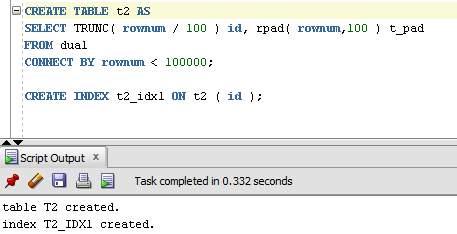
# Table access full scan

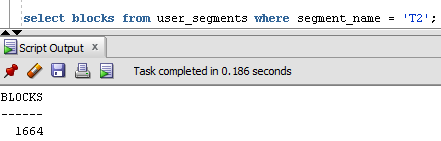
## Task 1: Full Scans and the High-water Mark and Block reading

Step 1:Step 2:

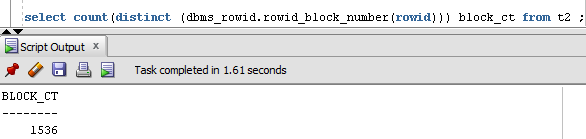


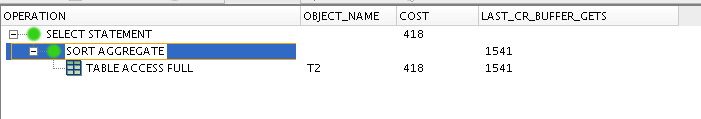
**Step 3:**

Block count:

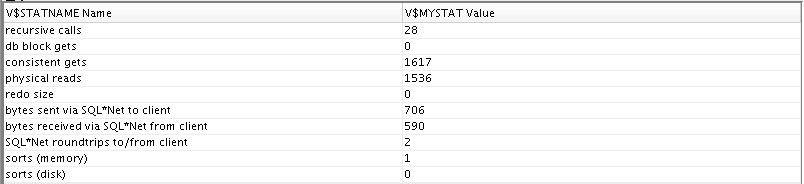


Used Block Count:





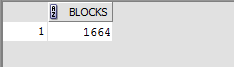
Explain Plan:



Step 4: Delete All Rows from table

DELETE FROM t2;

**Step 5:** Repeat Step 3 and collect results.



Used Block Count



Step 6: Insert 1 row

**Step 7:** Repeat Step 3 and collect results.

Block Count



Used Block Count



Step 8: Truncate Table

**Step 9:**  Repeat Step 3 and collect results.

Block Count



Used Block Count



**Task Results:**

Expected:

Summary table with all result and text description of analyses this results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| create | 1664 | 1536 | 99.999 | 1615 | Our table |
| delete | 1664 | 0 | 0 | 1615 | Our blocks is empty and unused after deletion. But when a full scan operation occurs, all blocks up to the high-water mark will be read in and scanned, even if they are empty. |
| insert | 1664 | 1 | 1 | 1615 | Still the same count of block, not matter of using count. |
| truncate | 8 | 0 | 0 | 5 | Count changed after truncate |
|  |  |  |  |  |  |

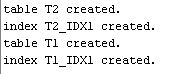
# Index Scan types

## Task 2: Index Clustering factor parameter

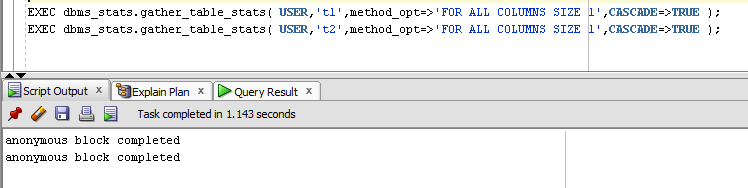
Step 1: Create table t2 as on task 1 step 1-2

Step 2: Create table t1 as listed below

Step 3:



Step 4: Calculate statistic for both tables:



**Step 5:** Select Clustering Factor



**Task Results:**

Expected:

* Screenshot of the step 5;
* Description of the parameter clustering factor;
* Explanation: why for indexes t1\_idx1 and t2\_idx1 we have different values ;
* Which Index has best selective performance in execution Select clause filtered by IN ( , list of values, );

Clustering factor marks index performance. For each entry in the index Oracle compares the entry's table rowid block with the block of the previous index entry. If the block is different, Oracle increments the clustering factor by 1.

In ‘t1’ every value of ‘id’ is unique(then clustering factor=rownum), in t2 there are lot of matching ‘id’. Then when we search data by ‘id’ in ‘t1’ we have to read every index row, and in ‘t2’ we get a lot of data by the every ‘id’

Select \* FROM t1 WHERE ID IN(10,25,35,50) COST=413

Select \* FROM t2 WHERE ID IN(10,25,35,50) COST=12

T2 index is much better in queries filtered by ‘IN’ clause

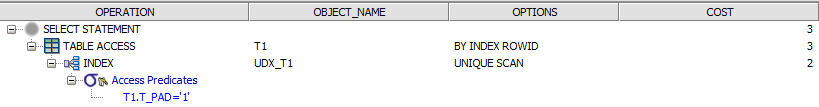
## Task 3: Index Unique Scan

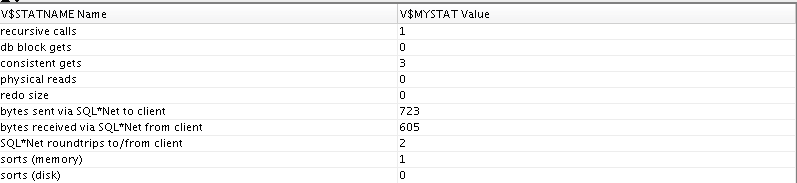
Step 1:

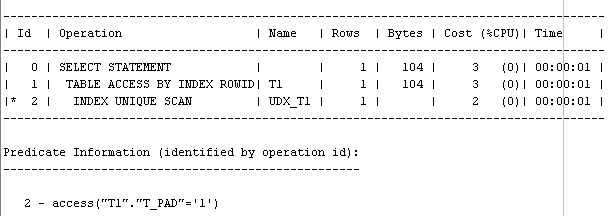
# CREATE UNIQUE INDEX udx\_t1 ON t1( t\_pad );

**Step 2**

# SELECT t1.\* FROM t1 where t1.t\_pad = '1';







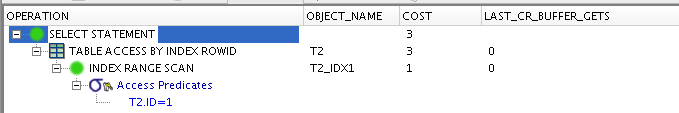
**Index Unique Scan** - is chosen when a predicate contains a condition using a column defined with a UNIQUE or PRIMARY KEY index. These types of indexes guarantee that only one row will ever be returned for a specified value.(c)

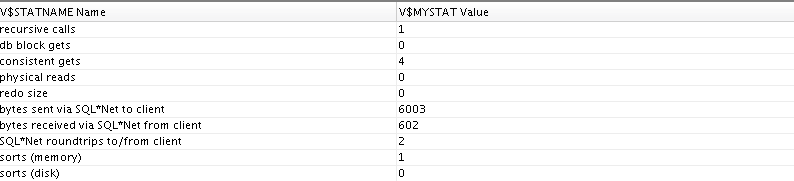
So we’ve built unique index, because it great useful on this table where ‘id’ is unique. So I suppose oracle simple read index B’tree until it finds value we’re looking for.

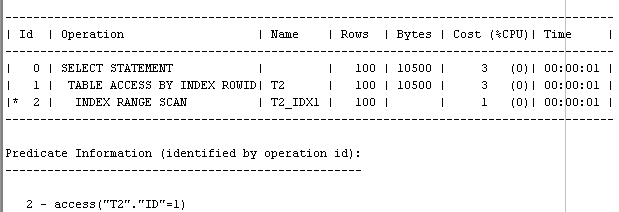
## Task 4: Index Range Scan

**Step 1:**

# SELECT t2.\* FROM t2 where t2.id = '1';





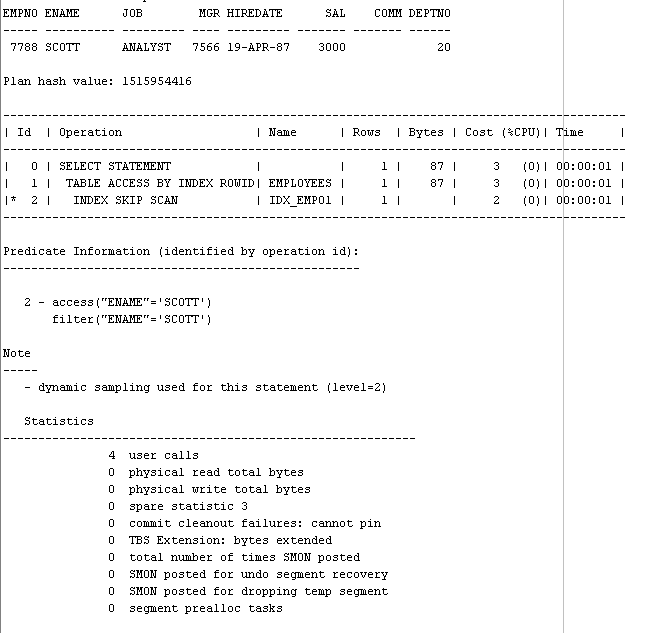


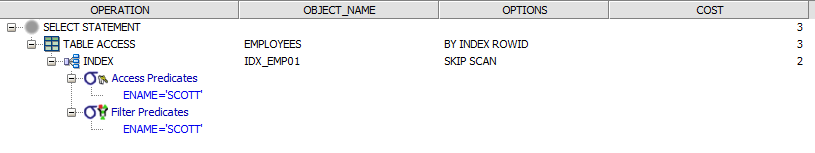
**Index range scan** - is chosen when a predicate contains a condition that will return a range of data. It is so because every single id matches 100 times, and return 100 rows. So we searching our data by the range of our ‘id’.

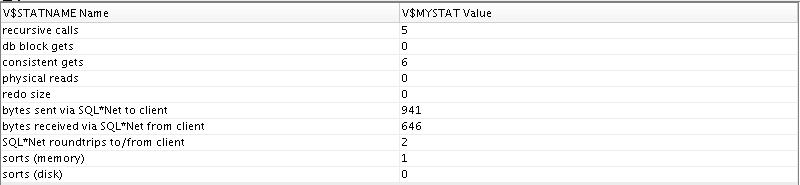
## Task 5: Index Skip Scan

**Step 3:**  Get trace and statistic of explain plan

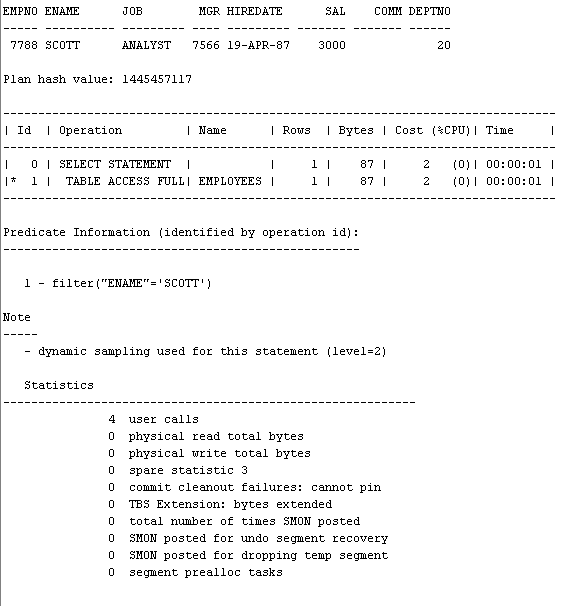
# SELECT /\*+INDEX\_SS(emp idx\_emp01)\*/ emp.\* FROM employees emp where ename = 'SCOTT';

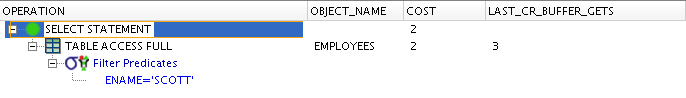


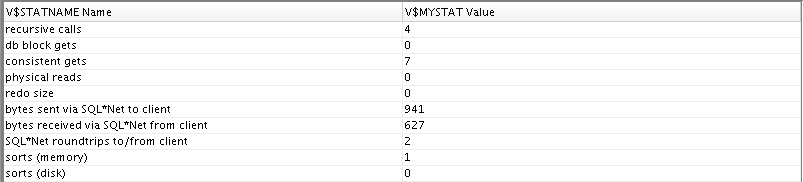




# SELECT /\*+FULL\*/ emp.\* FROM employees emp WHERE ename = 'SCOTT';







**Task Results:**

Expected:

* 2 Screenshots of the step 3;
* Description of process: How oracle analyses index that was created on step 2;
* Summary table with all result and text description of analyses this results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
|  | 8 | 1 | 1 | 6 | Index skip can |
|  | 8 | 1 | 1 | 7 | Table access full |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Index Skip Scan** - A skip scan works by logically The number of logical subindexes is determined by the number of distinct values in the leading columns of the index. Therefore, the more distinct the leading columns are, the more logical subindexes would need to be created.(c)

We created index on 3 columns. Then we can split a multi-column index into smaller subindexes. I suppose oracle scans our data only by index on columns by which we filter in “where” clause.( ename in that case) and skips all other indexes.

Full access select works by the same time, but seems it is because our table is small and has only 14 rows.