|  |
| --- |
|  |
| Oracle Data Access and Optimizer |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | Vitaliya Adamchuk | 08-11-2017 |  |  |

Contents

[**1.** **Full Scan, High-Water Mark and Consistent Gets** 3](#_Toc497935085)

[**2. Index Clustering Factor** 4](#_Toc497935086)

[**3. Index Unique Scan** 5](#_Toc497935087)

[**4. Index Range Scan** 6](#_Toc497935088)

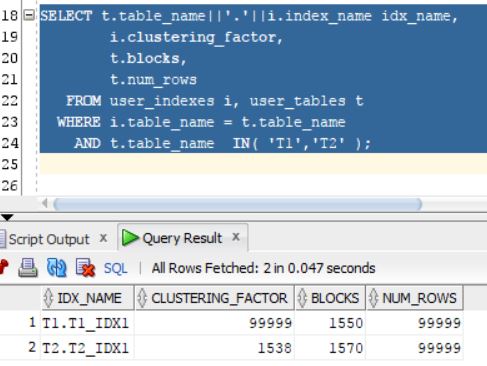
[**5. Index Skip Scan** 7](#_Toc497935089)

## **Full Scan, High-Water Mark and Consistent Gets**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1 | 1664 | 1536 | 99999 | 1541 | Creating table with 99999 number of rows. There are some unused blocks |
| 2 | 1664 | 0 | 0 | 1619 | Deleting all rows from the table, but blocks are still dirty |
| 3 | 1664 | 1 | 1 | 1542 | Inserting one row, but blocks are still not empty. But in fact there is only one row |
| 4 | 8 | 0 | 0 | 5 | Table were truncate and blocks are cleaned |

The autotrace report is generated after successful SQL DML statements. It is useful for monitoring and tuning the performance of these statements.

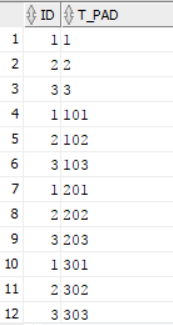
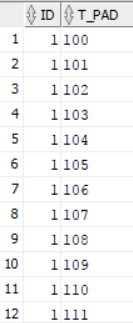
## **2. Index Clustering Factor**



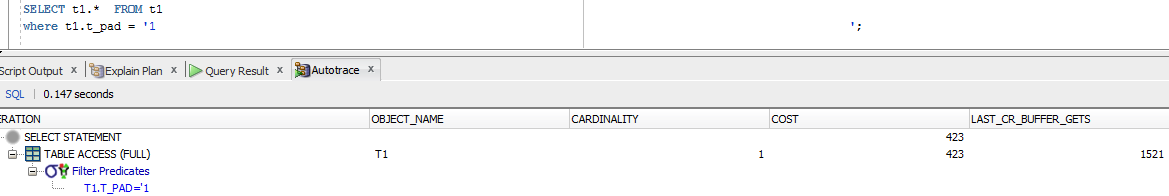
The index clustering factor is a measure of how many rows the database would perform if it were to read every row in that table via the index in index order or how many time the database should go to the datablocks to retrieve all needed data.

In comparison table 1 with table 2, table 2 has a better performance because the rows are sorted by index key, so the next row from the index key will be the next row in the table block. So it is not needed to go through the whole blocks.

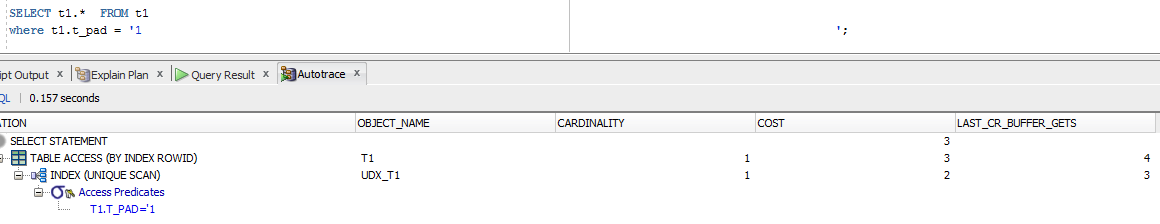
Table 1 Table 2

## **3. Index Unique Scan**

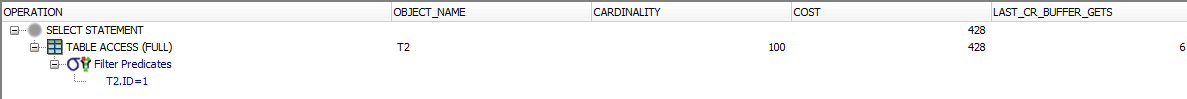


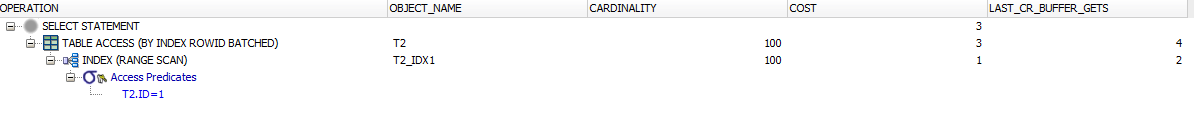
Without the index, database will go through the whole values regardless whether the needed values were found or not



With unique index when the database find the needed valie in the tree, its stops searching. That’s why the cost in the second case is much smaller the in the first one.

## **4. Index Range Scan**

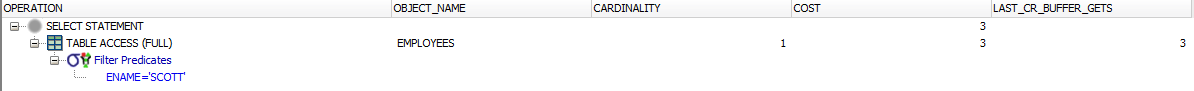


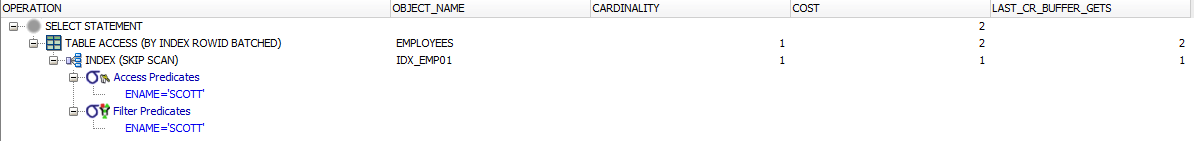


With index range scan there is a condition that return a range of values

It is not necessary to search through the whole table because it is begin from the root block to the first leaf block containing an entry matching the specified condition. Here he retrieves a RowId and then retrieves a table data block. After that the next rowed will be retrieved. This back-and-forth between the index leaf blocks and the data blocks will continue until all the matching index entries have been read. If the range of entries matching the condition is large enough, it is likely that more than one leaf block will have to be accessed, so the next leaf block needed can be read using a pointer stored in the current leaf block that leads to the next leaf block (there’s also a pointer to the previous leaf block). Since these pointers exist, there is no need to go back up to the branch block to determine where to go next.

## **5. Index Skip Scan**





Here the skip scan is being used due to the fact that predicate contains a condition on non-leading column in an index and the leading columns are distinct.