Report

on the

U1M4.LW.Access and

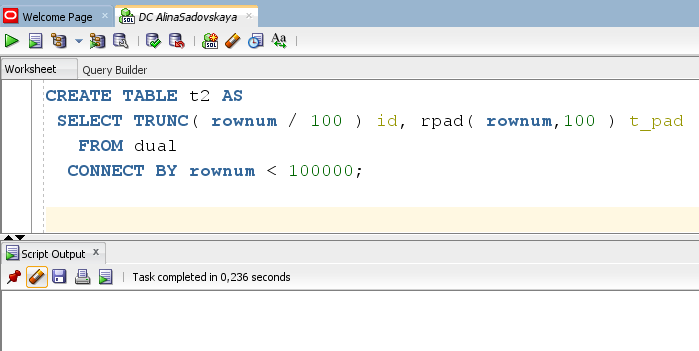
Join Methods Part 1

Alina Sadovskaya

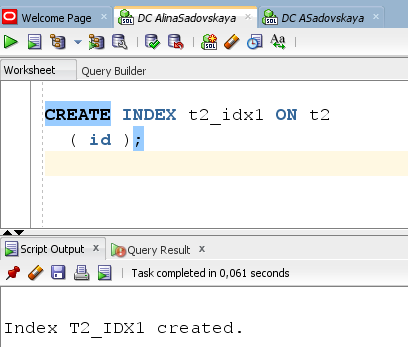
# 1. Table access full scan

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

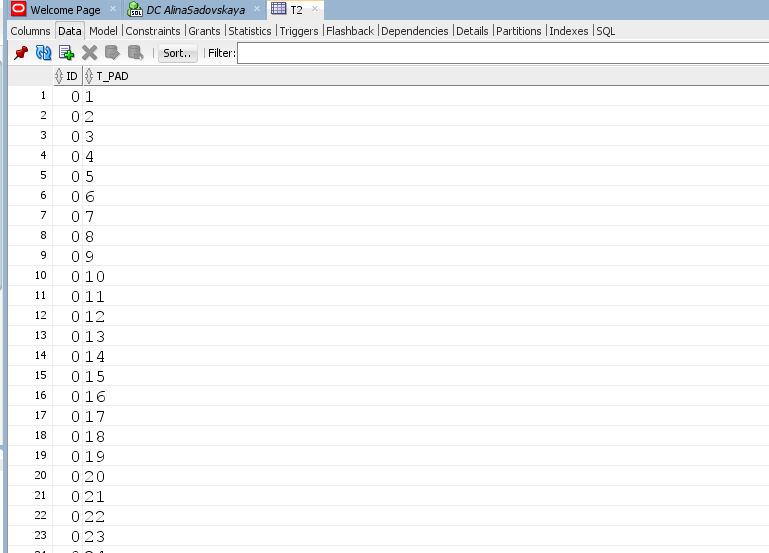
**Step 1:**

****

**Step 2:**

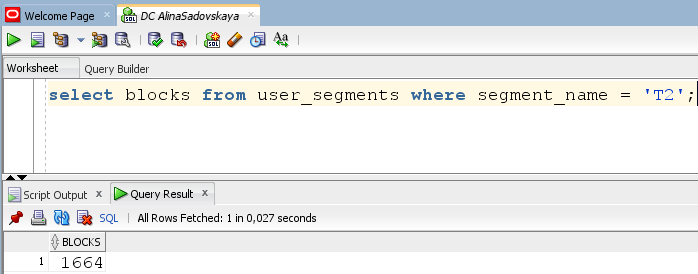


Example of a table:

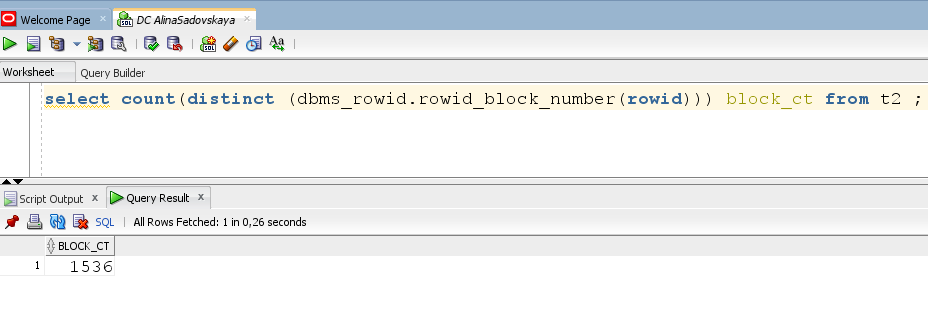


**Step 3:**

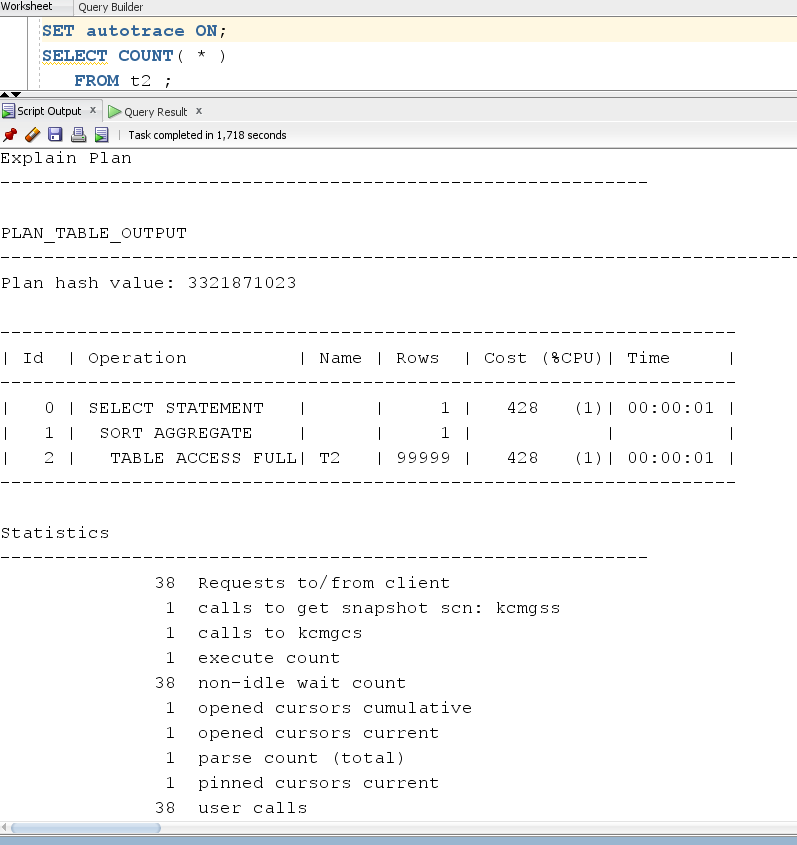
**Block count:**

**

**Used Block Count:**

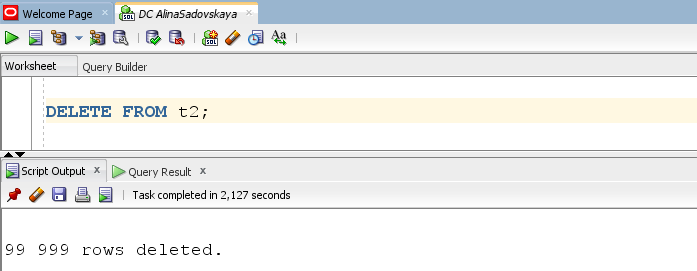


**Explain Plan:**



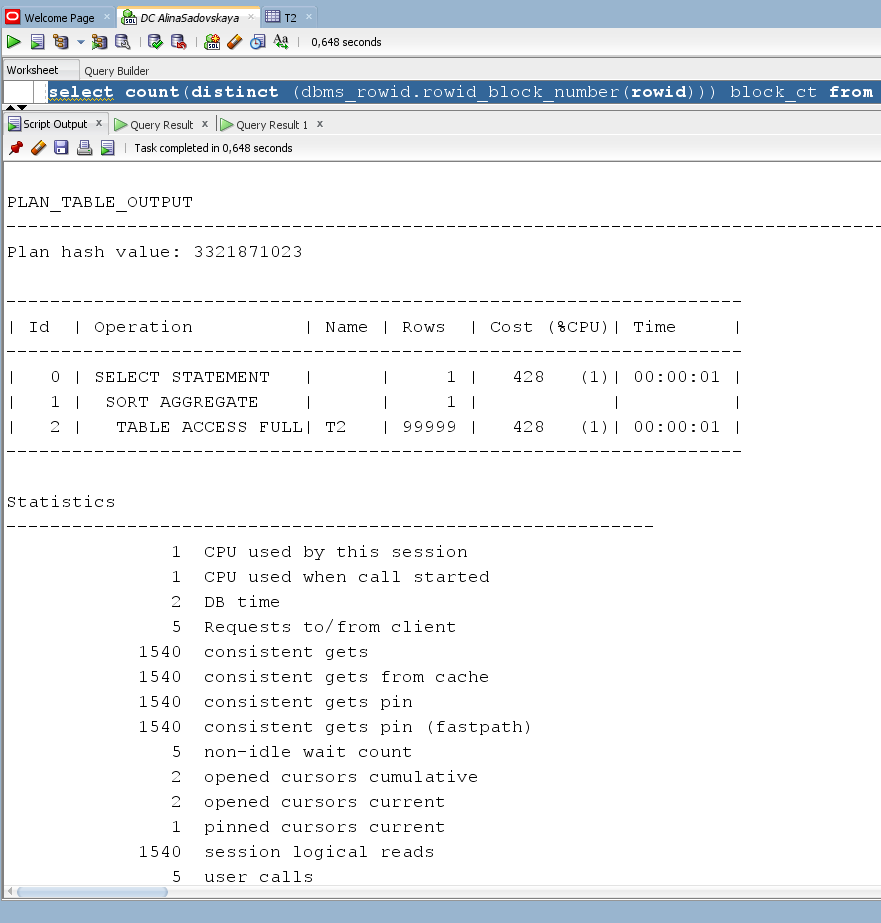
**Step 4:**

Delete All Rows from table



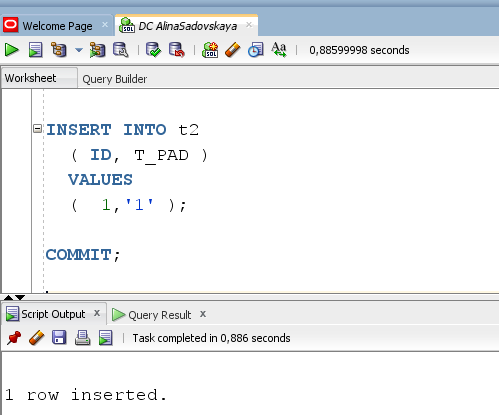
**Step 5:**

**Explain Plan:**

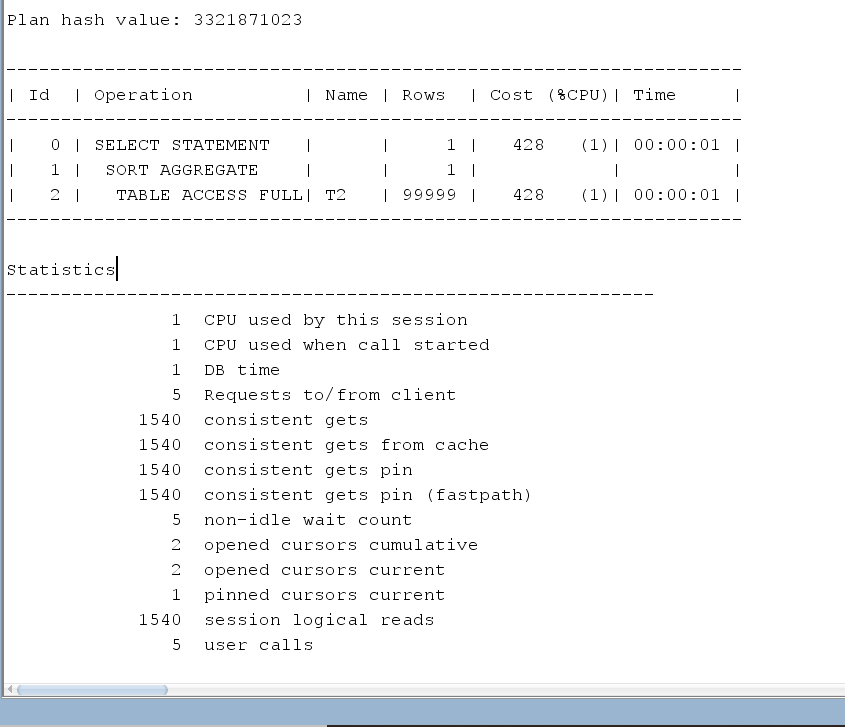


**Step 6:**

Insert 1 row:



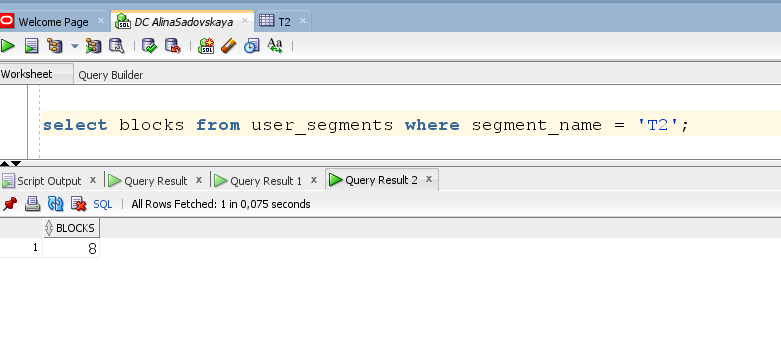
**Step 7:**

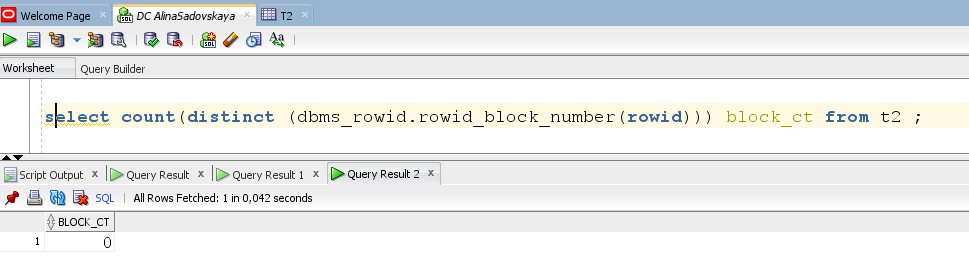


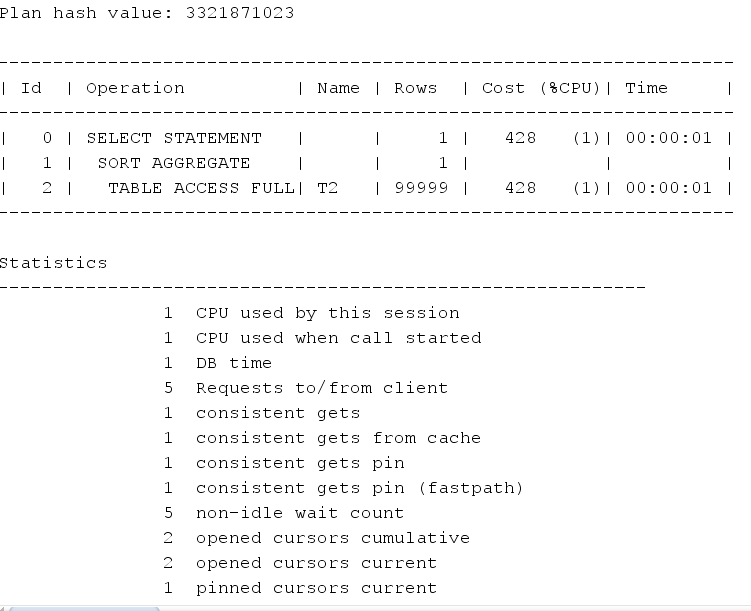
**Step 8:**

TRUNCATE TABLE t2

Statistics:







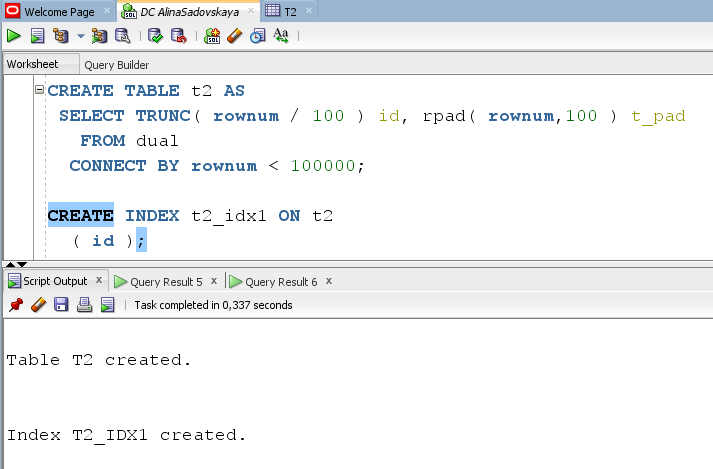
\*99999(statistics – without dropping WM)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 1-3 | 1664 | 1536 | 99999 | 1540 | The field “persistent gets” specifies the number of requests to the block. At this stage, the table was filled in and HWM was installed (the maximum number of rows is observed). |
| 4-5 | 1664 | 0 | 0 | 1540 | 'Consistent gets' has not changed, since the HWM remains the same(when performing the 'delete' operation, the object's statistics and allocated space are saved). |
| 6-7 | 1664 | 1 | 1 | 1540 | 'Consistent gets' has not changed, because the HWM remains the same(when performing the 'insert' operation (row\_count = 1), the maximum number of elements has not changed(max\_row\_count = 9999)). |
| 8-9 | 8 | 0 | 0 | 1 | 'Consistent gets' has changed, since the HWM has also changed(when performing the TRUNCATE operation, all table data is freed, so TRUNCATE deletes all statistics, allocated space, and resets the HWM). |

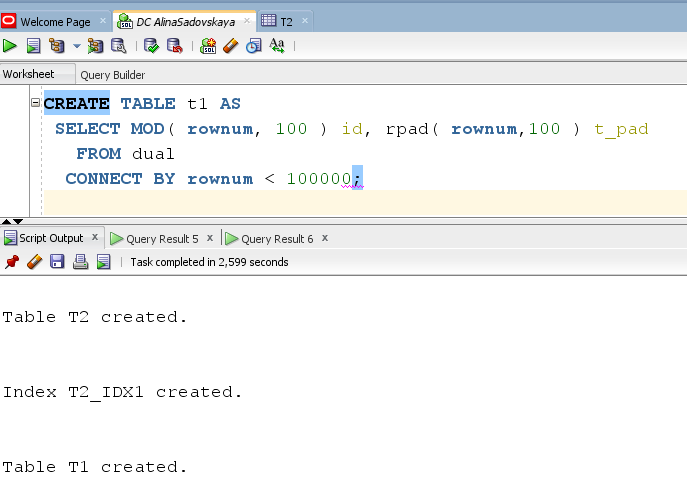
# 2. Index Scan types

2.1. Task 2: Index Clustering factor parameter

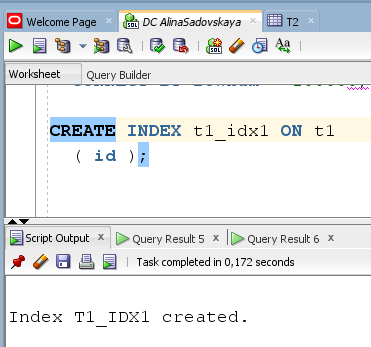
**Step 1:** Create table t2 as on task 1 step 1-2



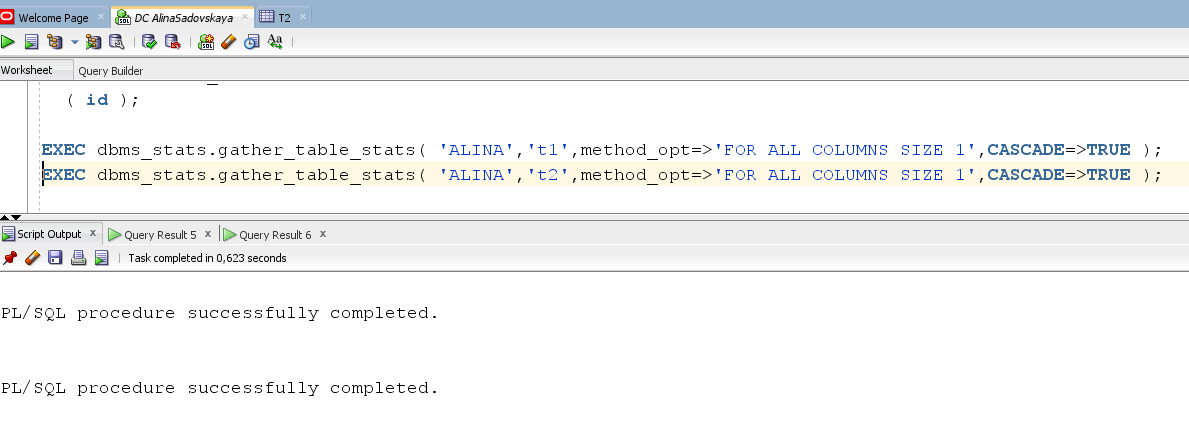
**Step 2:** Create table t1 as listed below



**Step 3:**



**Step 4:** Calculate statistic for both tables:



**Step 5:** Select Clustering Factor

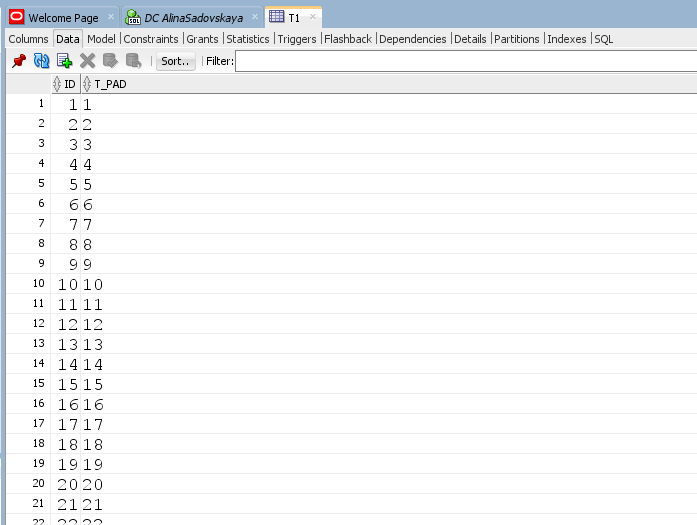
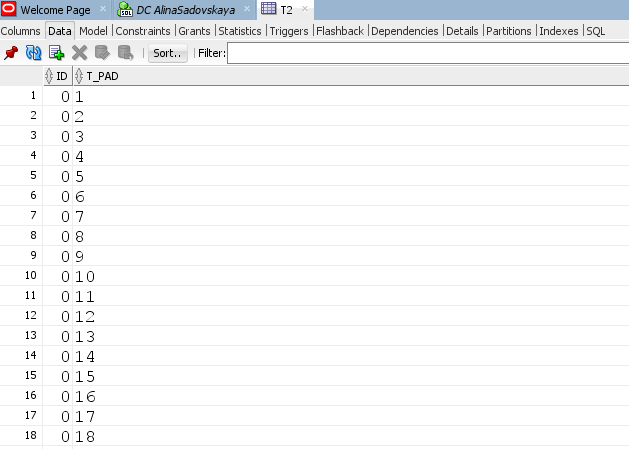
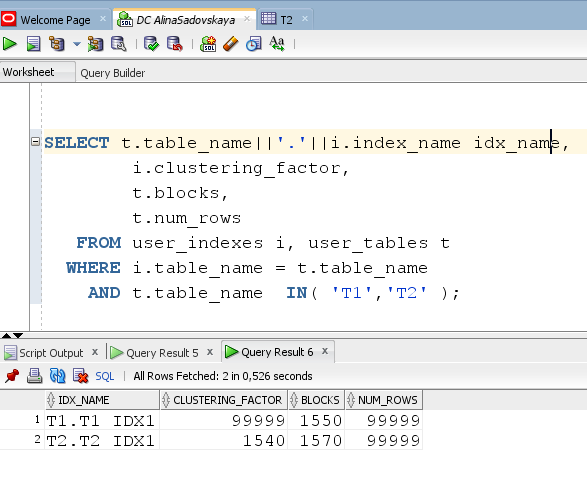


Table T1

Table T1 Table T2



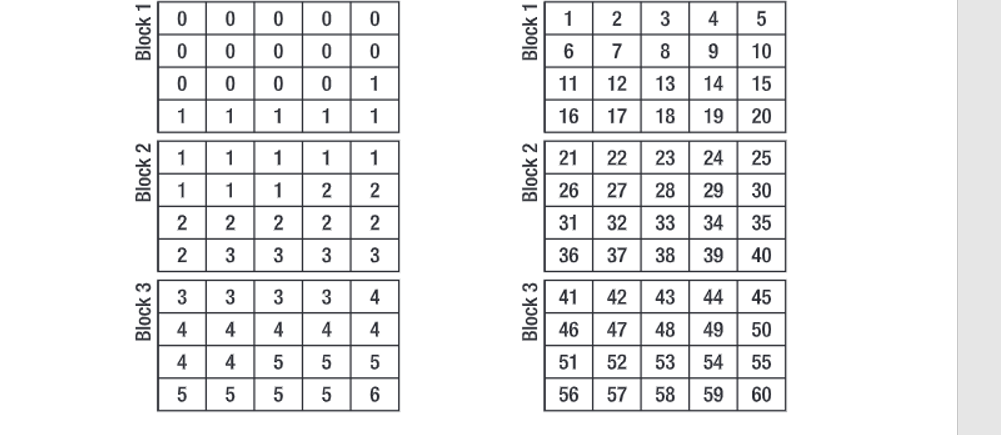
The Oracle Database Reference guide describes the purpose of the CLUSTERING \_ FACTOR column as follows. Specifies the order of rows in the table based on index values.

• If the value is close to the number of blocks, the table is very well ordered. In this case, index entries in a single leaf block tend to point to rows in the same data blocks.

• If the value is close to the number of rows, the table is ordered very randomly. In this case, it is unlikely that index entries from the same leaf block will point to rows in the same data blocks. The clustering factor can also be considered as a number representing the number of logical I / o operations in a table that must be performed to read the entire table through the index.

In other words, CLUSTERING \_ FACTOR is a sign of how the table is ordered in relation to the index itself.

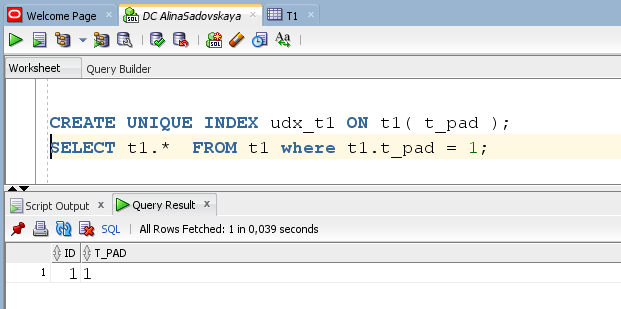
As you can see from the following image, table T2 is well ordered, and table T1 is slightly worse (CLUSTERING \_ FACTOR\_T1 > CLUSTERING \_ FACTOR\_T2)

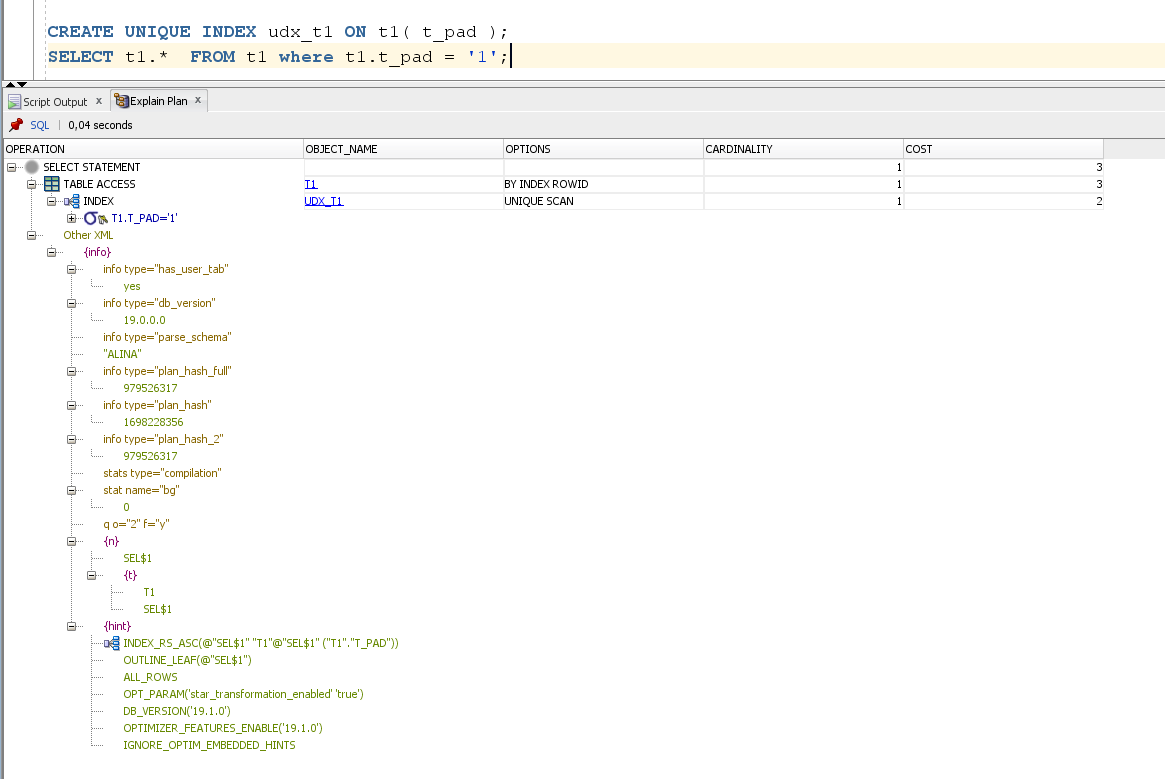


* We can conclude that Bitmap Index has best selective performance in execution Select clause filtered by IN ( , list of values, ). Range Scan Unique will also give good results (if the indexes are unique).

## 2.2. Task 3: Index Unique Scan

**Step 1-2:**

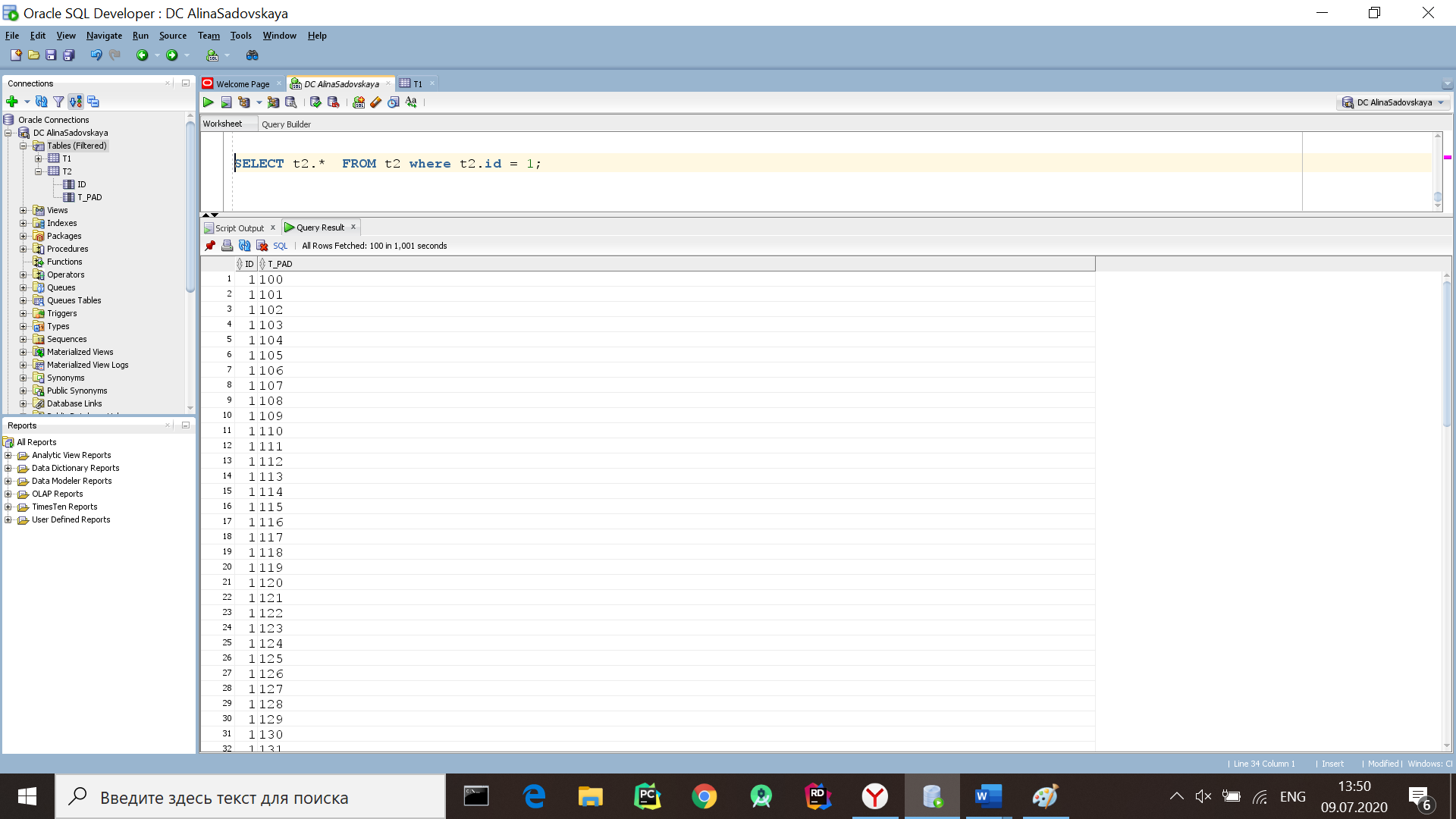


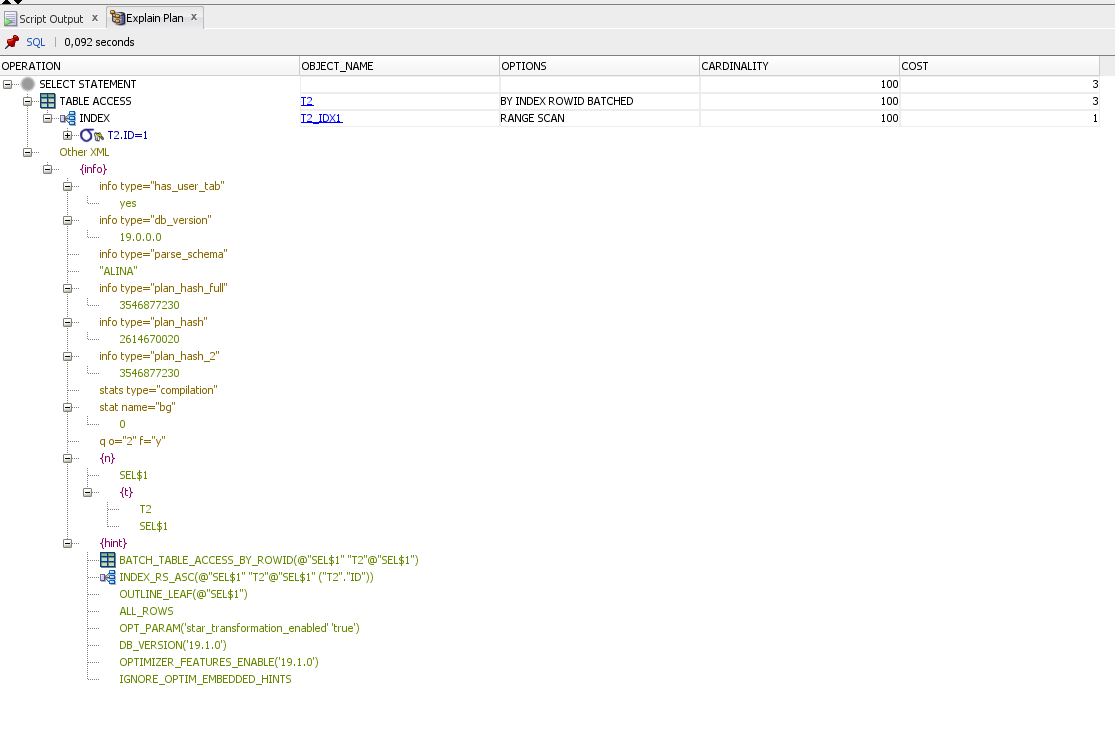


An index unique scan is chosen when a predicate contains a condition using a column defined with a UNIQUE or PRIMARY KEY index. These types of indexes guarantee that only one row will ever be returned for a specified value. In this cases, the index structure will be traversed from root to leaf block to a single entry, retrieve the rowid, and use it to access the table data block containing the one row. TABLE ACCESS BY INDEX ROWID step in the plan indicates the table data block access. The number of block accesses required will always be equal to the height of the index plus one unless there are special circumstances like the row is chained or contains a LOB that is stored elsewhere.

## 2.3. Task 4: Index Range Scan

**Step 1:**



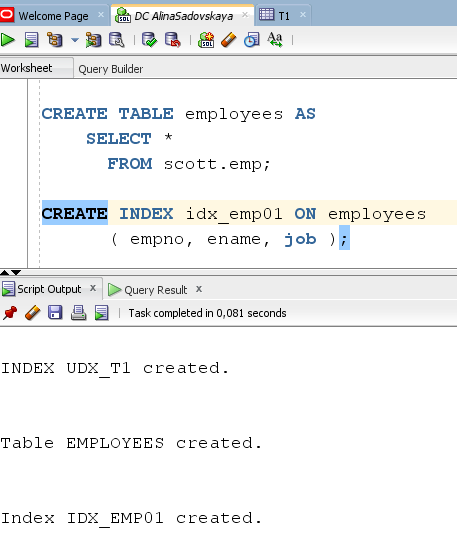


An index range scan is chosen when a predicate contains a condition that will return a range of data. The index can be unique or non-unique as it is the condition that determines whether or not multiple rows will be returned or not. The conditions specified can use operators such as <, >, LIKE, BETWEEN and even =. In order for a range scan to be selected, the range will need to be fairly selective. The larger the range, the more likely a full scan operation will be chosen instead.

A range scan will traverse the index structure from the root block to the first leaf block containing an entry matching the specified condition. From that starting point, a rowid will be retrieved from the index entry and the table data block will be retrieved (TABLE ACCESS BY INDEX ROWID). After the first row is retrieved, the index leaf block will be accessed again and the next entry will be read to retrieve the next rowid. This back-and-forth between the index leaf blocks and the data blocks will continue until all the matching index entries have been read. Therefore, the number of block accesses required will include the number of branch blocks in the index (this can be found using the blevel statistic for the index) plus the number of index entries that match the condition multiplied by two. You have to multiply by two because each retrieval of a single row in the table will require that the index leaf block be accessed to retrieve the rowid and then the table data block will be accessed using that rowid.

## 2.4. Task 5: Index Skip Scan

**Step 1-2:**

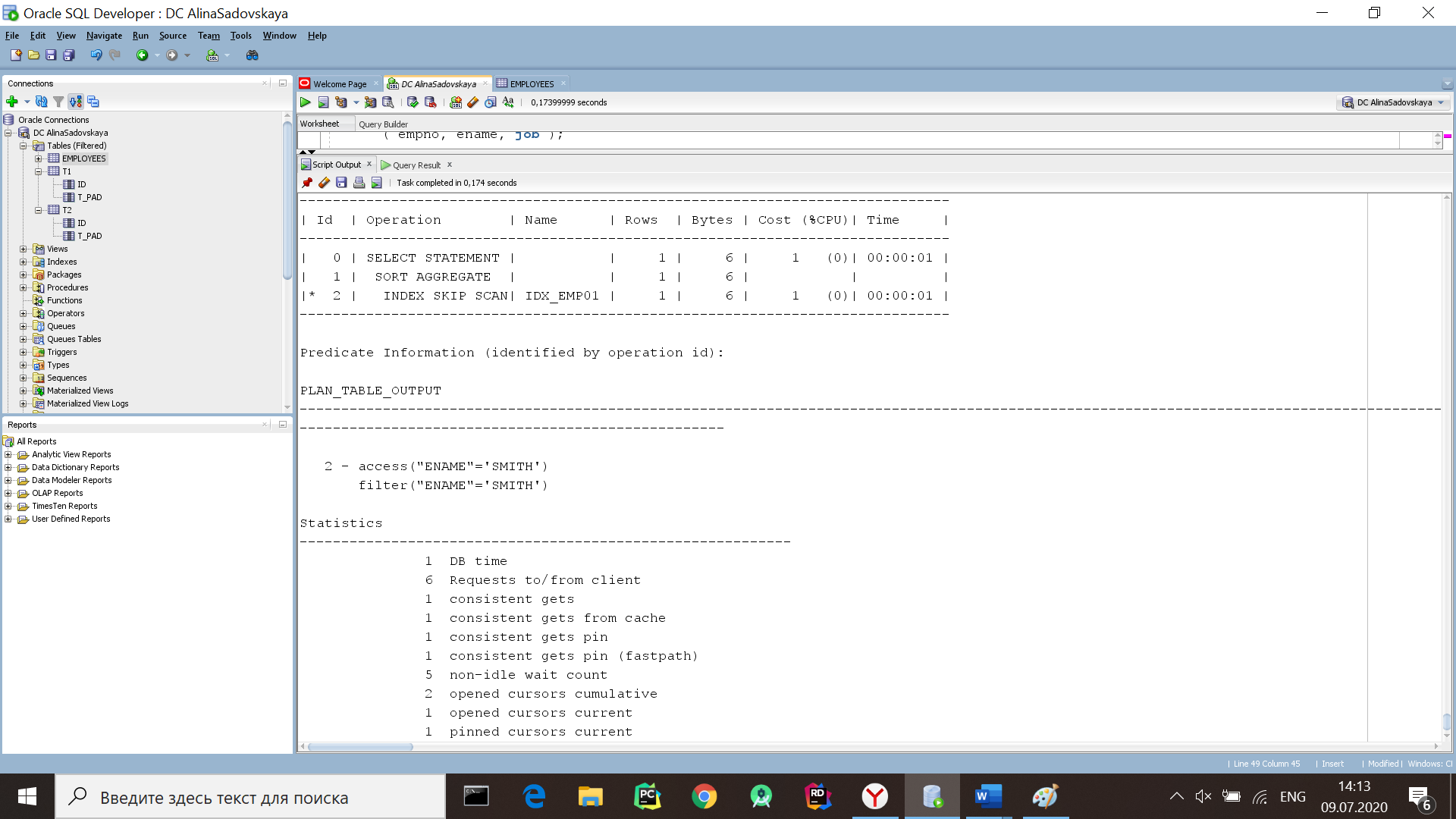


A skip scan works by logically splitting a multi-column index into smaller subindexes. The number of logical subindexes is determined by the number of distinct values in the leading columns of the index. Therefore, the more distinct the leading columns are, the more logical subindexes would need to be created. If too many subindexes would be required, the operation won’t be as efficient as simply doing a full scan. However, in the cases where the number of subindexes needed would be smaller, the operation can be many times more efficient than a full scan as scanning smaller index blocks can be more efficient than scanning larger table blocks.

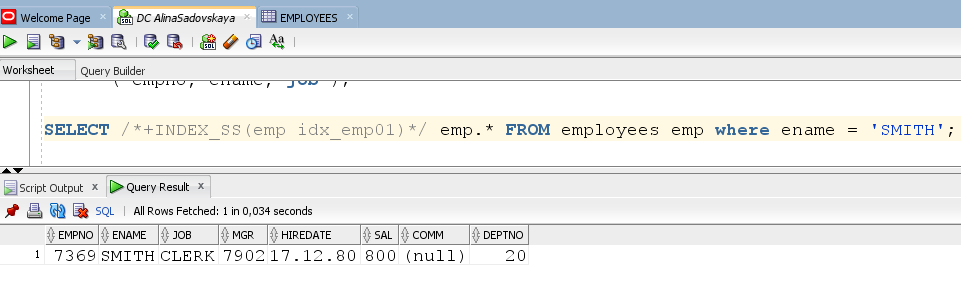
**Step 3:**  Get trace and statistic of explain plan

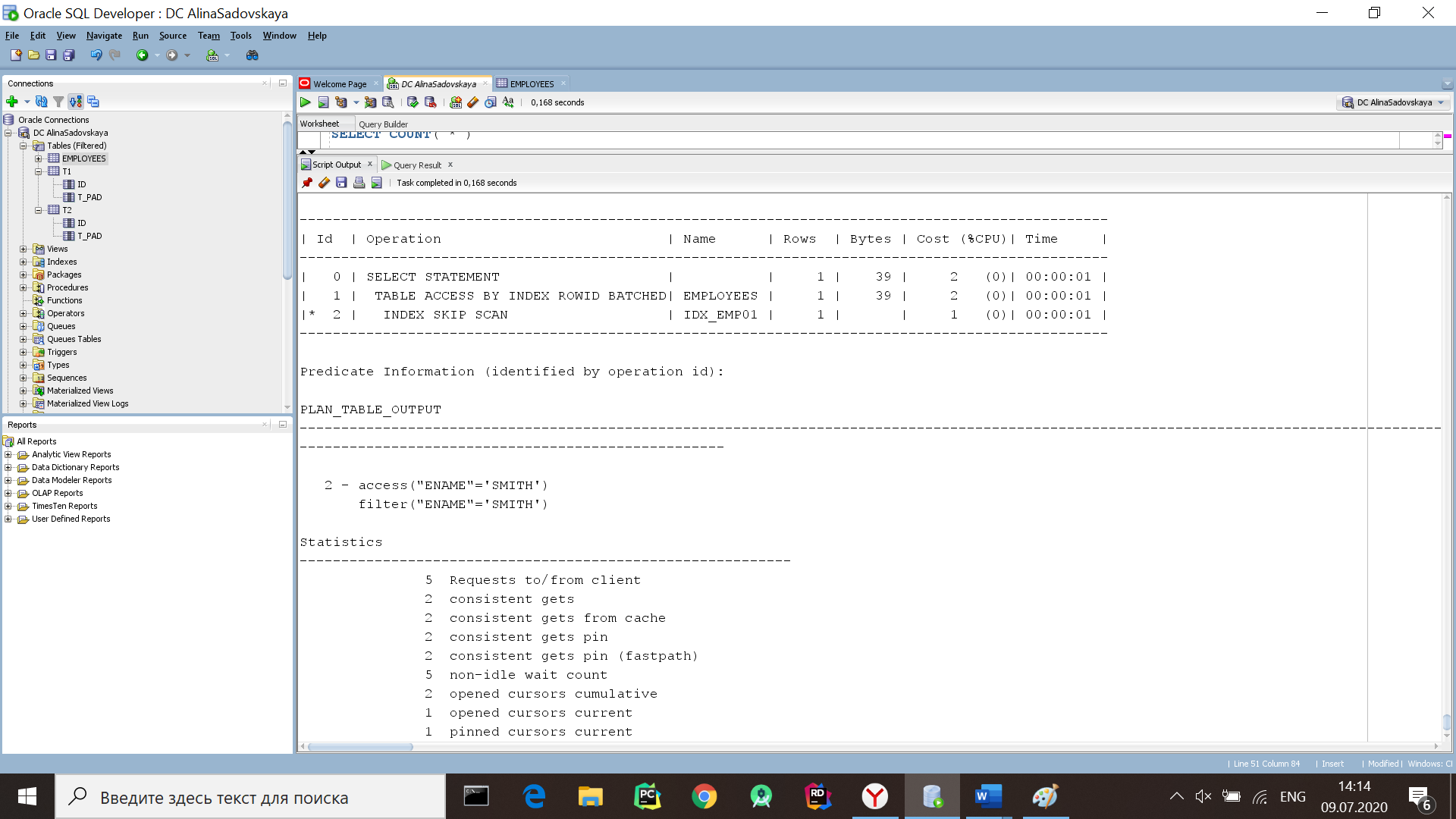
1) **SELECT COUNT( \* )**

**FROM EMPLOYEES emp where ename = 'SMITH';**

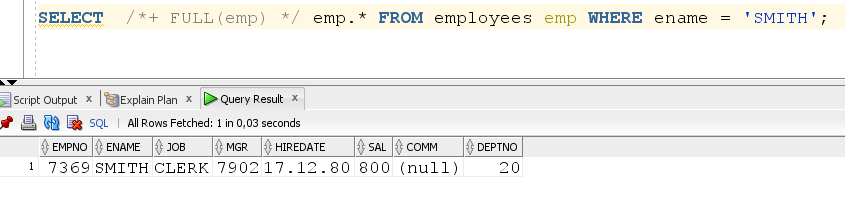


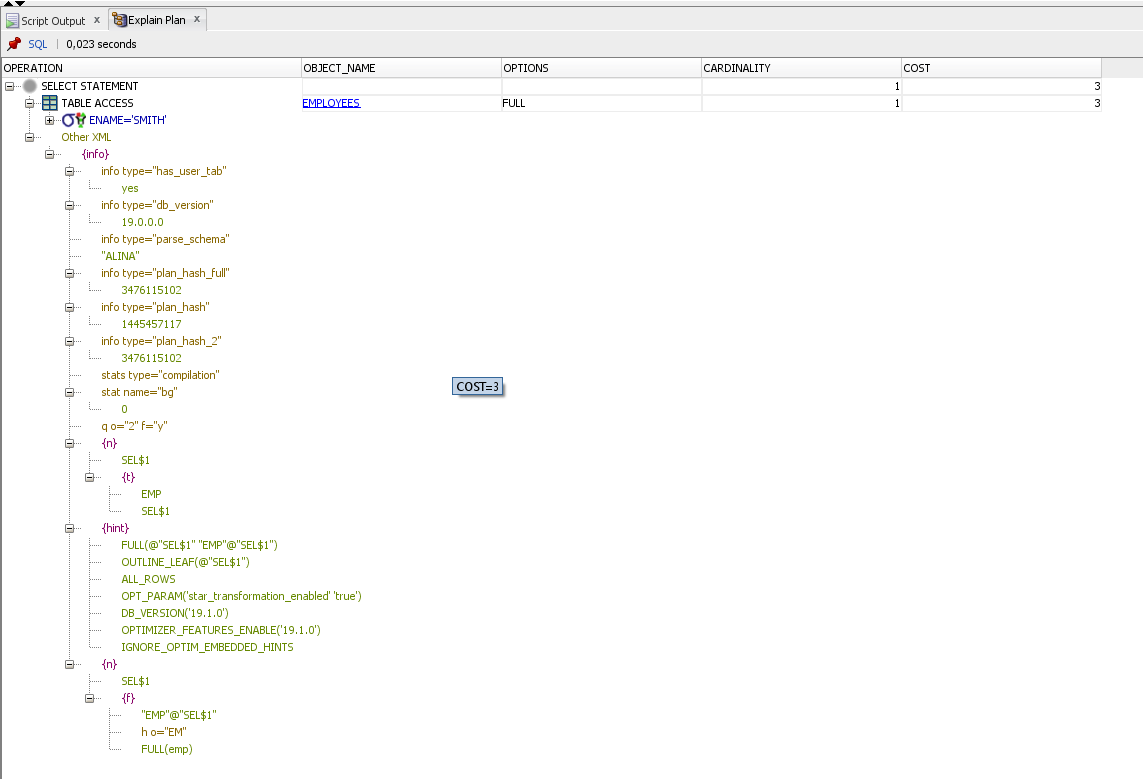
2) **SELECT /\*+INDEX\_SS(emp idx\_emp01)\*/ emp.\* FROM employees emp where ename = 'SCOTT';**

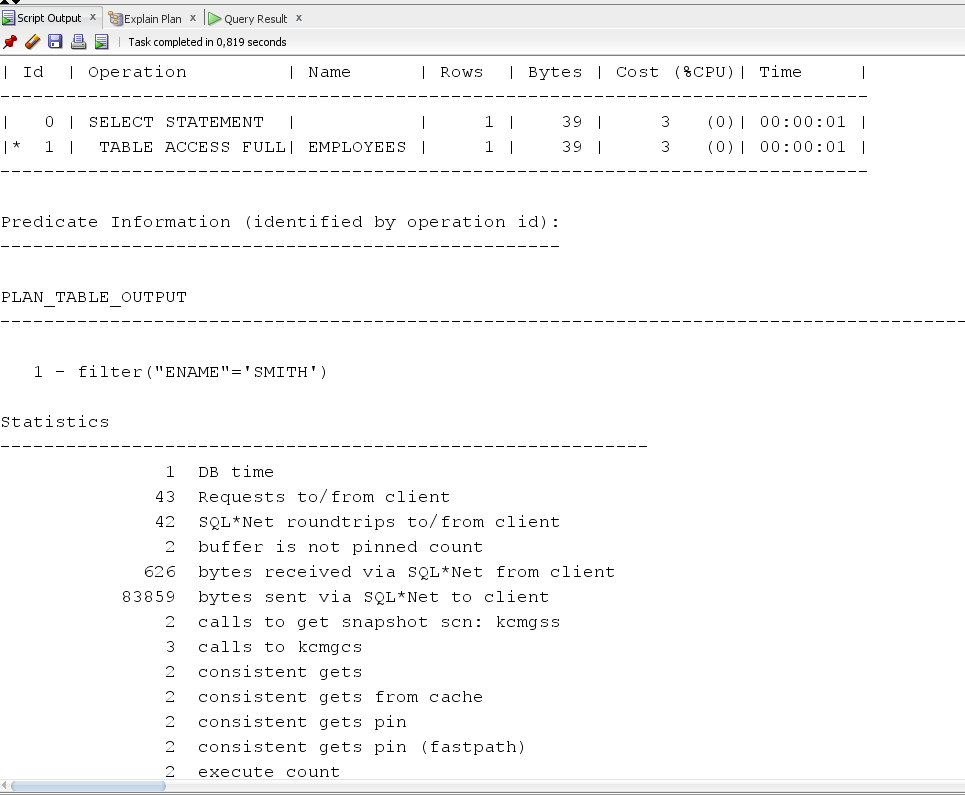




3) **SELECT /\* +FULL(emp) \*/ emp.\* FROM employees emp WHERE ename = 'SCOTT';**







|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Count of Blocks | Count of Used Blocks | Count of Rows | Consistent gets | Description |
| 3.1 | 20 | 1 | 1 | 1 | In this case, Oracle performed SELECT (without using hints) and used the most appropriate Index Scip Scan algorithm(see point 2.4) |
| 3.2 | 20 | 1 | 1 | 2 | Oracle performed SELECT (using the hint /\*+INDEX\_SS(emp idx\_emp01)\*/) and used the Index Scip Scan algorithm(see point 2.4) |
| 3.3 | 20 | 1 | 1 | 2 | Oracle performed SELECT (using the hint /\* +FULL(emp)\*/) and used the Full Scan algorithm(see point 1.1) |