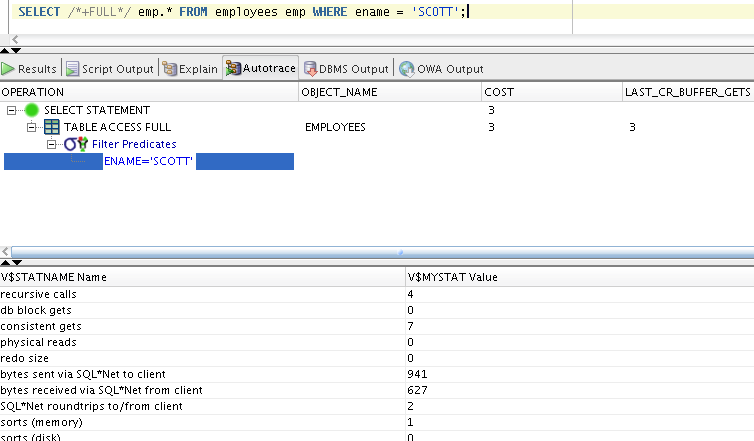
****



In where conduction we use ID field, which have index on each petal we have information about 100 rows and query returned a range of data.

## Task 5: Index Skip Scan





On the 2nd step we created composite index. Multi-column index splitting into smaller sub-indexes. This structure is effective only when we have not many distinct values in leading column and we use in where clause all columns, which is used for creating index.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 14 | 4 | 1 | 6 | /\*+INDEX\_SS(emp idx\_emp01)\*/ |
| 2 | 14 | 4 | 1 | 7 | /\*+FULL\*/ |

Using skip-scan will be better (analyze Consistent gets).

Using Full scan we read the index structure in following order : Empno -> ename -> job

Skip scan allows us read index like this: Empno -> ename .

Field job don’t used in our query.