Here are the results of labwork 05:

**Task 1. Auto Trace configuration training**

In SQL\*Plus it’s possible to automatically get a report on the execution path used by the SQL optimizer and the statement execution statistics. The report is generated after a successful SQL DML statement, such as SELECT, DELETE, UPDATE or INSERT. It is useful for monitoring and tuning the performance of these DML statements.

Summary table with all result and text description of analyses this results is presented below,

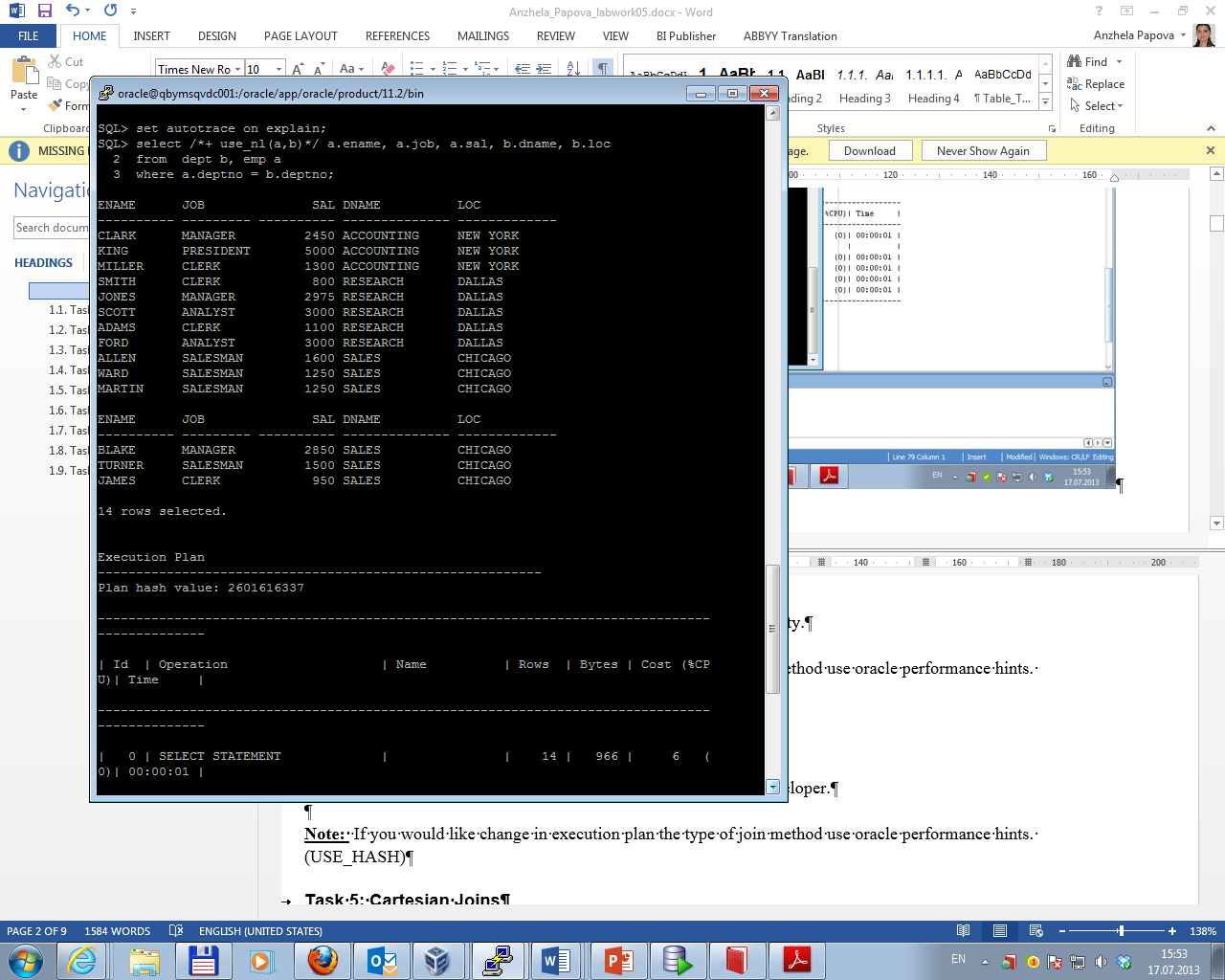
|  |  |  |  |
| --- | --- | --- | --- |
| № | Auto Trace Configuration Options | Expected Results | Description |
| 1 | set autotrace off | No report is generated | Default option |
| 2 | set autotrace on | The execution plan as well as statistics of the statement | Useful when it’s necessary to switch on the function |
| 3 | set autotrace traceonly | The execution plan and the statistics (as set autotrace on does), but doesn't print a query's result | Useful when it’s necessary to analyze a query that selects a large number of data |
| 4 | set autotrace on explain | The optimizer execution plan only | Useful when it’s necessary to switch on the function |
| 5 | set autotrace on statistics | The SQL statement execution statistics only |
| 6 | set autotrace on explain statistics | The optimizer execution plan and the SQL statement execution statistics (the same as p.2) |
| 7 | set autotrace traceonly explain | The execution plan without printing a query's result | Useful when it’s necessary to analyze a query that selects a large number of data |
| 8 | set autotrace traceonly statistics | The SQL statement execution statistics without printing a query's result |
| 9 | set autotrace traceonly explain statistics | The execution plan and the SQL statement execution statistics without printing a query's result (the same as p.3) |
| 10 | set autotrace off explain | No the execution plan is generated | Useful when it’s necessary to switch off the function |
| 11 | set autotrace off statistics | No the SQL statement execution statistics is generated |
| 12 | set autotrace off explain statistics | No report is generated (the same as p.1) |

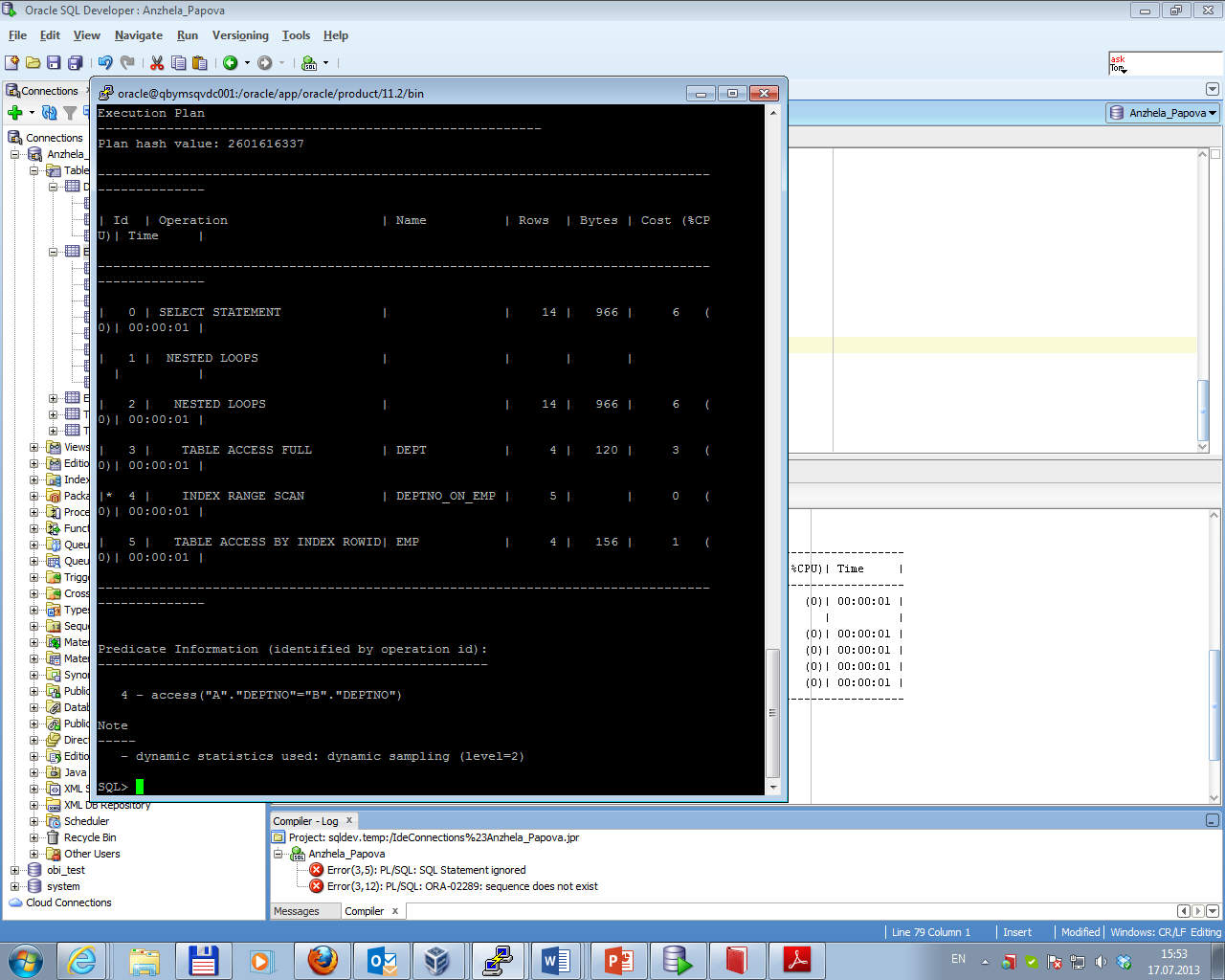
**Join Methods**

Task 2 Nested Loops Joins

Nested Loops Joins are typically most effective if the result set is limited in size and indexes are present on the columns used for the join. With nested loops, the cost of the operation is based on reading each row of the outer row source and joining it with the matching row of the inner row source.

The result of Nested Loops Join is presented below.

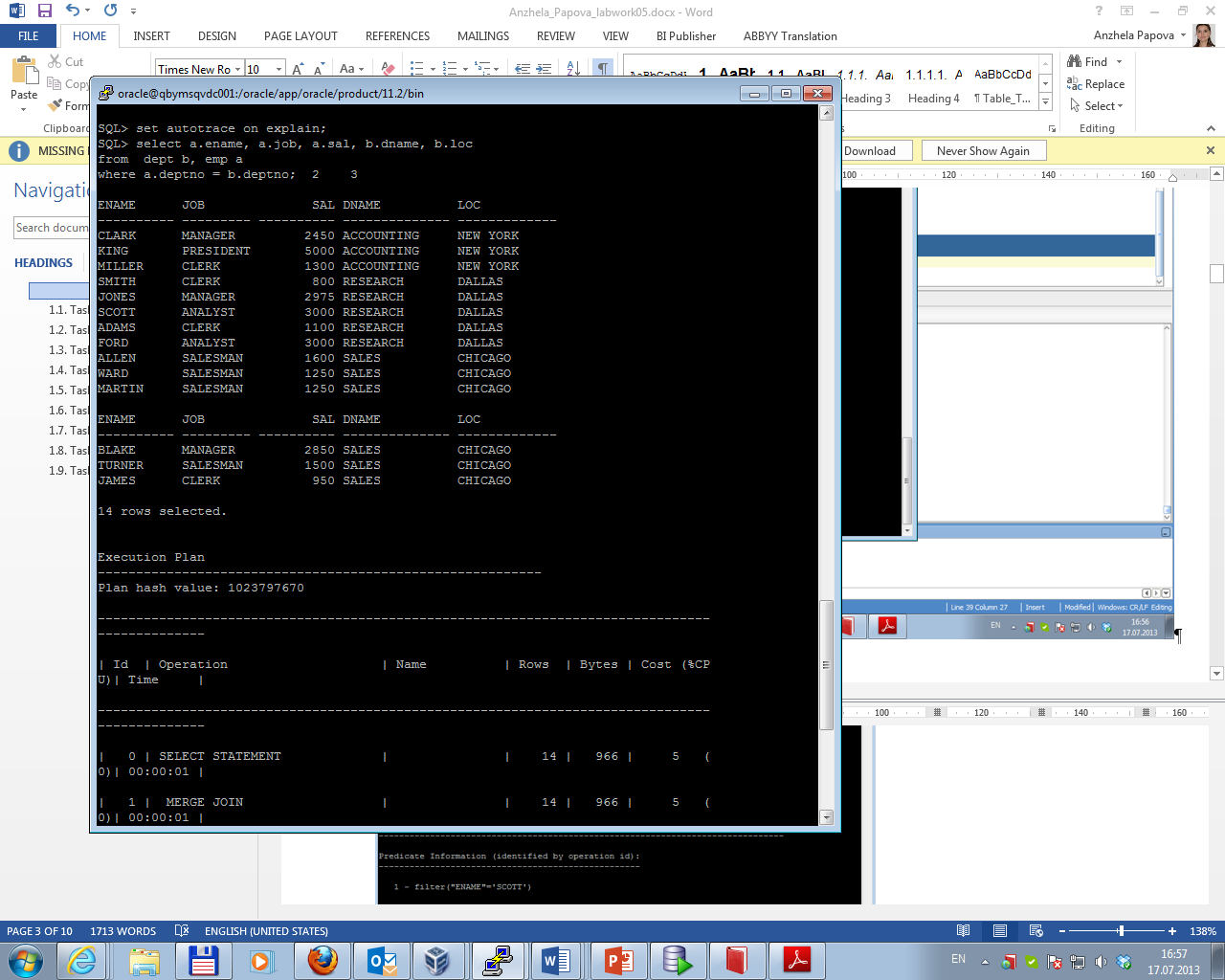


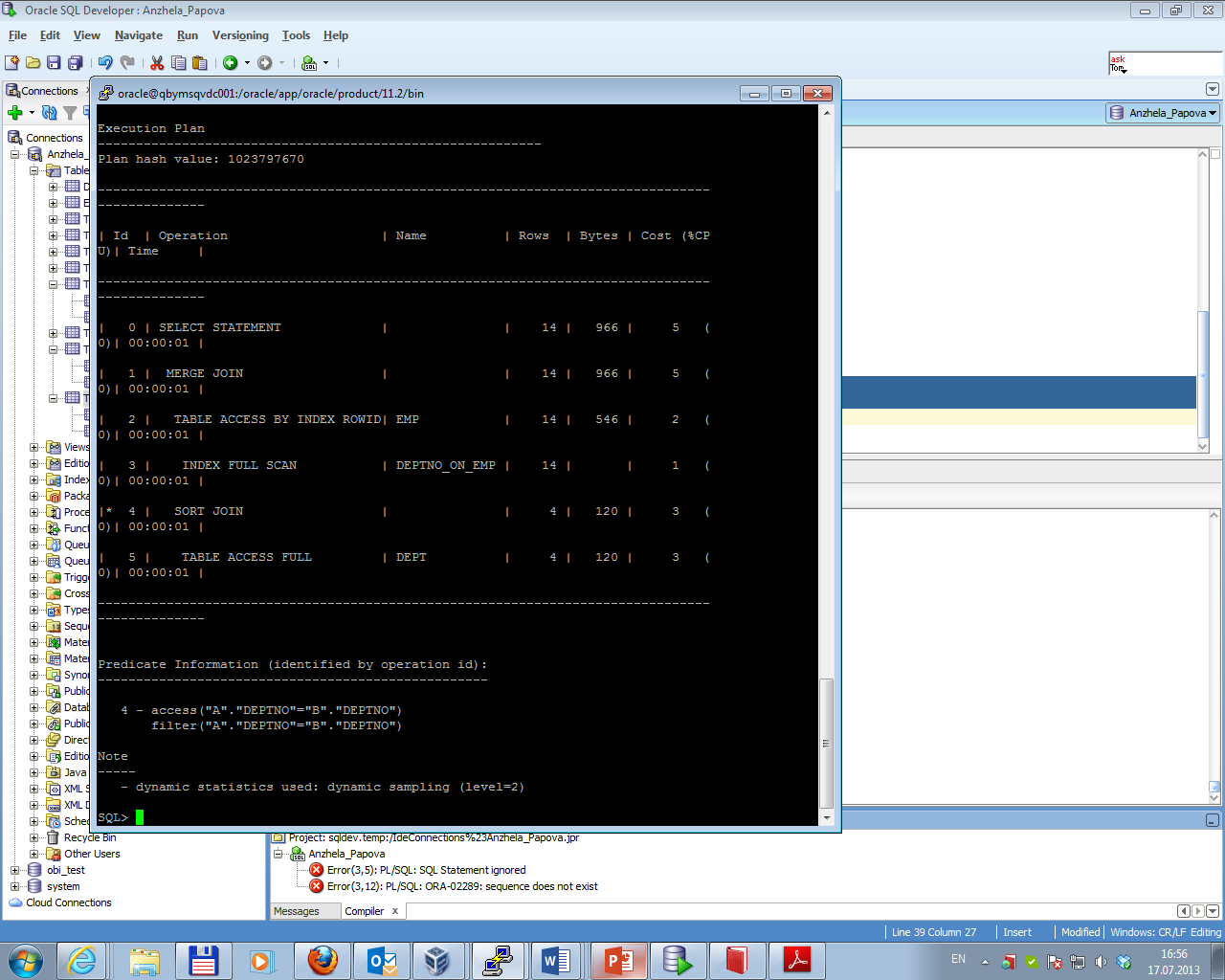


Task 3 Sort-Merge Joins

Sort-merge joins read the two tables to be joined independently, sorts the rows from each table (but only those rows that meet the conditions for the table in the WHERE clause) in order by the join key, and then merges the sorted rowsets. The sort operations are the most expensive part for this join method.

The result of Sort-Merge Join is presented below.



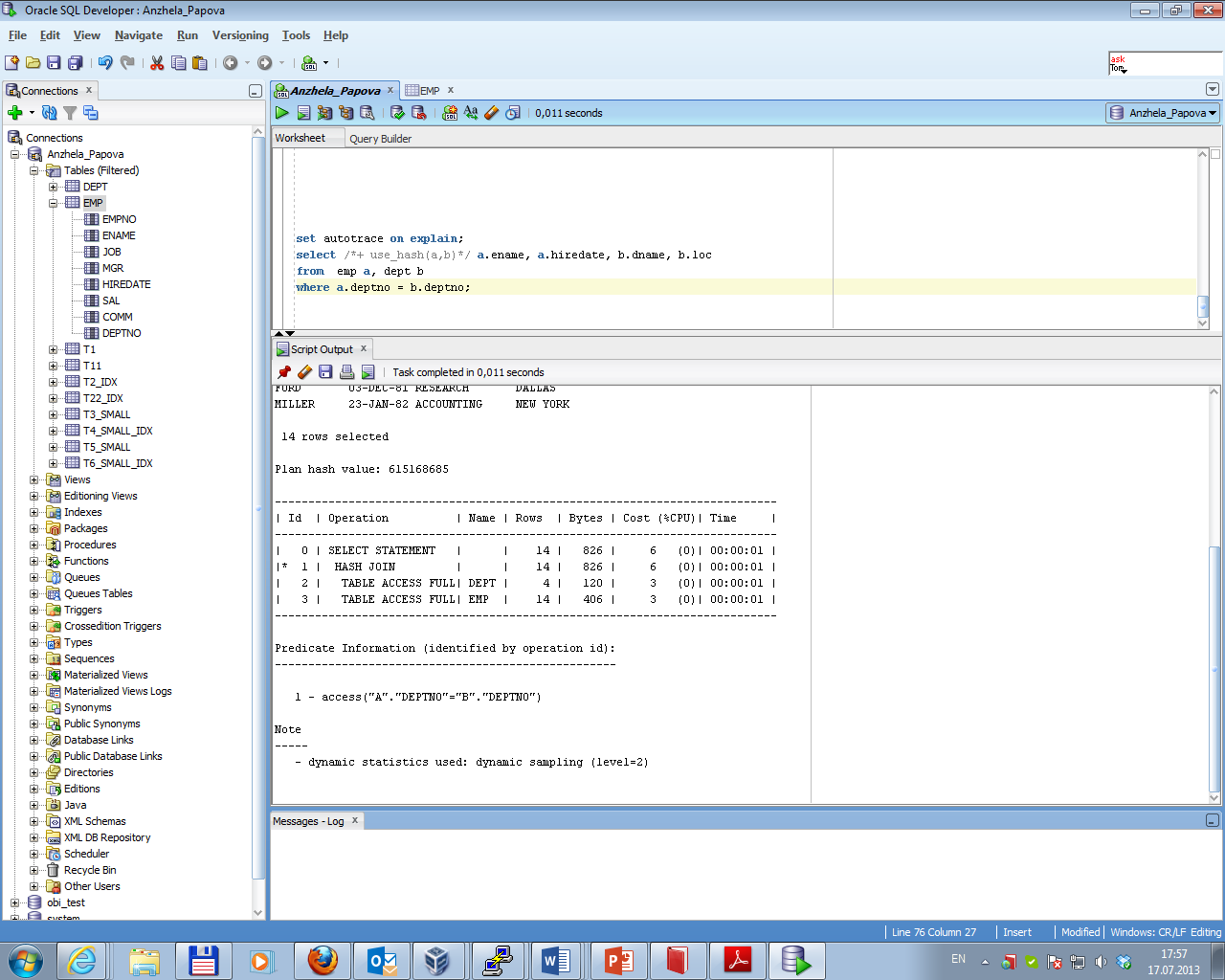


Task 4 Hash Joins

Hash joins, like sort-merge joins, first reads the two tables to be joined independently and applies the criteria in the WHERE clause. Based on table and index statistics, the table that is determined to return the fewest rows will be hashed in its entirety into memory. This hash table includes all the row data for that table and is loaded into hash buckets based on a randomizing function that converts the join key to a hash value.

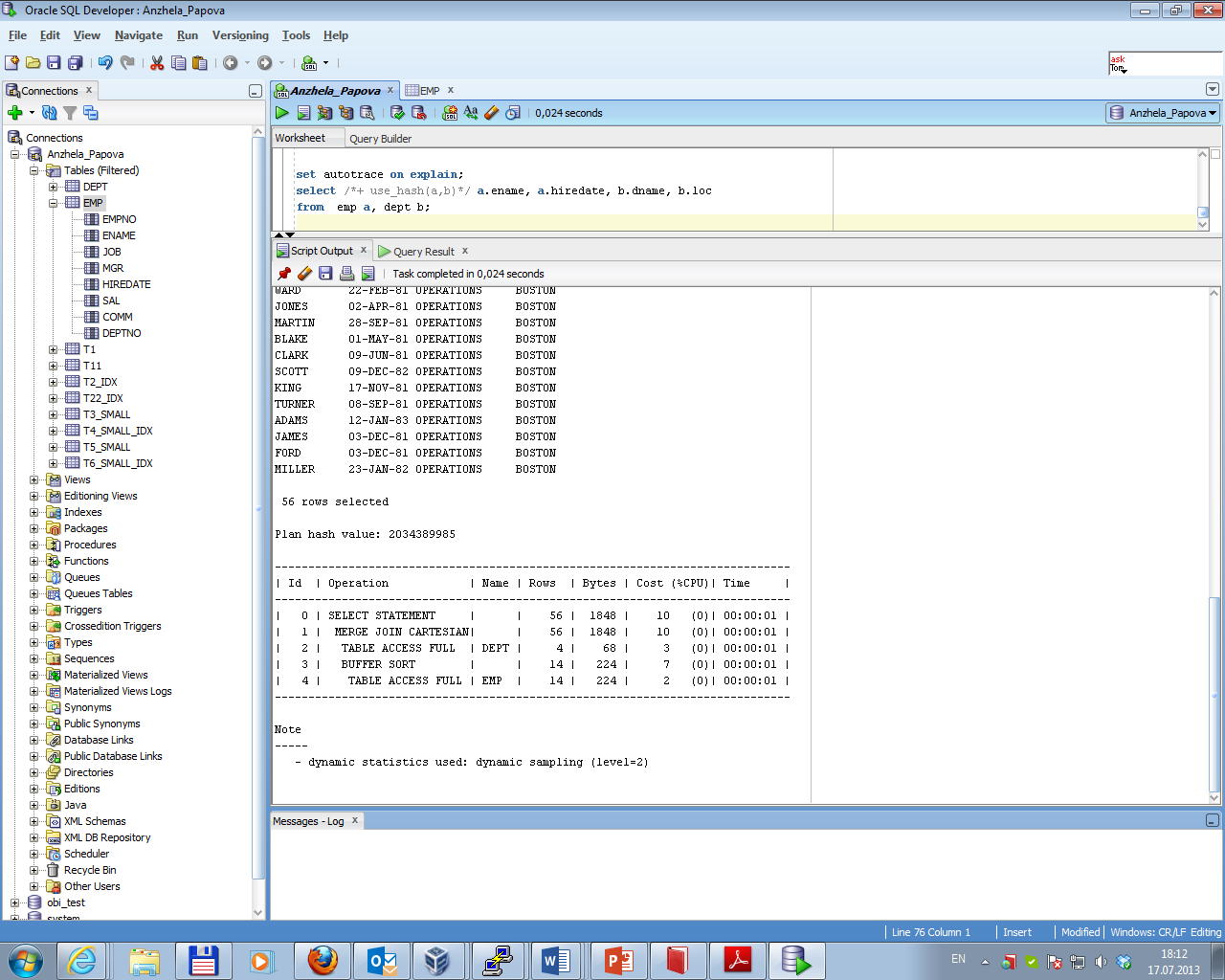
The next step is for the other larger table to be read and the hash function is applied to the join key column. That hash value is then used to probe the smaller in memory hash table for the matching hash bucket where the row data for the first table resides.

The result of Hash Join is presented below.



Task 5 Cartesian Joins

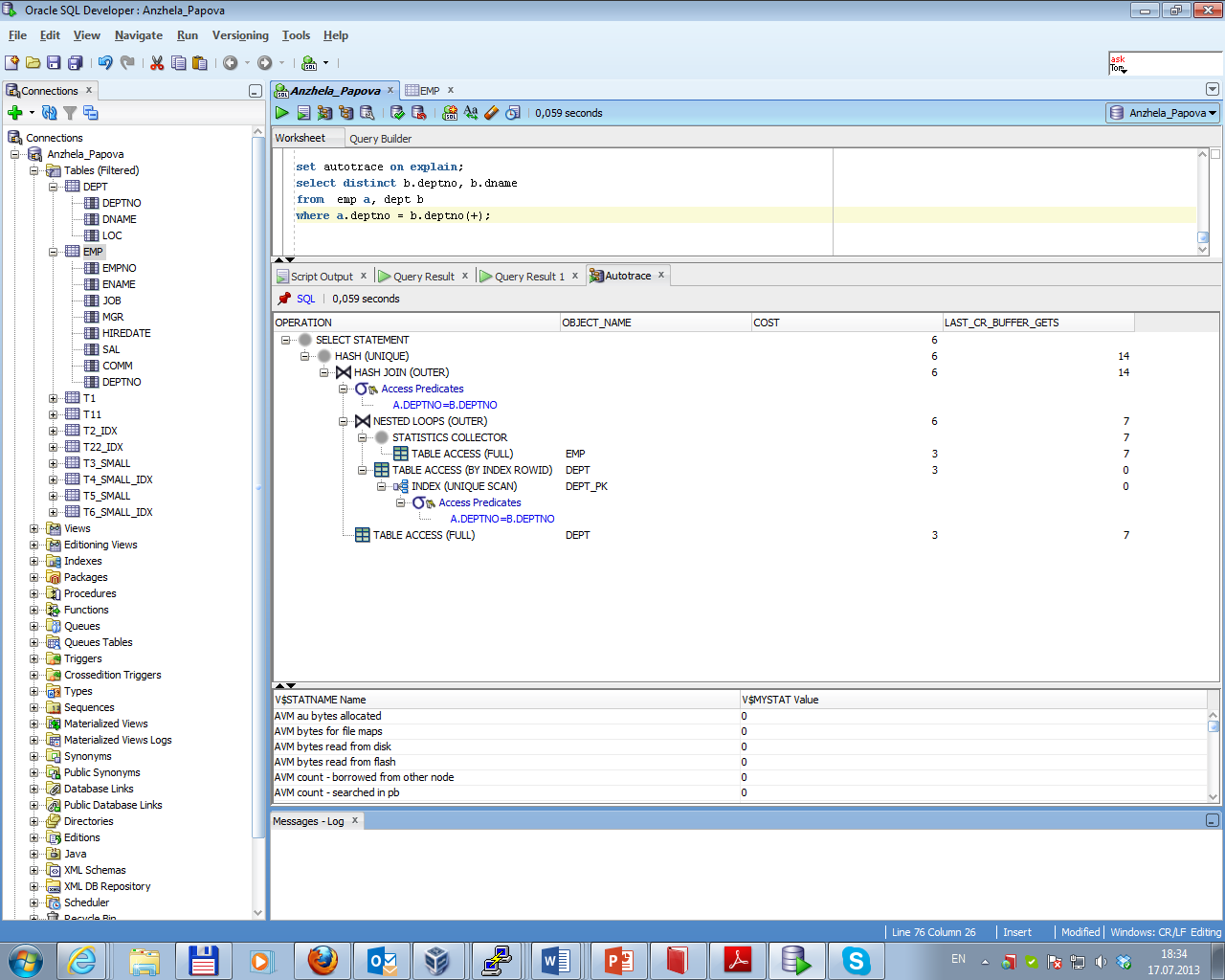
Cartesian joins occur when all the rows from one table are joined to all the rows of another table. Therefore, the total number of rows resulting from the join equals the number of rows from one table multiplied by the number of rows in the other table.

The result of Cartesian Join is presented below.

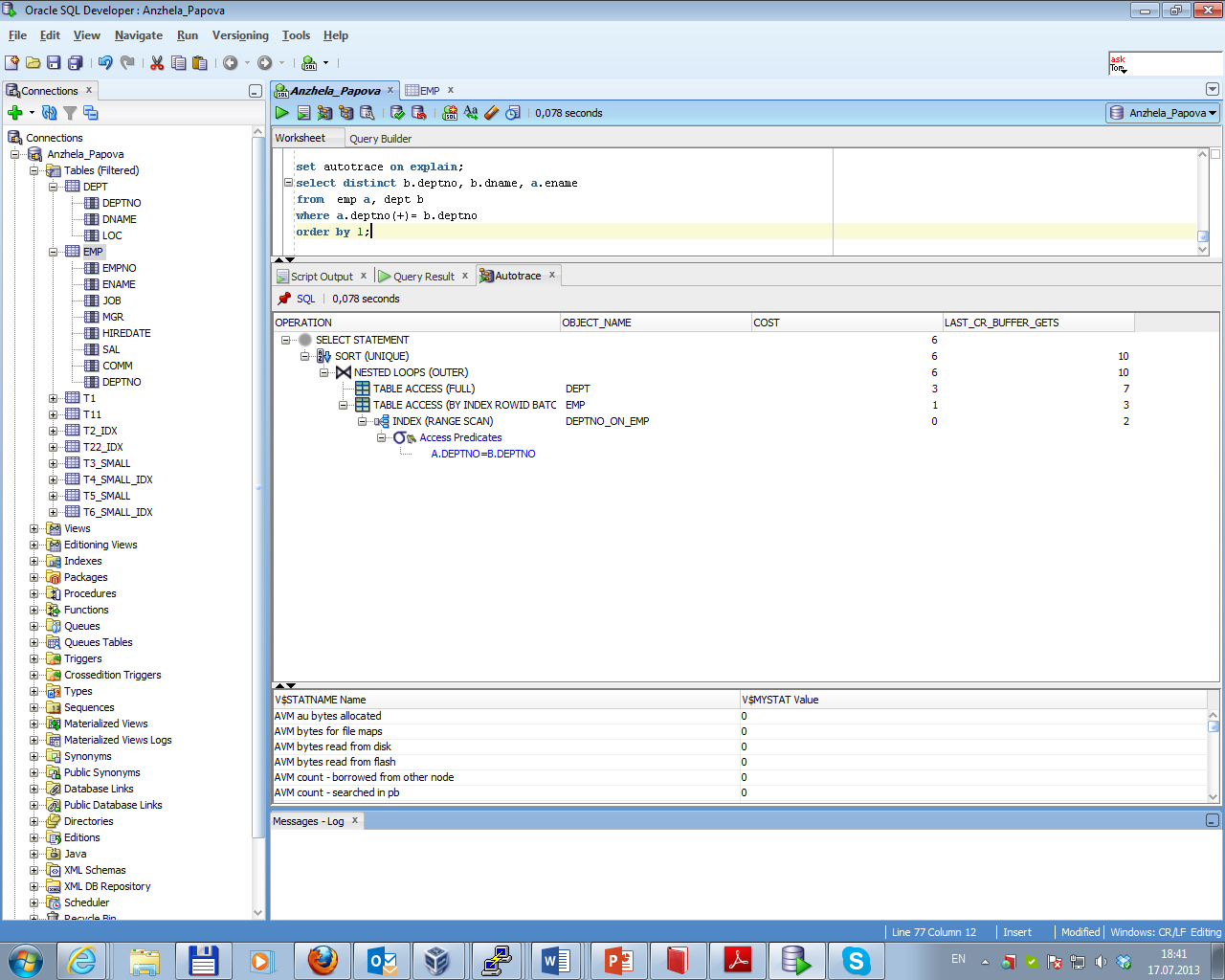
Task 6 Left/Right Outer Joins

An outer join returns all rows from one table and only those rows from the joined table where the join condition is met.

Autotrace for Left Outer Join is presented below.

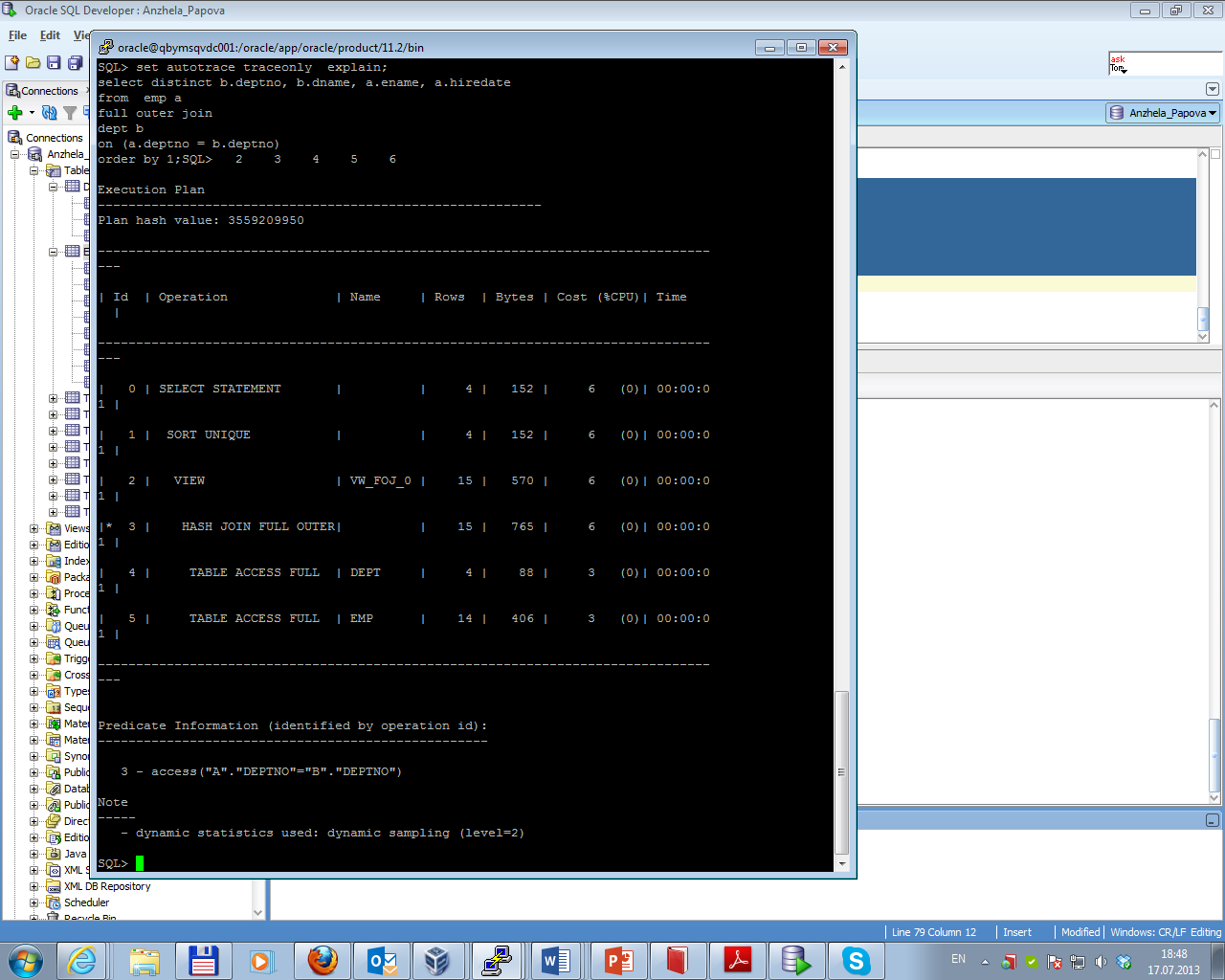


Autotrace for Right Outer Join is presented below.



Task 7 Full Outer Join

Explain plan for Right Outer Join is presented below

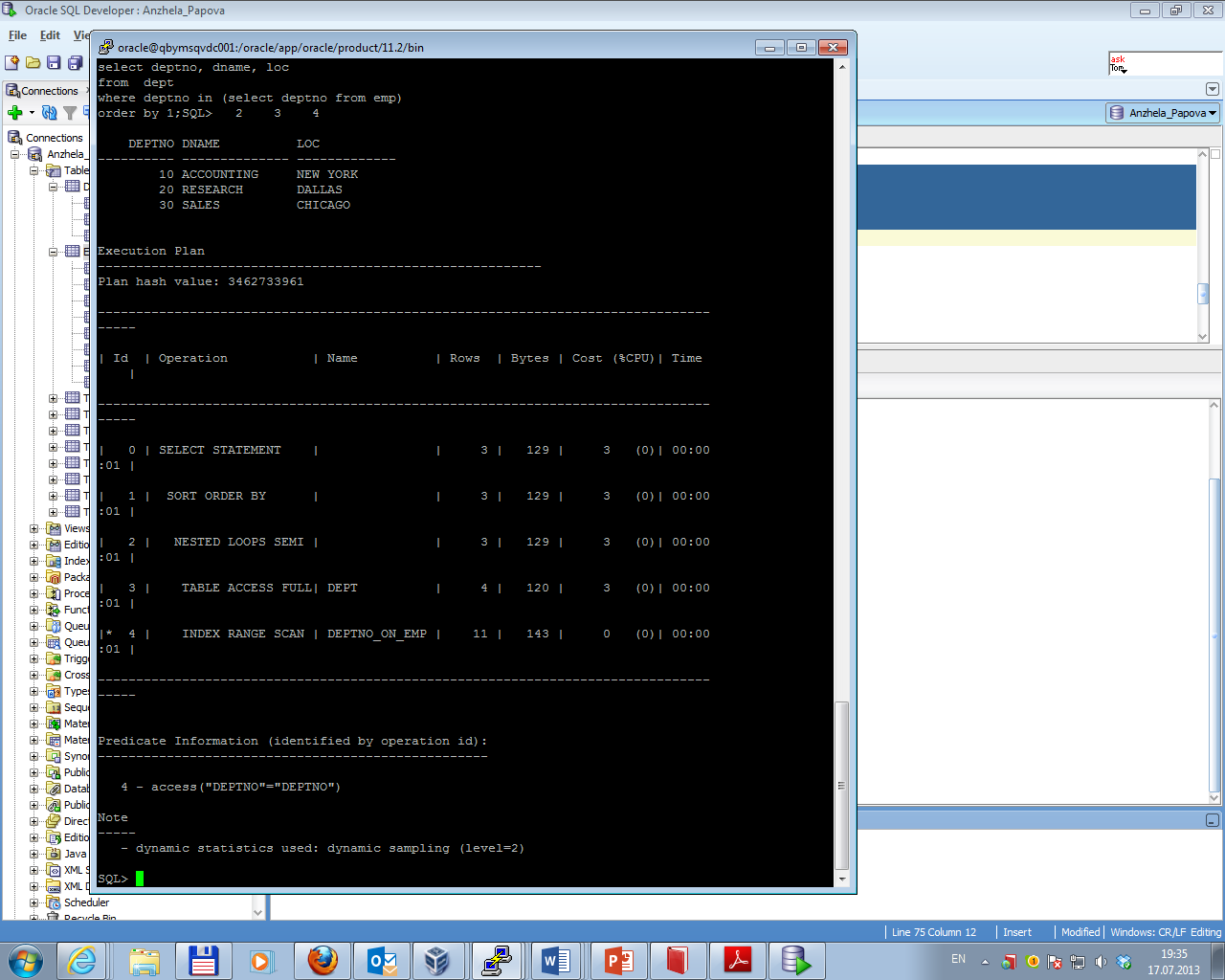


Task 8 Semi Joins

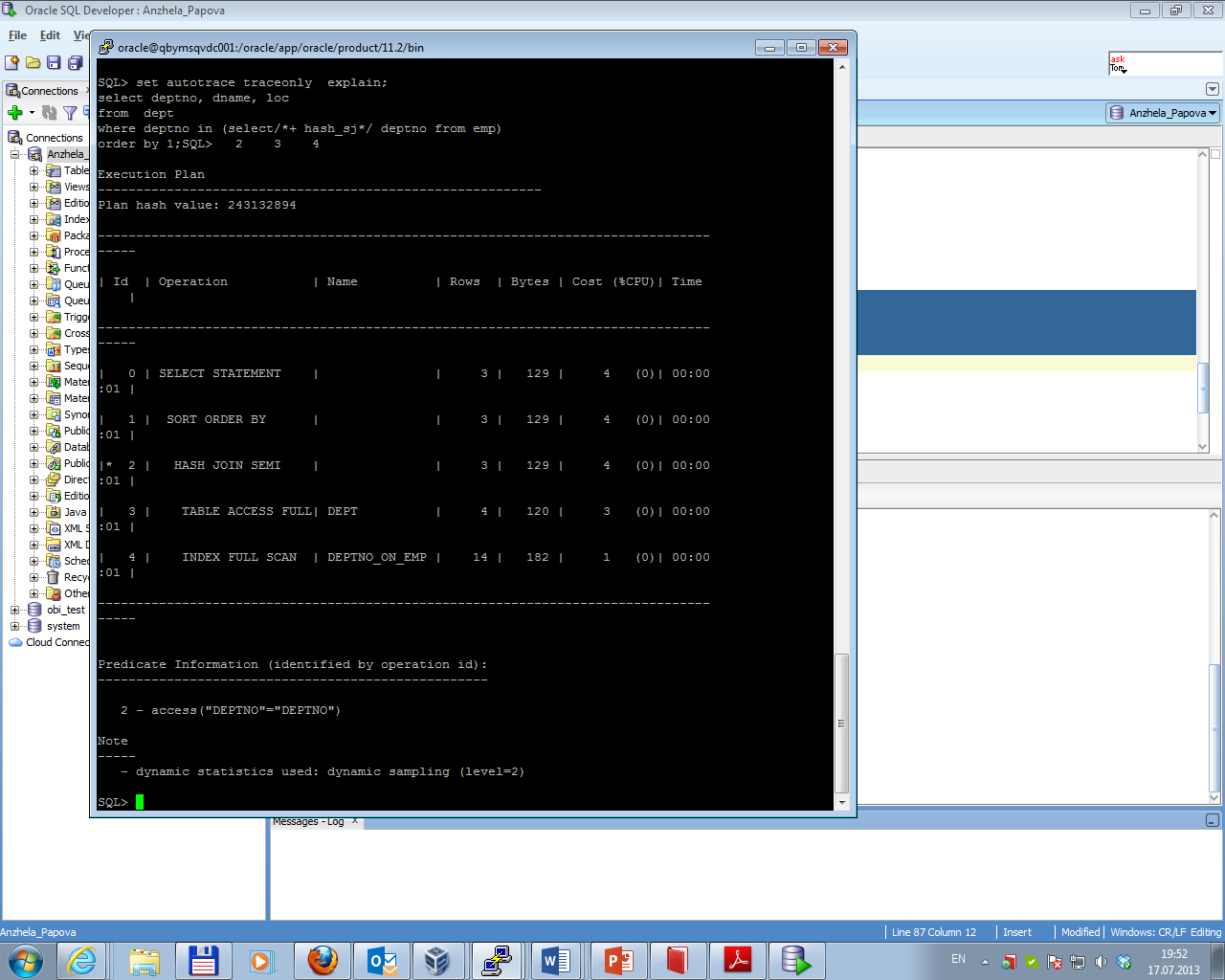
A semi-join is a join between two sets of data (tables) where rows from the first set are returned, based on the presence of at least one matching row in the other set. The main difference between a normal inner join and a semi-join is that with a semi-join, each record in the first set is returned only once, regardless of how many matches there are in the second set.

The three most common join methods in Oracle are nested loops, hash joins, and merge joins. The results of using each of them and option no\_semijoin are presented below.

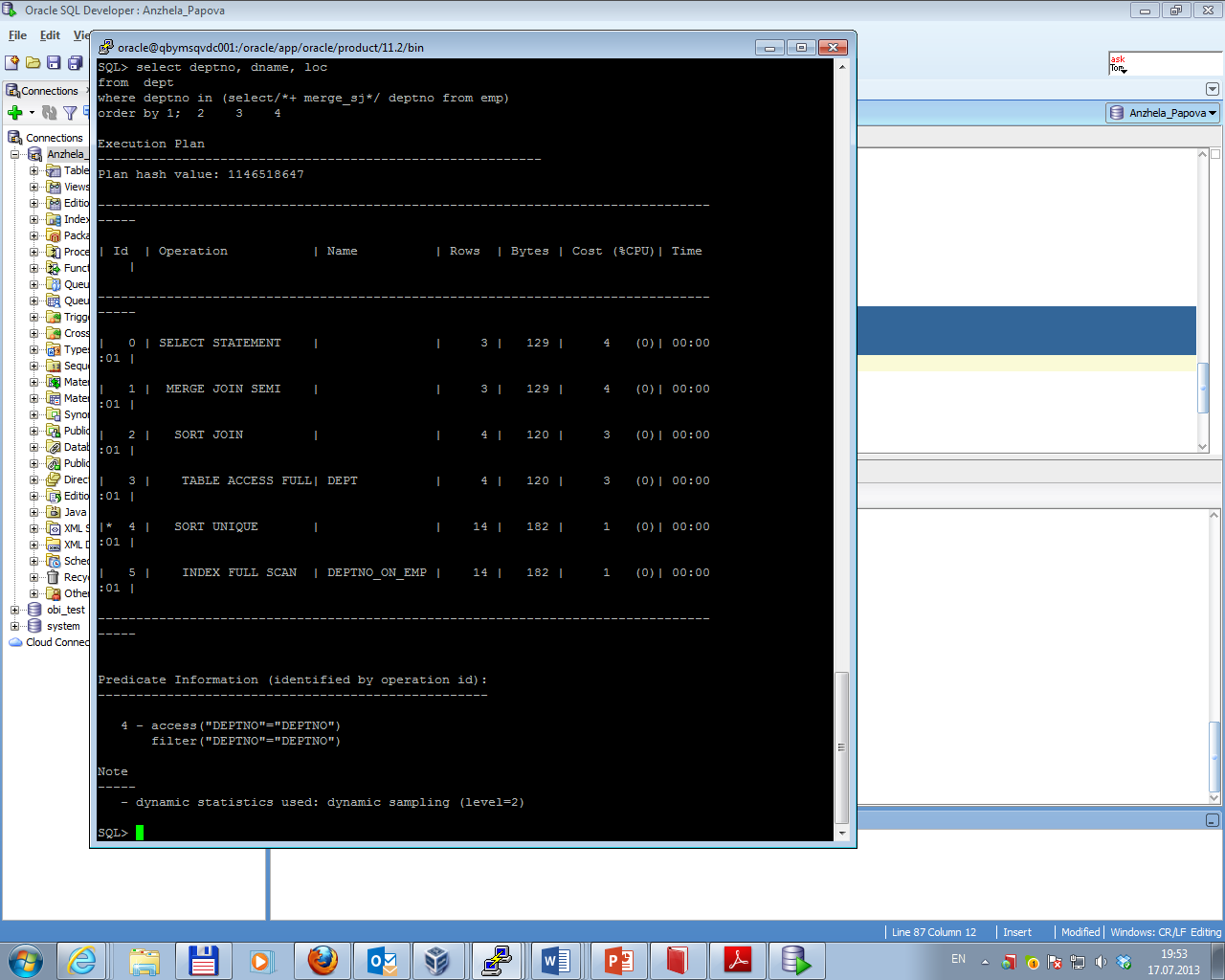
*Nested loops semi-join:*



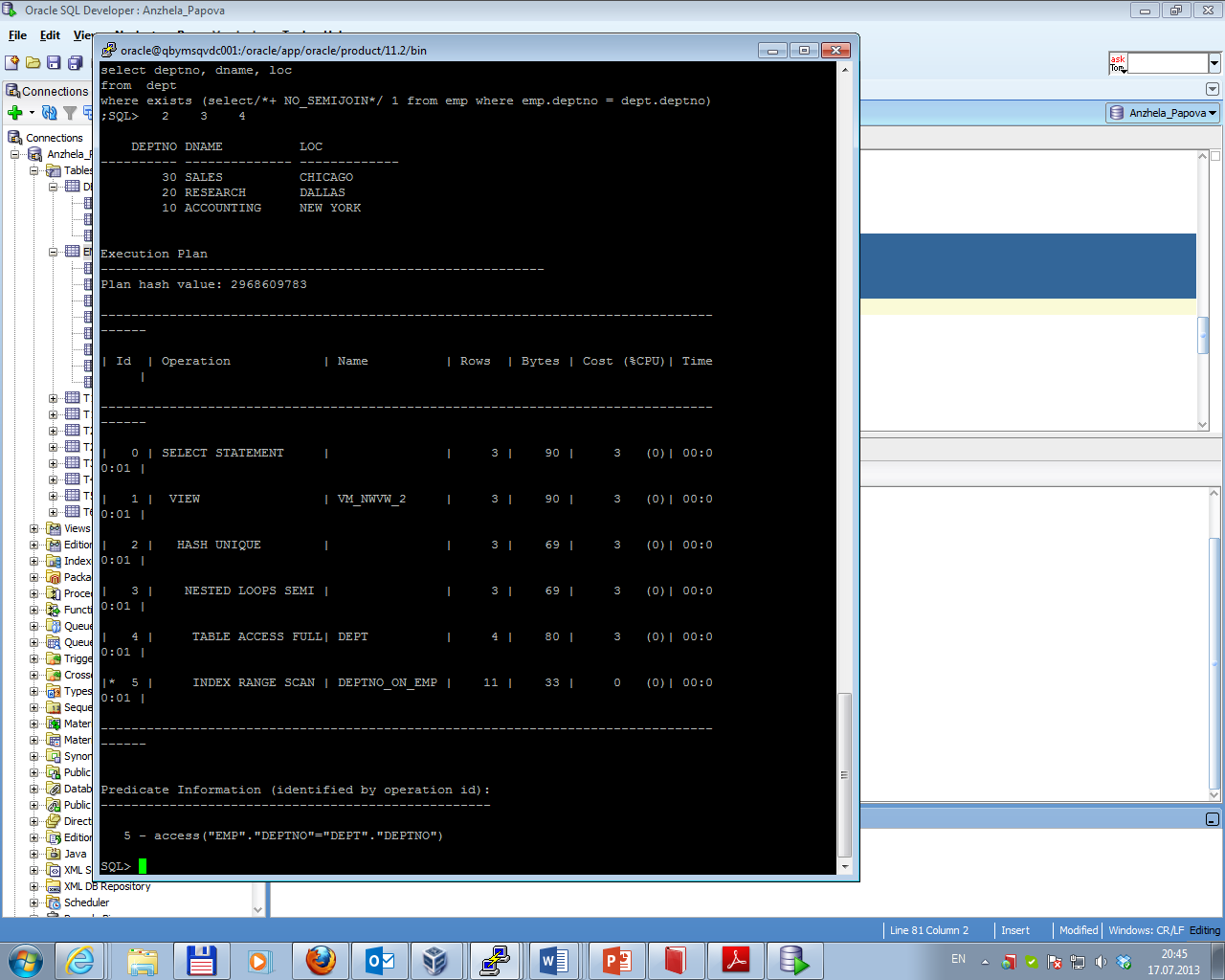
*Hash semi-join:*



*Merge semi-join:*



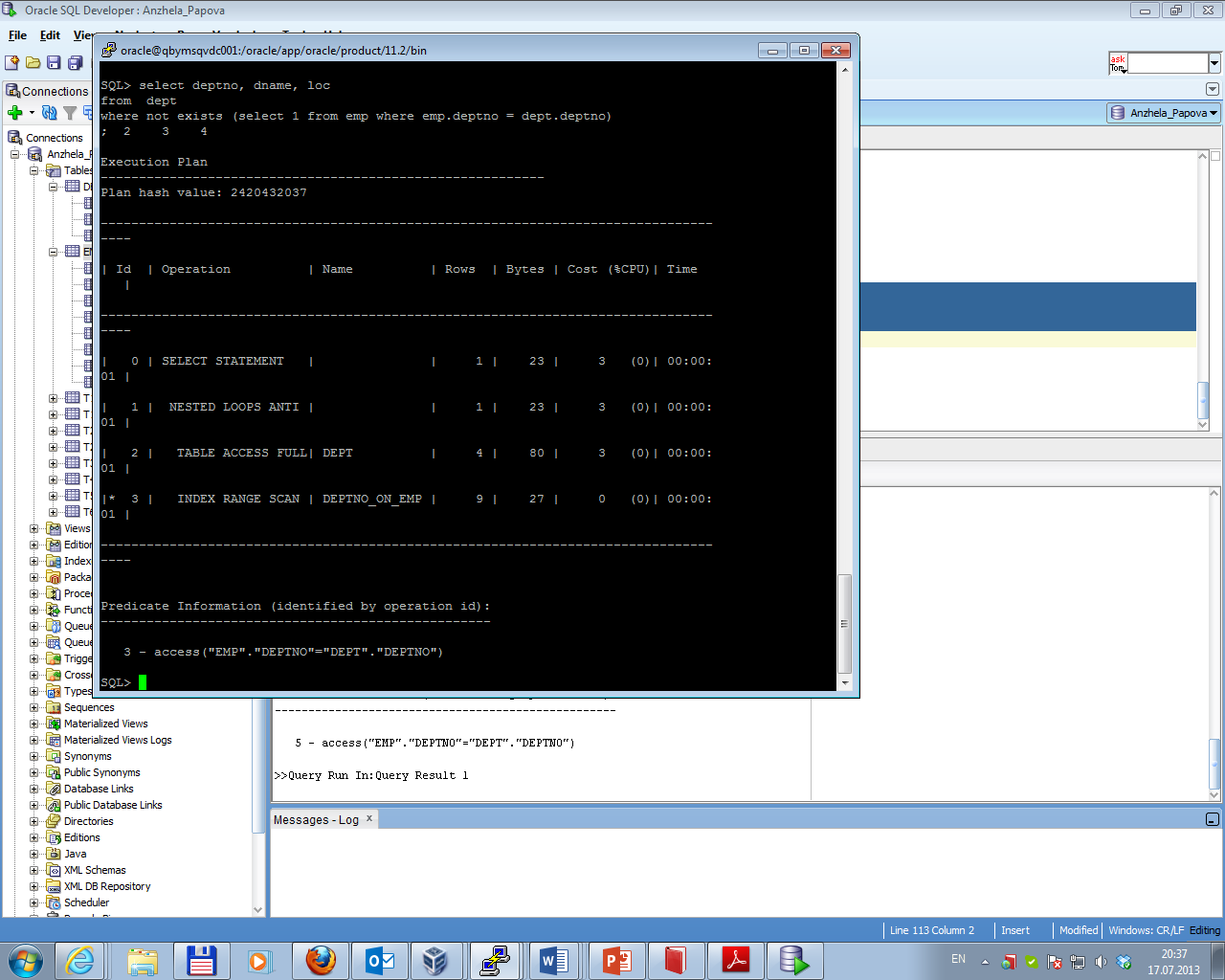
*No semi-join:*



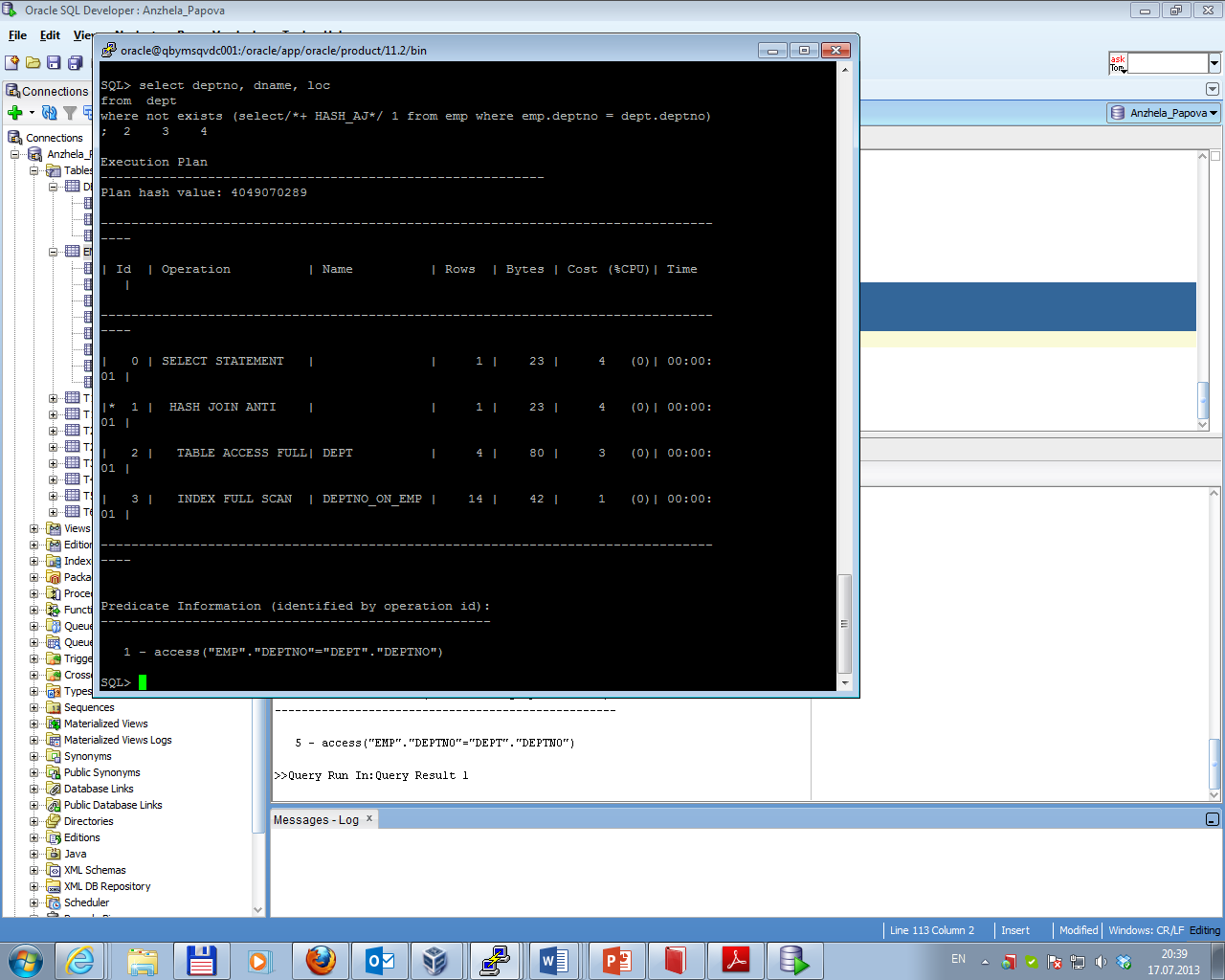
Task 9 Anti Joins

Anti-joins are basically the same as semi-joins in that they are an optimization option that can be applied to nested loop, hash, and merge joins. The results of using each of them are presented below.

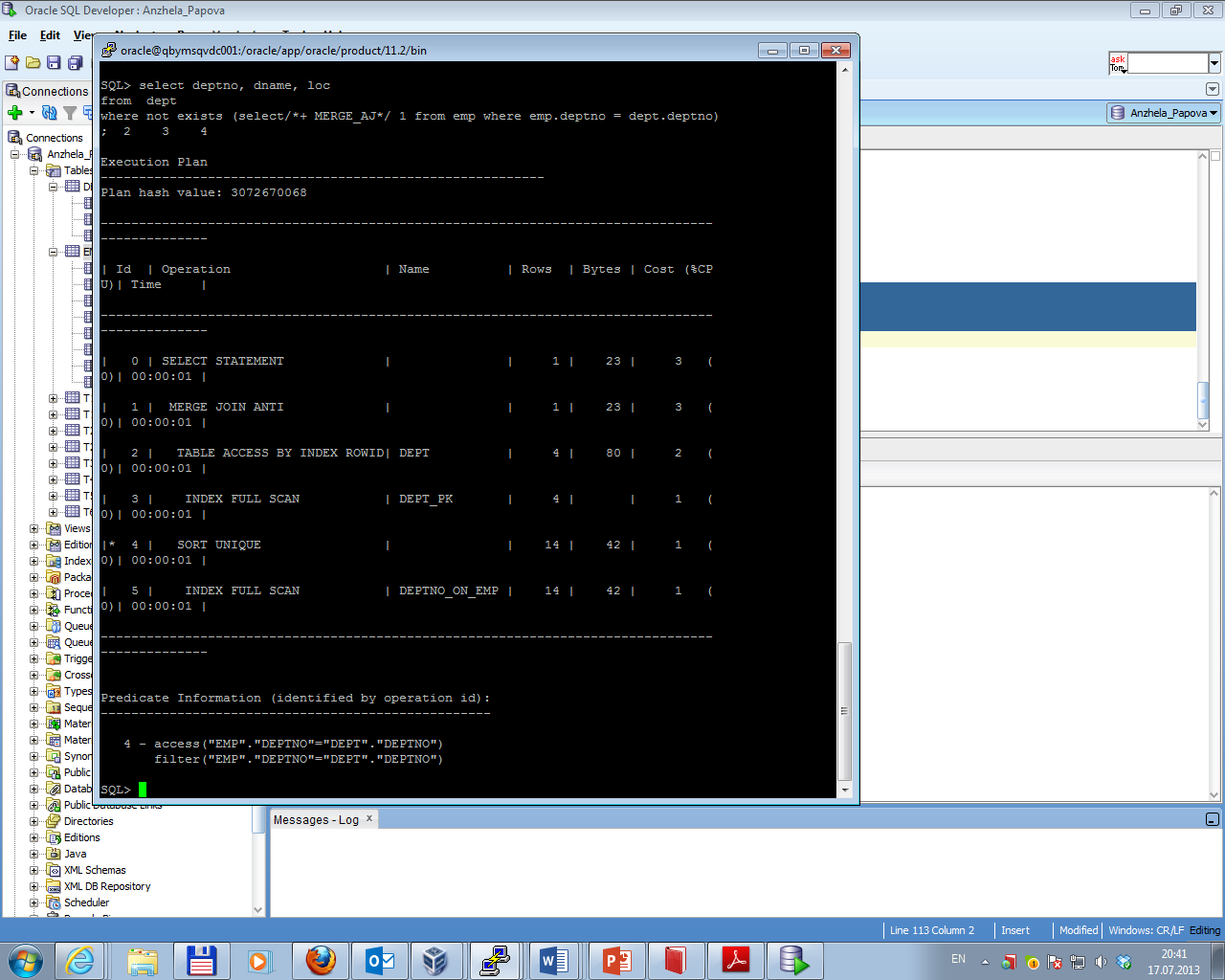
*Nested loops anti-join:*



*Hash anti-join:*



*Merge anti-join:*



Task 10 Summary table

Comparison of possible variant of join methods and join access methods is presented in the table below.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Join Access “A” | Join Access “B” | Nested Loop | | Hash Join | | Sort-Merge Join | | Anti-Join | | Semi-Join | |
| cost\* | mark | cost\* | mark | cost\* | mark | cost\* | mark | cost\* | mark |
| Small Table | Small Table | 58 | ++ | 6 | +++ | 6 | +++ | 6 | +++ | 6 | +++ |
| Small Table | Indexed Small Table | 102 | ++ | 6 | +++ | 5 | +++ | 3 | +++ | 5 | +++ |
| Indexed Small Table | Indexed Small Table | 3 | +++ | 4 | +++ | 3 | +++ | 3 | +++ | 3 | +++ |
| Small Table | Large Table | 87624 | - | 891 | ++ | 6582 | - | 891 | + | 891 | + |
| Small Table | Indexed Large Table | 201 | ++ | 929 | + | 5427 | - | 102 | ++ | 52 | ++ |
| Indexed Small Table | Indexed Large Table | 101 | ++ | 927 | + | 5425 | - | 102 | ++ | 52 | ++ |
| Large Table | Large Table | 884M | - | 4241 | +/- | 10609 | - | 3956 | - | 3956 | - |
| Large Table | Indexed Large Table | 2001K | - | 4041 | +/- | 9169 | - | 900 | + | 3689 | - |
| Indexed Large Table | Indexed Large Table | 10014 | - | 934 | + | 5431 | - | 586 | + | 586 | + |

*\*Cost values were defined using sample tables*