**Report**

**on the**

**U1M5.LW.Access and**

**Join Methods Part 2**

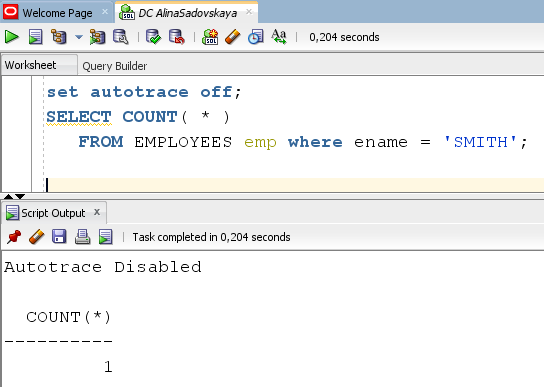
**Alina Sadovskaya**

# 1. Auto Trace & Explain Plan

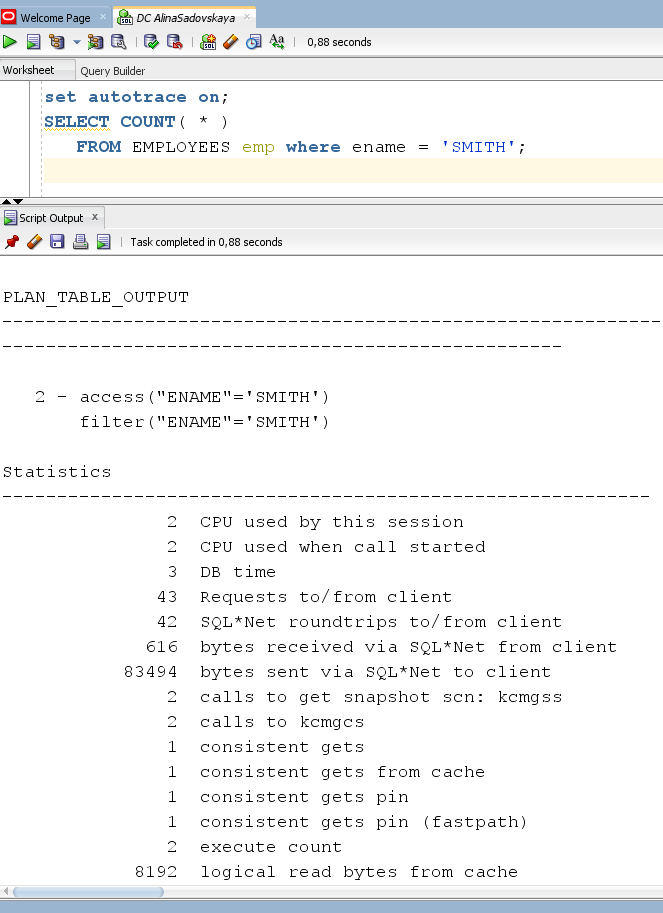
## 1.1. Task 1: Auto Trace configuration training

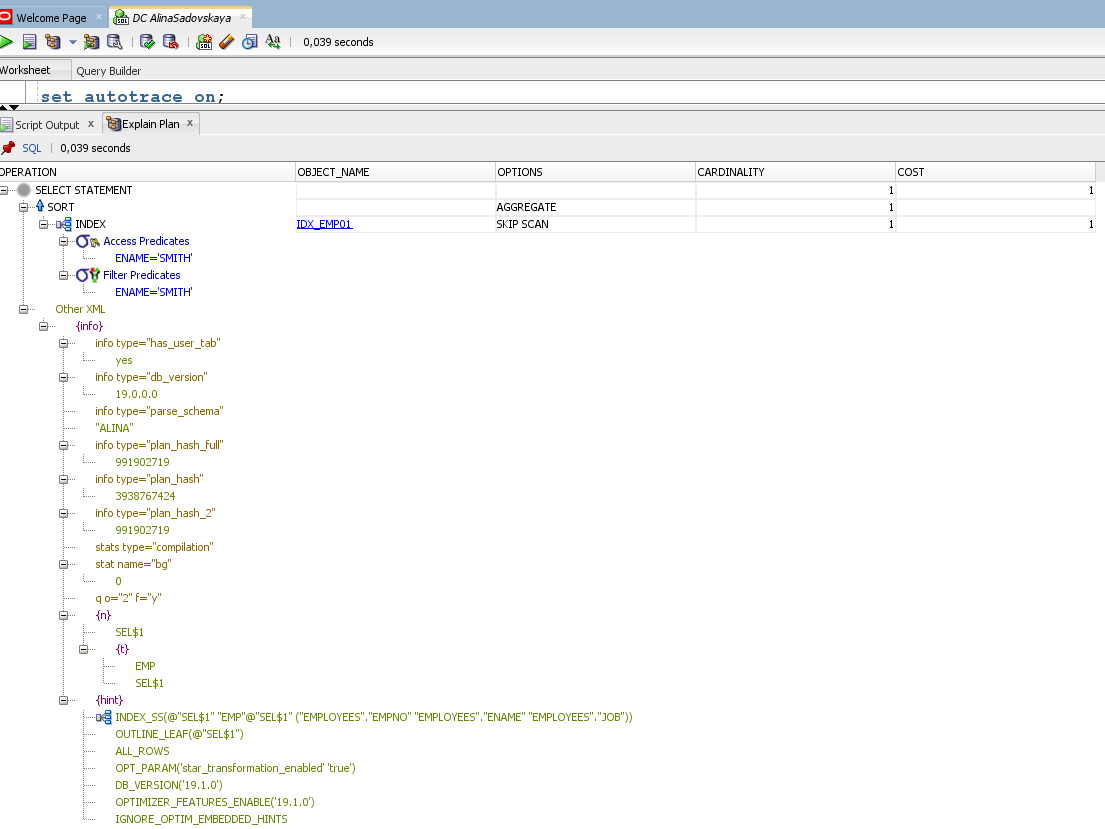
AUTOTRACE is an SQL\*Plus tool that displays the execution plan for running queries and information about the resources they used.

1. set autotrace off

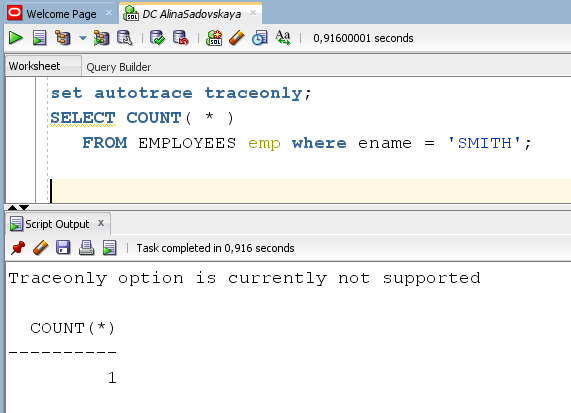


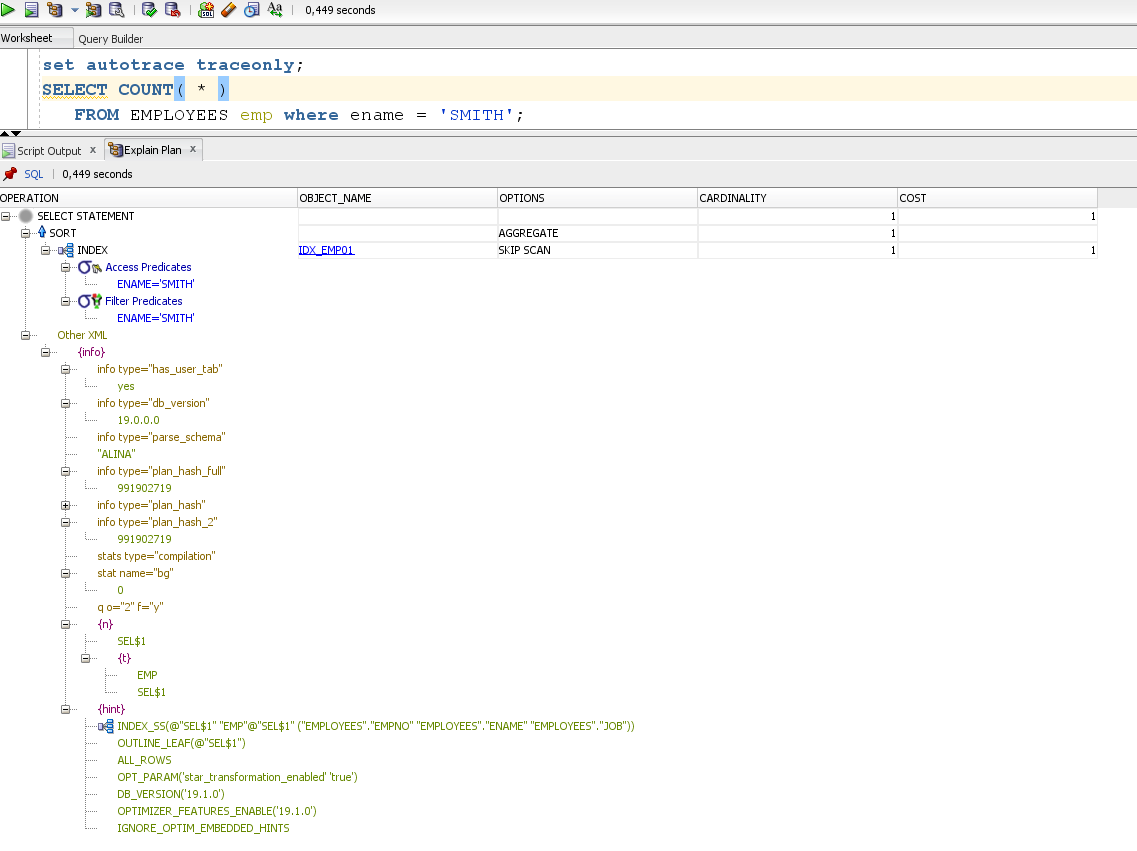
1. set autotrace on



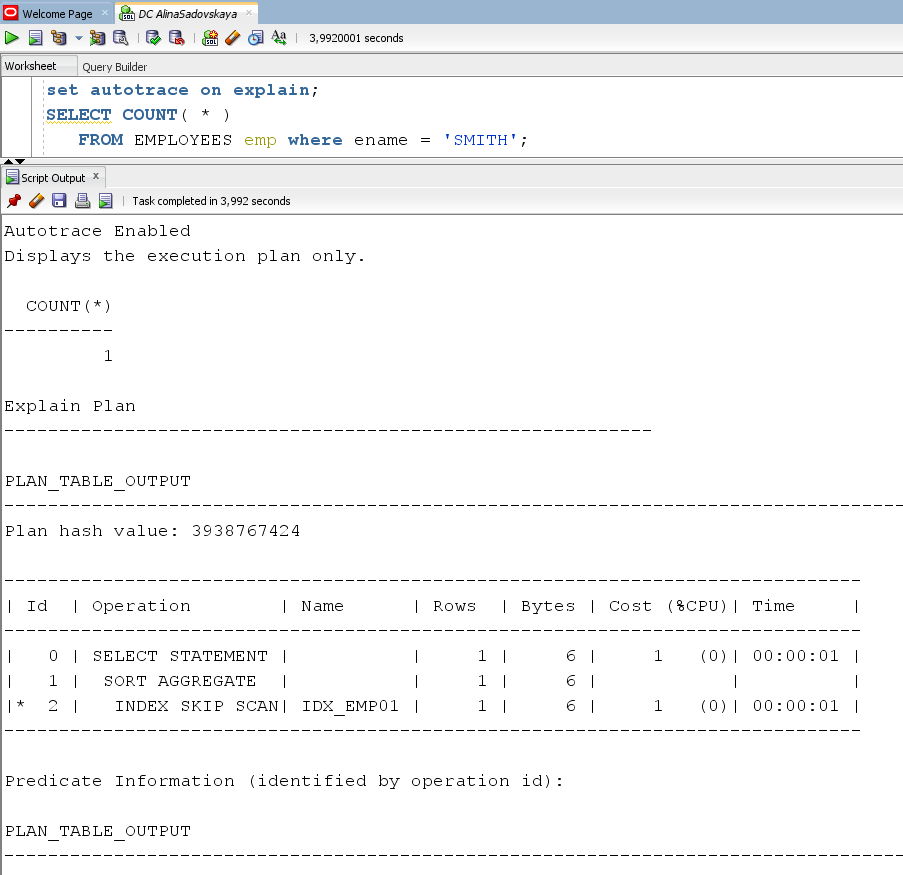


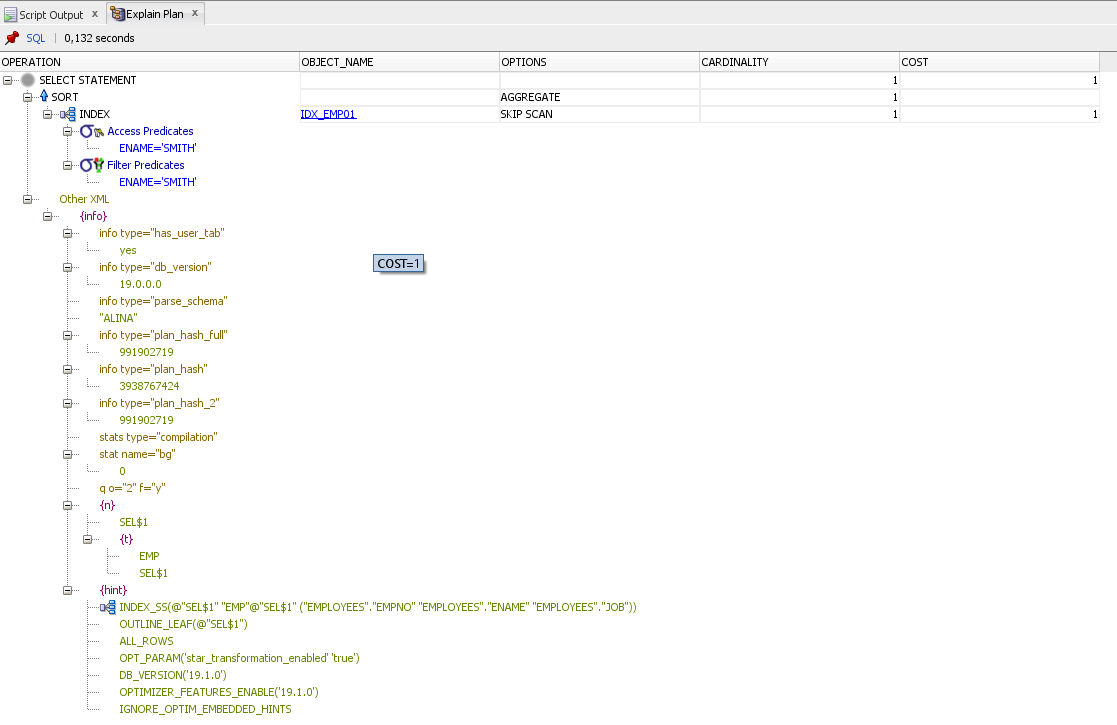
1. set autotrace traceonly



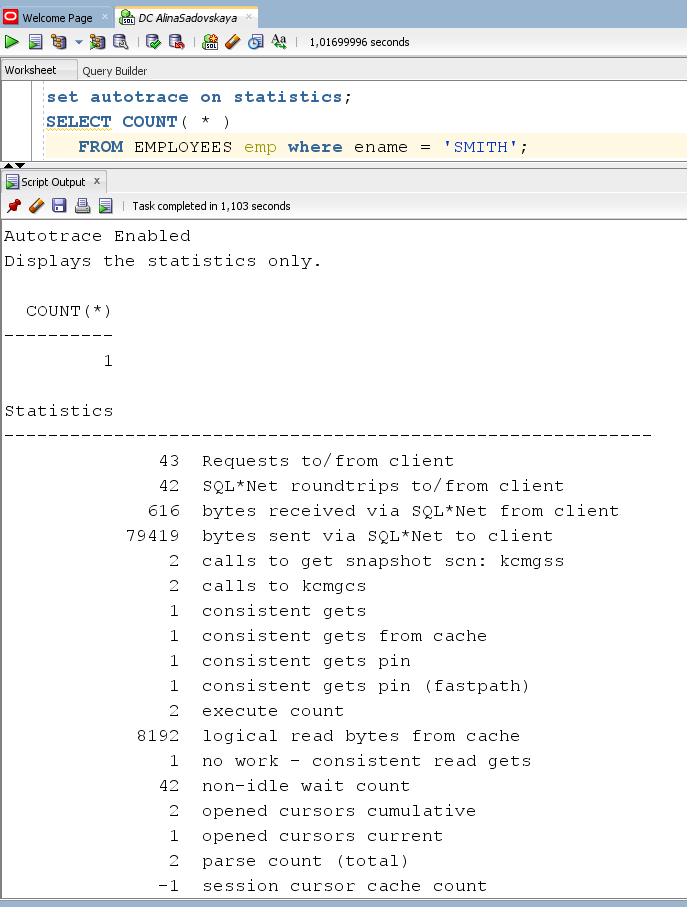


d) set autotrace on explain

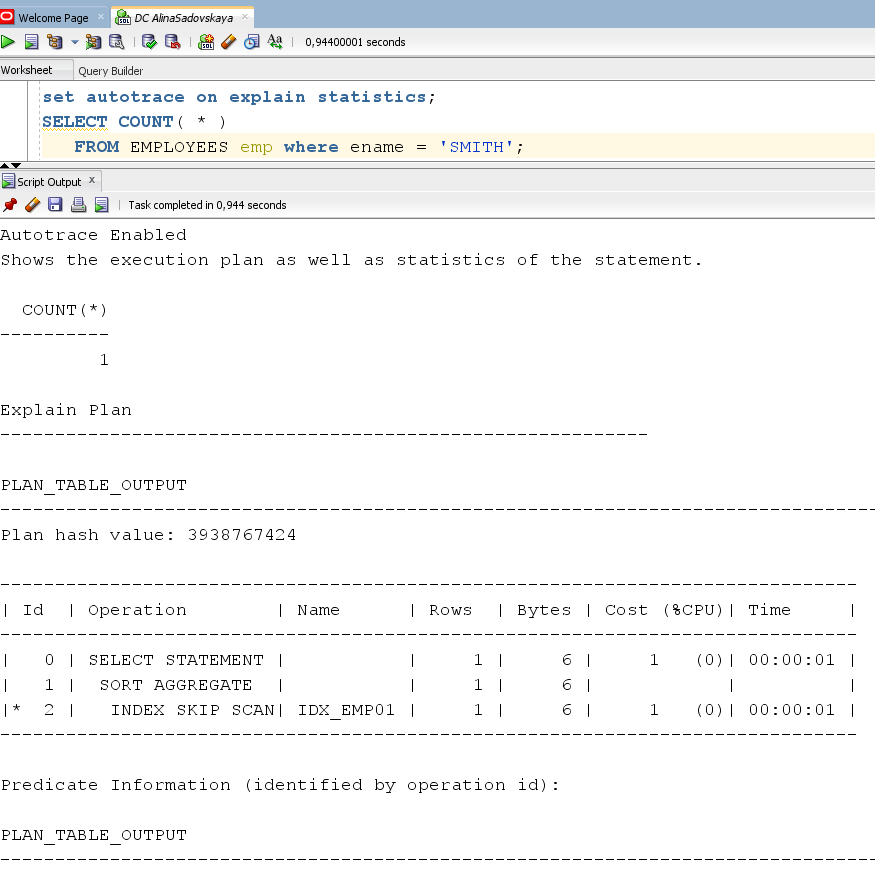


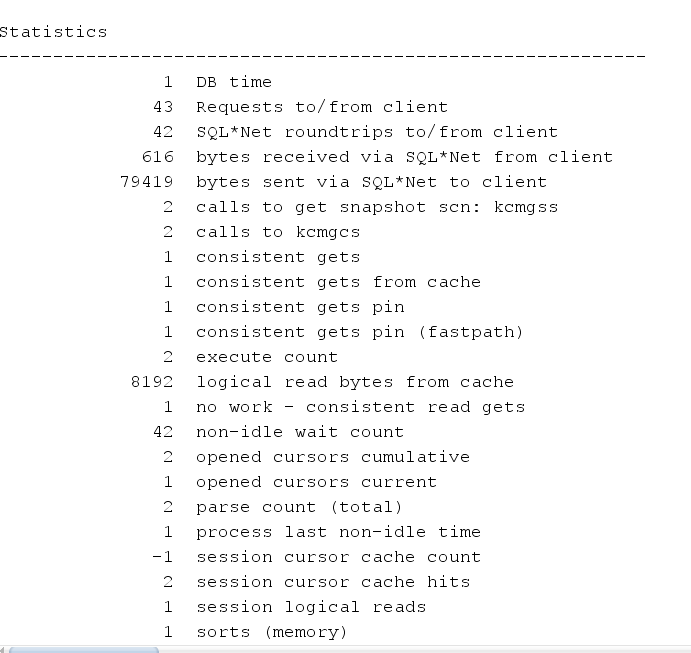


1. set autotrace on statistics



1. set autotrace on explain statistics





g) set autotrace traceonly explain

set autotrace traceonly statistics

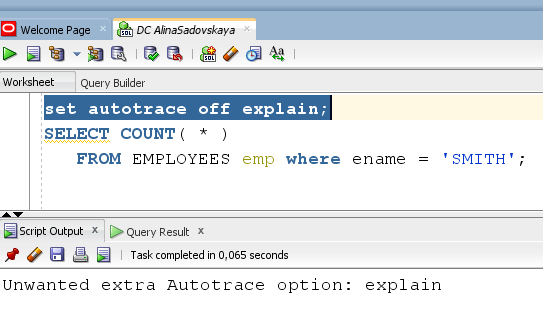
set autotrace traceonly explain statistics

//not supported in sqldeveloper

h) set autotrace off explain

set autotrace off statistics

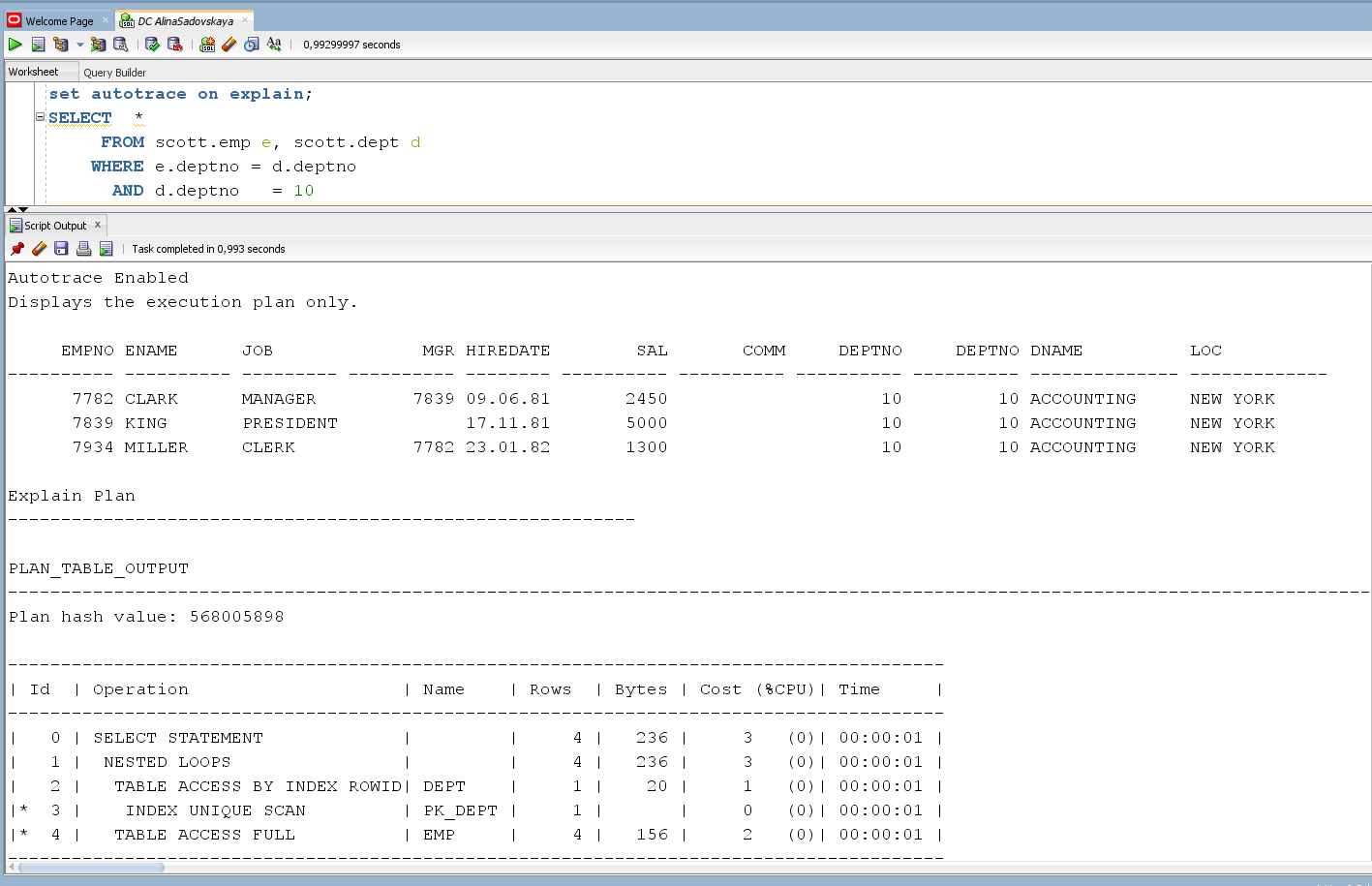
set autotrace off explain statistics



|  |  |  |  |
| --- | --- | --- | --- |
| № | Auto Trace Configuration Options | Expected Results | Description |
| a | set autotrace off | The AUTOTRACE report is not generated. | No report is generated.This is the default. |
| b | set autotrace on | Shows the execution plan as well as statistics of the statement. | The AUTOTRACE report will only display the execution path used by the optimizer. |
| c | set autotrace traceonly | Traceonly option is currently not supported(If you are looking for traceonly option in sqldeveloper then answer is no , they query you executed on oracle CMD will not work on SQL Developer.)  Shows the execution plan as well as statistics of the statement, but suppresses the output of the user's query. | Similar to SET AUTOTRACE ON, but suppresses the output of the user's query, if there is one. |
| d | set autotrace on explain | Displays the execution plan only. | The AUTOTRACE report will only display the execution path used by the optimizer. |
| e | set autotrace on statistics | Displays the statistics only. | The AUTOTRACE report will only display statistics on SQL statement execution. |
| f | set autotrace on explain statistics | Shows the execution plan as well as statistics of the statement. | The AUTOTRACE report will display the execution path used by the optimizer and statistics on SQL statement execution(similar to SET AUTOTRACE ON) |
| g | 1. set autotrace traceonly explain 2. set autotrace traceonly statistics 3. set autotrace traceonly explain statistics | Traceonly option is currently not supported in sqldeveloper.   1. Shows only the execution path used by the optimizer, but suppresses the output of the user's query. 2. Shows only display statistics on SQL statement execution, but suppresses the output of the user's query. 3. Shows the execution plan as well as statistics of the statement, but suppresses the output of the user's query. | 1. Similar to SET AUTOTRACE ON, but suppresses the output of the user's query (if there is one) and also execution statistics. 2. Similar to SET AUTOTRACE ON, but suppresses the output of the user's query (if there is one) and the execution path used by the optimizer 3. Similar to SET AUTOTRACE ON, but suppresses the output of the user's query (set autotrace traceonly). |
| h | * set autotrace off explain * set autotrace off statistics * set autotrace off explain statistics | **Currently not used.** | ------ |

# 2. Join Methods

2.1. Task 2: Nested Loops Joins

****

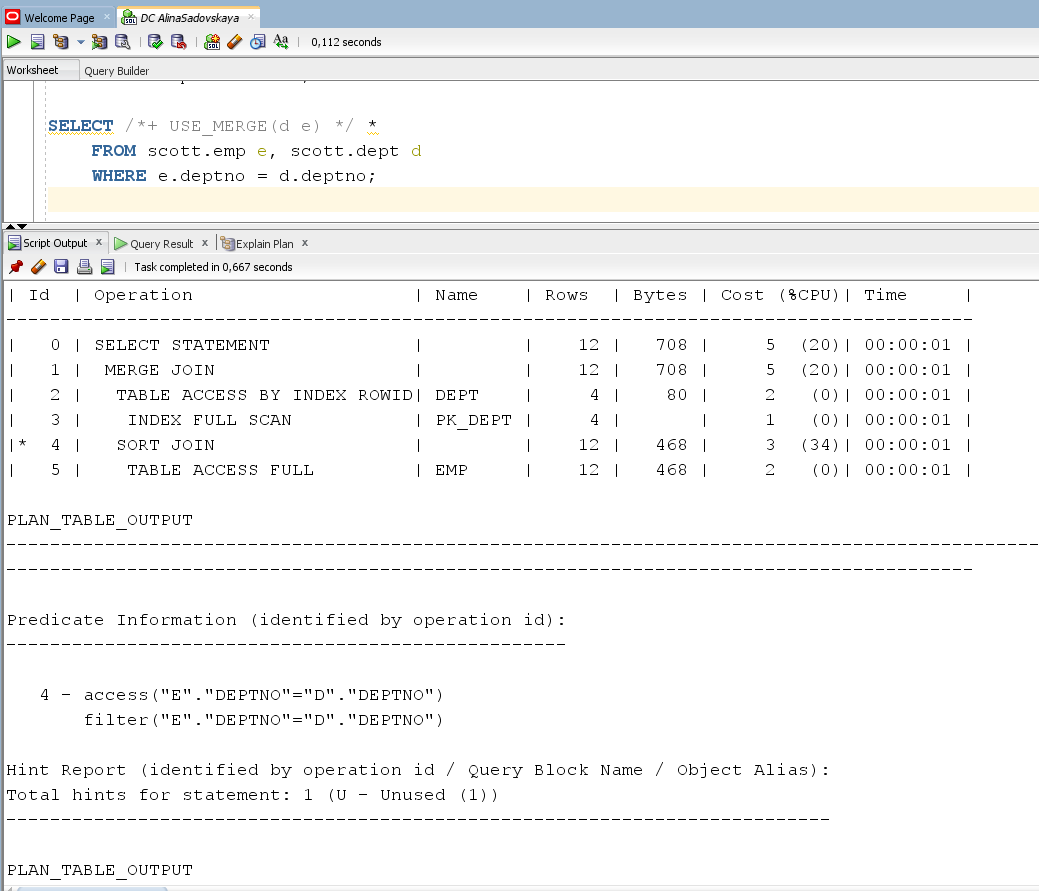
## 

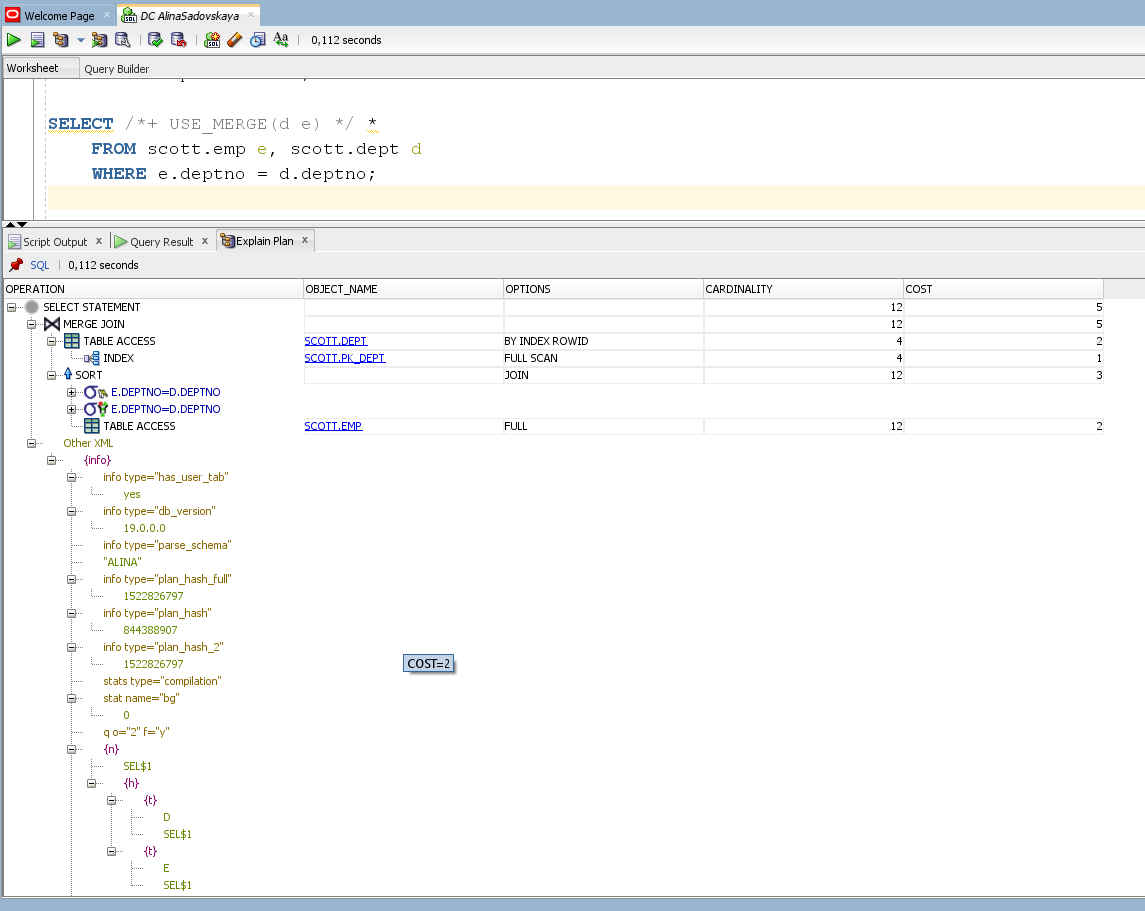
Nested loops joins use each row of the query result reached through one access operation to drive into another table. These joins are typically most effective if the result set is limited in size and indexes are present on the columns used for the join. With nested loops, the cost of the operation is based on reading each row of the outer row source and joining it with the matching row of the inner row source. A nested loops join is, as its name implies, a loop inside a loop. The outer loop is basically a query against the driving table that uses only the conditions from the WHERE clause that pertain to that table.

These kinds of joins are quite robust in that they use very little memory. Since row sets are built one row at a time, there is little overhead required. For that reason, they are actually good for huge result sets except for the fact that building a huge result set one row at a time can take quite a long time.

## 2.2. 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 expensive part for this join method. For large row sources that won’t fit into memory, the sorts will end up using temporary disk space to complete. This can be quite memory and time-consuming to complete. But once the rowsets are sorted, the merge happens quickly. To merge, the database alternates down the two lists, compares the top rows, discards rows that are earlier in the sort order than the top of the other list, and only returns matching rows. Sort-merge joins are typically best suited to queries that have limited data filtering and return lots of rows. They are also often a better choice if there are no suitable indexes that can be used to access the data more directly. Finally, a sort-merge is often the best choice when the join is an inequality (if the row sources are large, the sort-merge will likely be the only viable choice).

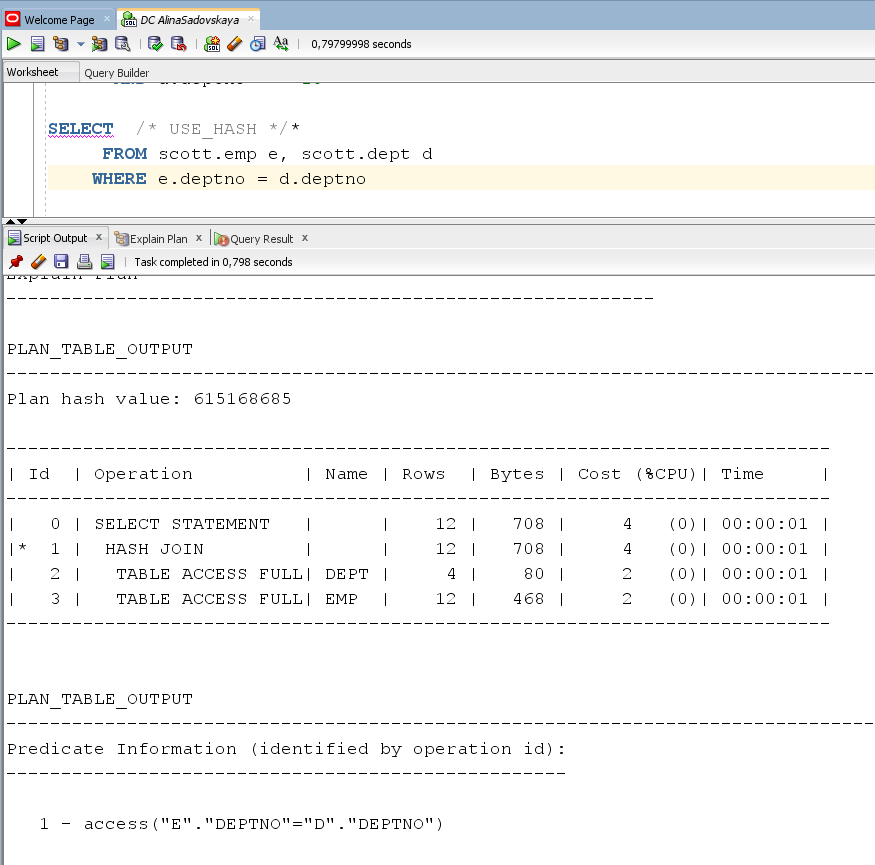


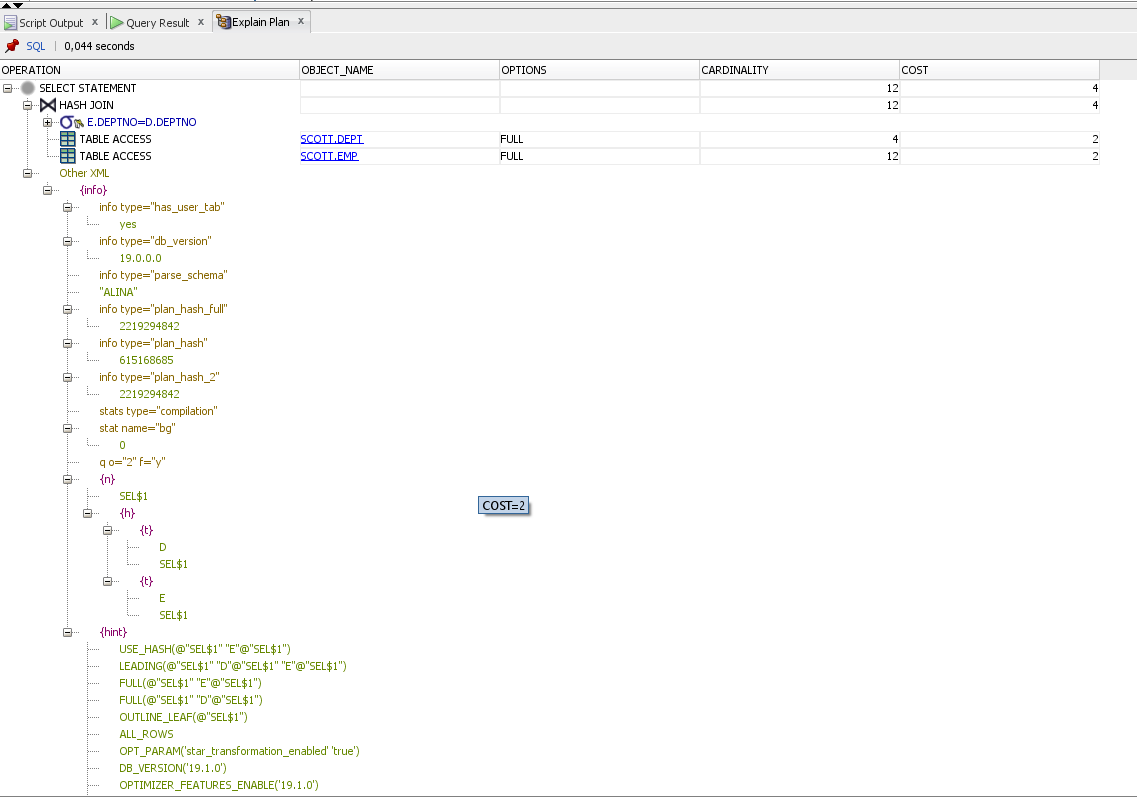


## 2.3. Task 4: Hash Joins

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. As long as there is enough memory available, this hash table will reside in memory. However, if there is not enough memory available, the hash table may be written to temp disk space. 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. Each bucket has a list (represented by a bitmap) of the rows in that bucket. That list is checked for matches with the probing row. If a match is made, the row is returned; otherwise it is discarded. The larger table is read only once and each row is checked for a match. This is different from the nested loops join where the inner table is read multiple times. So really in this case, the larger table is the driving table as it is read only once and the smaller hashed table is probed many times. Unlike a nested loops join plan, however, the tables are listed in the plan output with the smaller hashed table first and the larger probe table second.

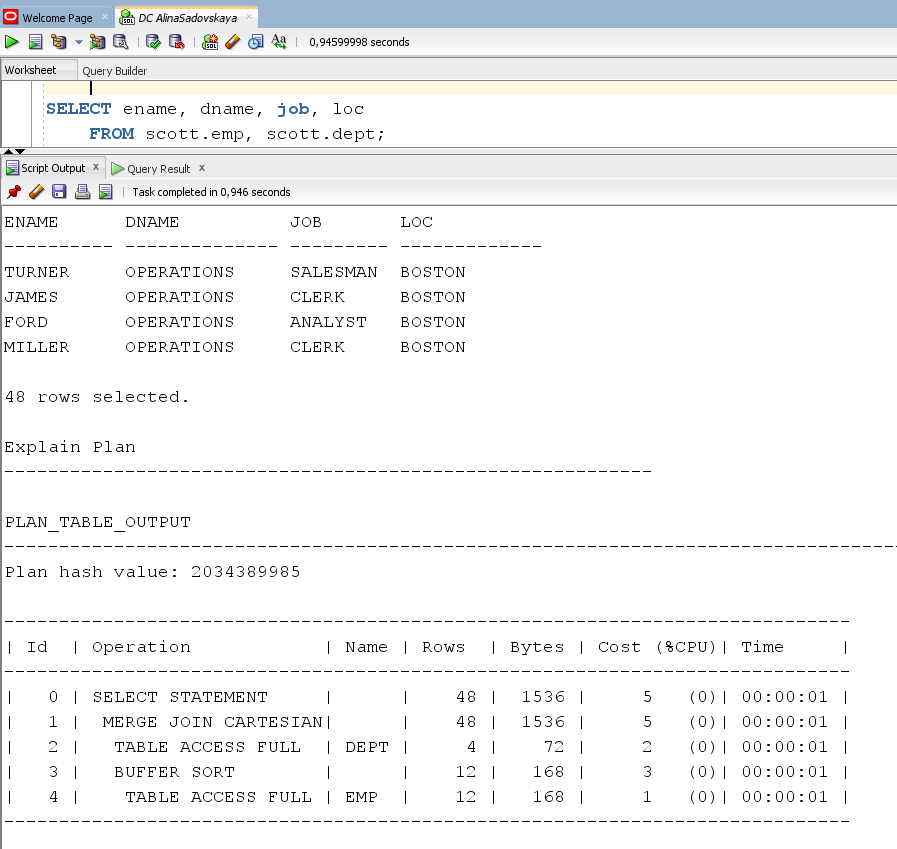
Hash joins are considered more preferable when the row sources are larger and the result set is larger as well. Also, if one of the tables in the join is determined to always return the same row source, a hash join would be preferable since it would only access that table once. If a nested loops join was chosen in that case, the row source would be accessed over and over again, requiring more work than a single independent access. Finally, if the smaller table can fit in memory, a hash join may be favored.

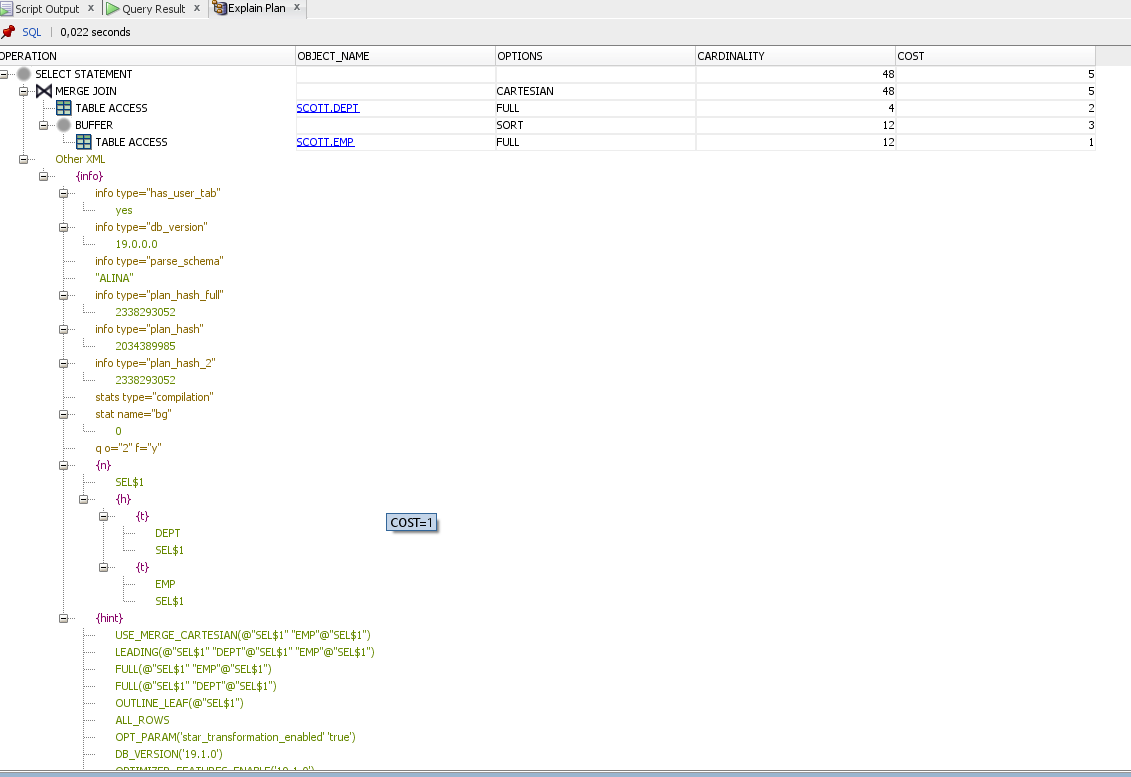
****

****

## 2.4. Task 5: Cartesian Joins

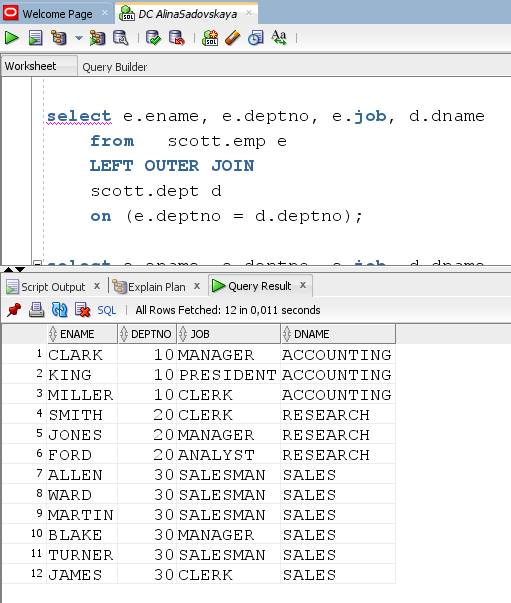
Cartesian joins occur when all the rows from one table are joined to all the rows of another table. Cartesian joins often occur when a join condition is overlooked or left out such that there isn’t a specified join column so the only operation possible is to simply join everything from one row source to everything from the other.

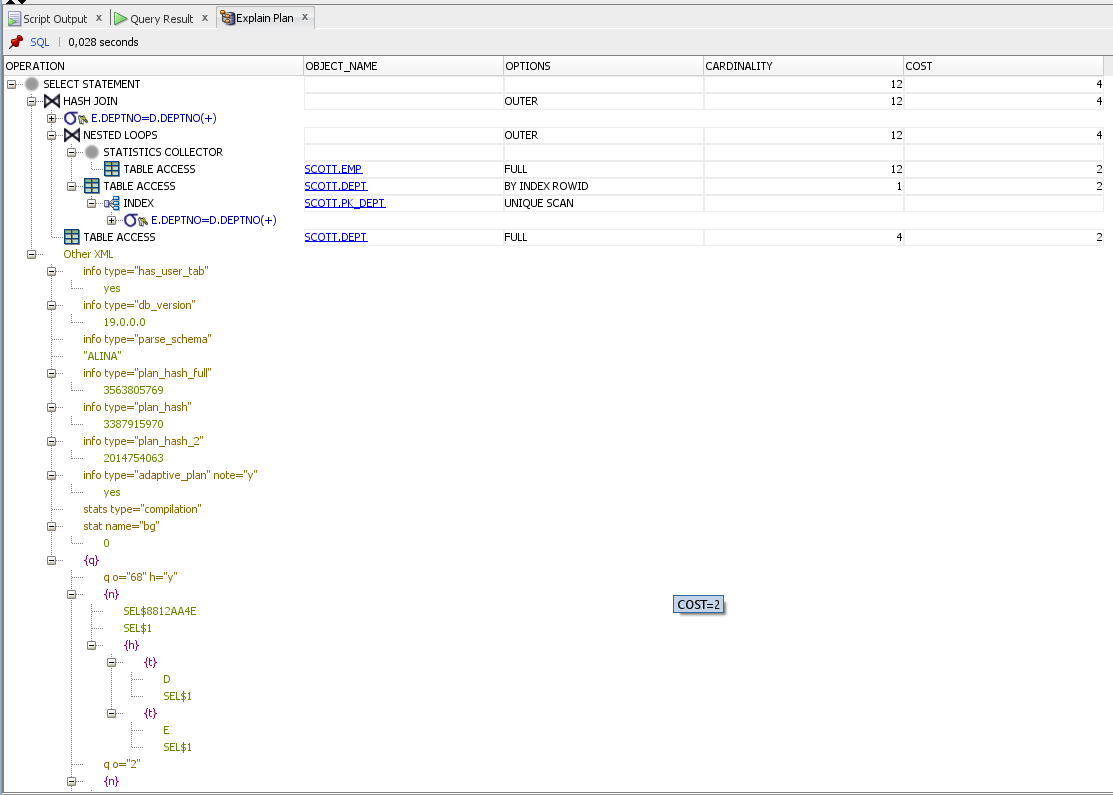
****

****

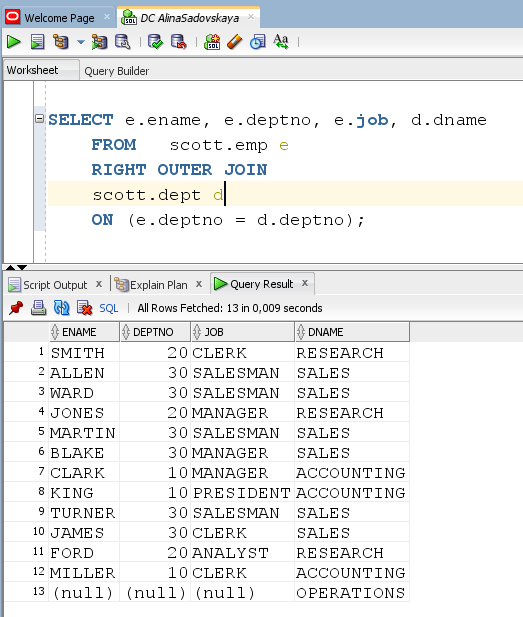
## 2.5. Task 6: Left/Right Outer Joins

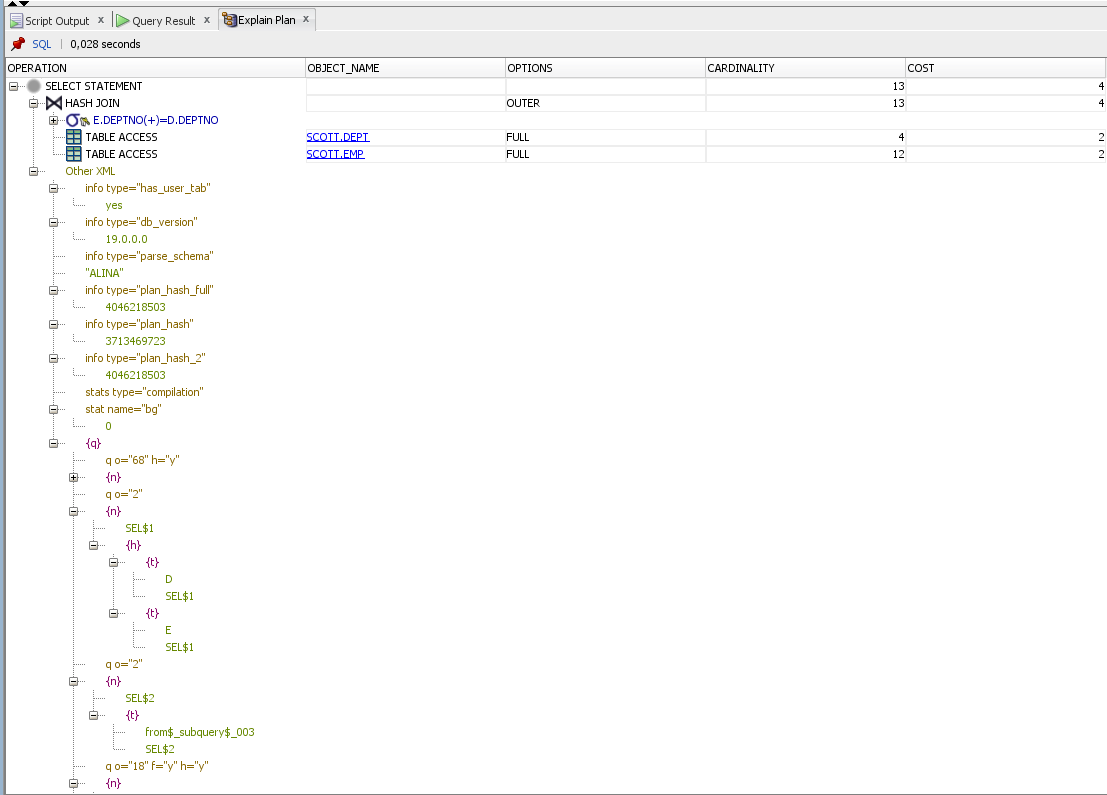
Left Outer Joins

****

****

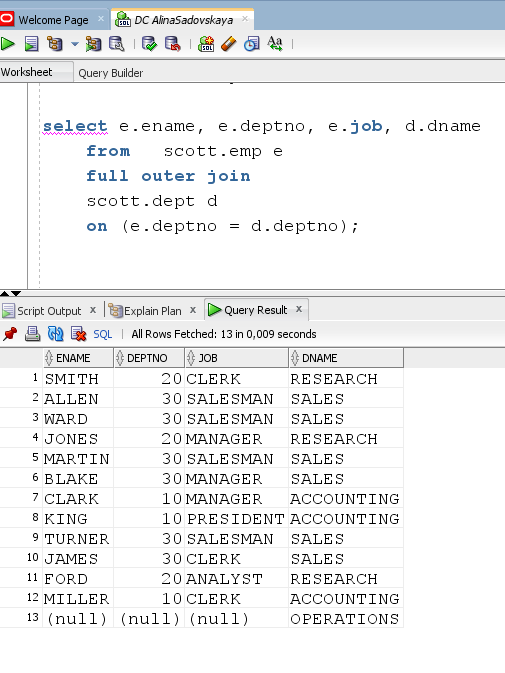
Right Outer Joins

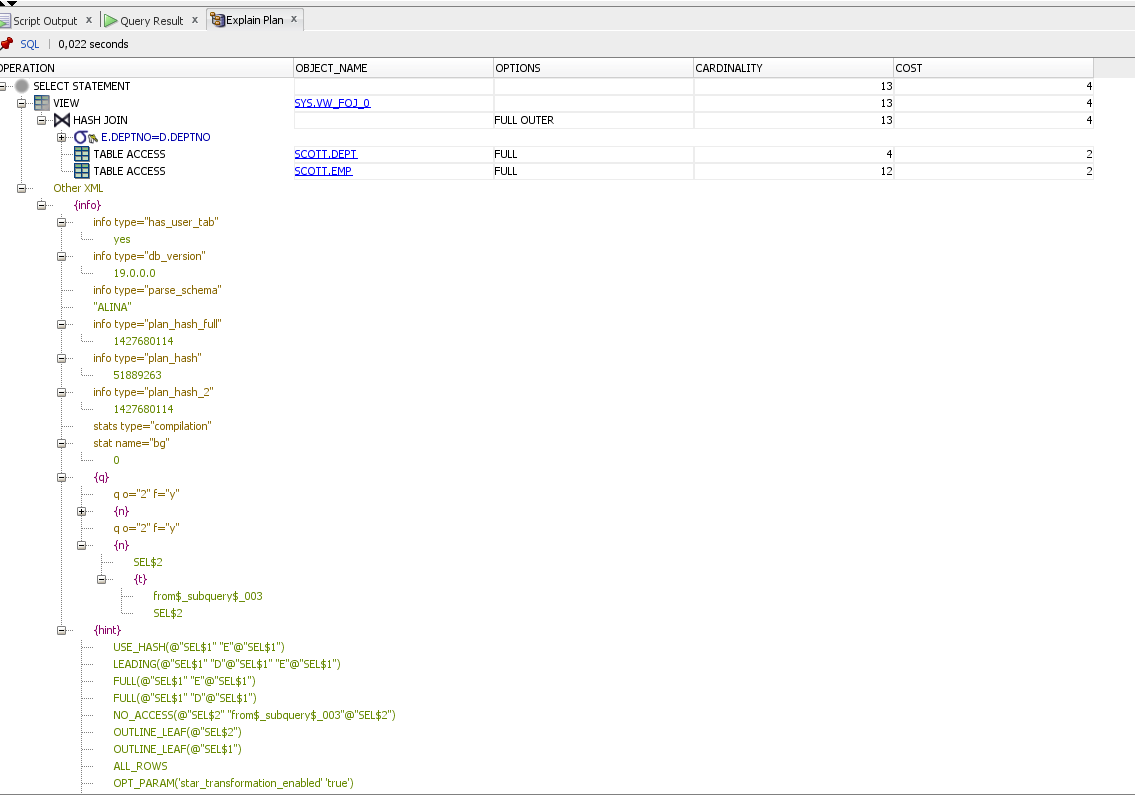
****

****

An outer join returns all rows from one table and only those rows from the joined table where the join condition is met. Oracle uses the + character to indicate an outer join. The + is placed in parentheses on the side of the join condition with the table where only rows that match is located. Outer joins will require that the outer joined table be the driving table. This can mean that join orders that might be more optimal will not be used.

## 2.6. Task 7: Full Outer Join

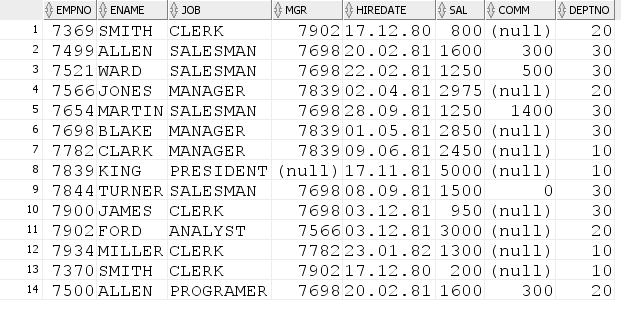
****

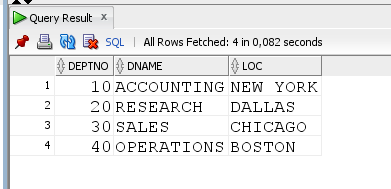
****

Another limitation of Oracle’s outer join syntax is that it doesn’t support full outer joins. A full outer join will join two tables from left-to-right and right-to-left. Records that join in both directions are output once to avoid duplication. The full outer join will return all the rows from both tables that match plus the rows that are unique to each table.

## 2.7. Task 8: Semi Joins

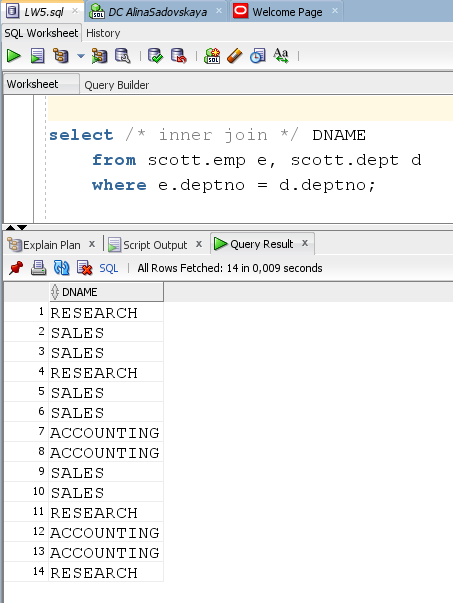
A semi-join is a join between two sets of data (tables) where rows from the first set are returned, based on the presence or absence of at least one matching row in the other set. The main difference between a normal inner join and a semi-join is that with a semi-join, each record in the first set is returned only once, regardless of how many matches there are in the second set. This join technique is a choice that’s available to Oracle’s cost-based optimizer when the query contains a sub-query inside of an **IN** or **EXISTS** clause (or inside the rarely used =ANY clause which is synonymous with IN).

The tables will look like this:



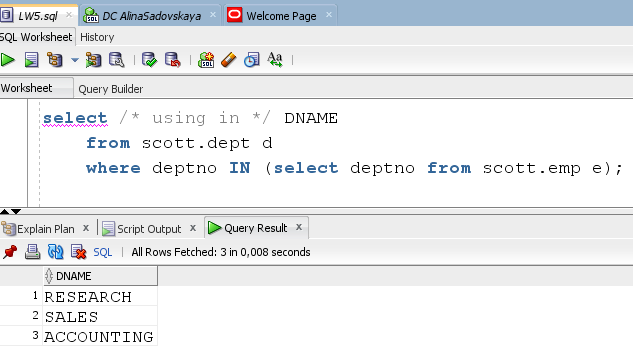
SCOTT.EMP SCOTT.DEPT

Let's apply the INNER JOIN operation and look at the result (the query returns 14 rows):

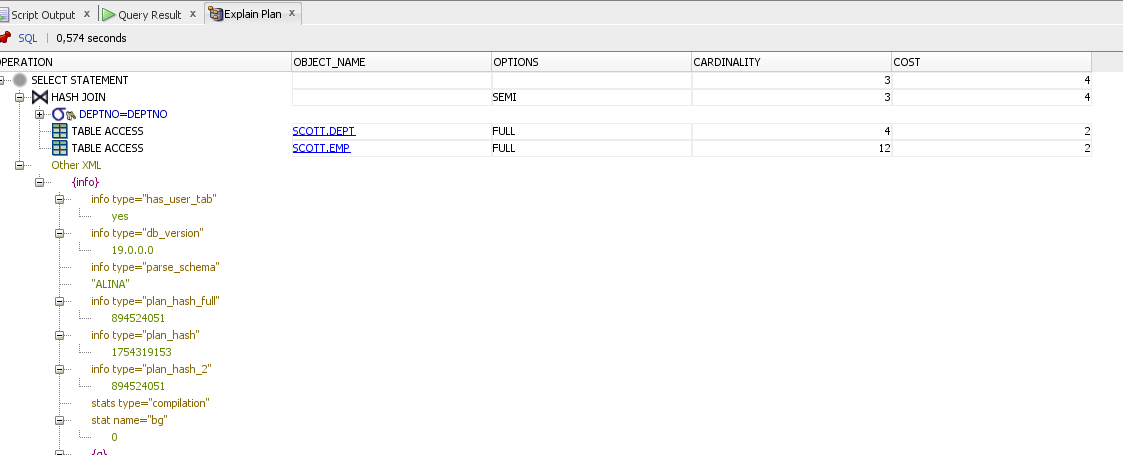


As we can see, data is repeated in the list. To avoid this, use Semi Join.

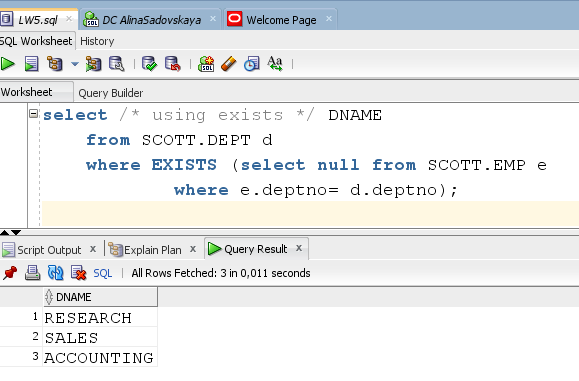
* Apply SEMI Join using IN:

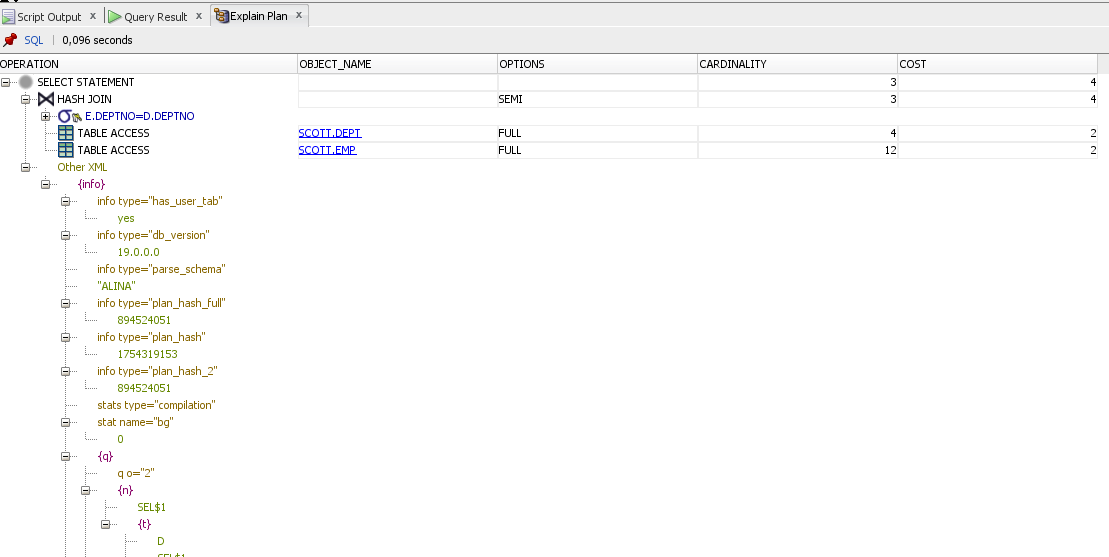


The query returns 3 rows (without repetition).



* Apply SEMI Join using EXISTS:

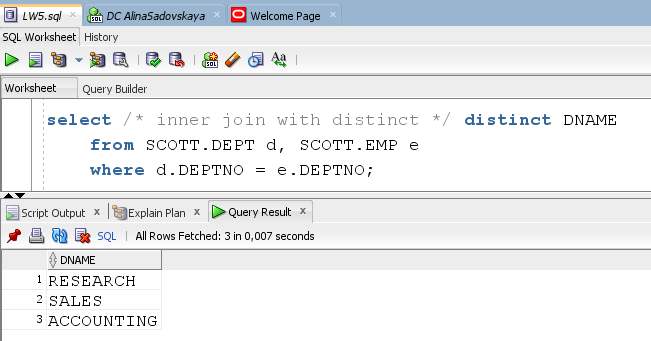


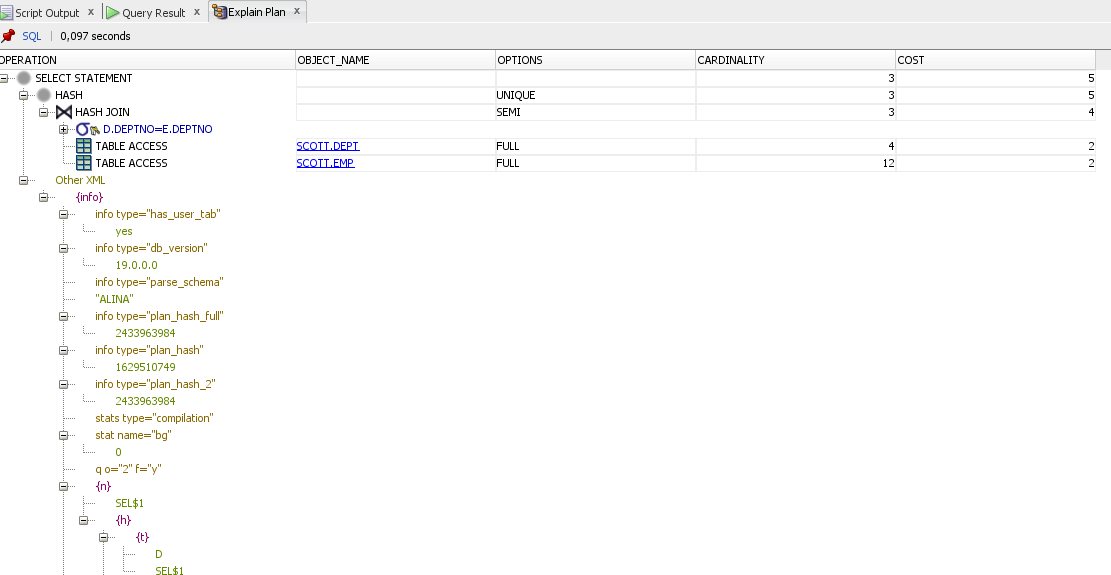


If you would like change in execution plan the type of join method use oracle performance hints.

1. SEMIJOIN – perform a semi-join (the optimizer gets to pick which kind)
2. NO\_SEMIJOIN – obviously means don’t perform a semi-join

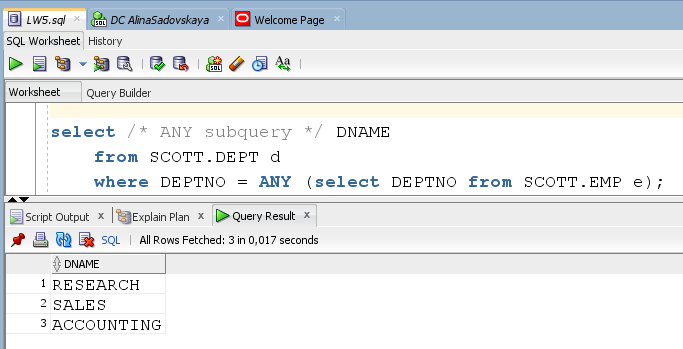
Obviously the inner join is not functionally equivalent to the semi-join due to the number of rows returned. You’ll also notice that there are many repeating values. Let’s try using a DISTINCT to eliminate the duplicates.

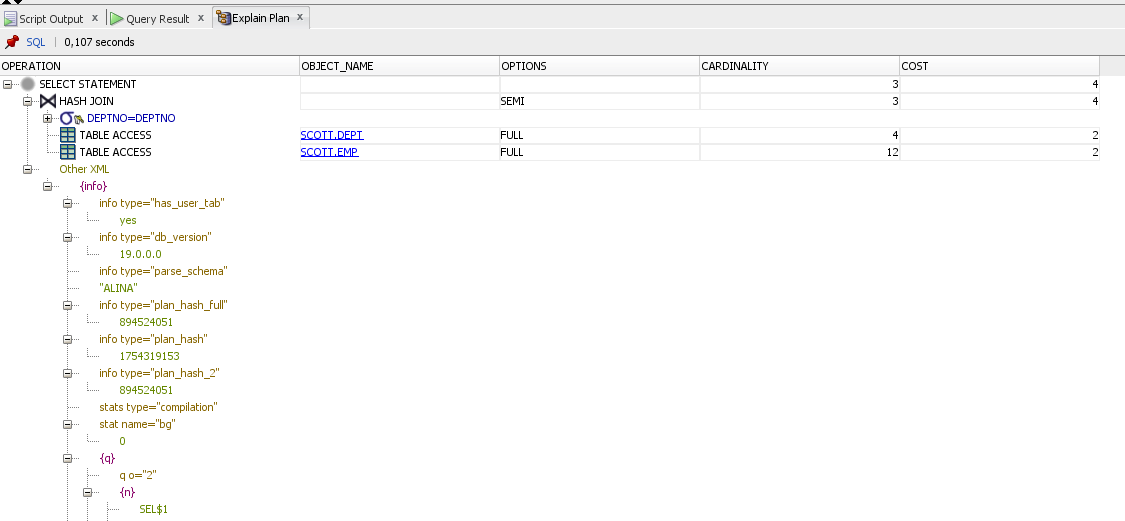




If we pay attention to the Cost column, we will see that when using Inner Join and Distinct, the cost will be higher than when using Semi Join. The IN/EXISTS form takes each record in the first set and, if there is at least one match in the second set, returns the record. It does not apply a DISTINCT operator at the end of the processing (i.e. it doesn’t sort and throw away duplicates). Therefore, it is possible to get repeating values, assuming that there are duplicates in the records returned by Query. The DISTINCT form, on the other hand, retrieves all the rows, sorts them, and then throws away any duplicate values. As you can see from the example, these are clearly not the same. And as you might expect from the description, the DISTINCT version can end up doing significantly more work, as it has no chance to bail out of the subquery early.

* Finally, let’s try the **=ANY** key





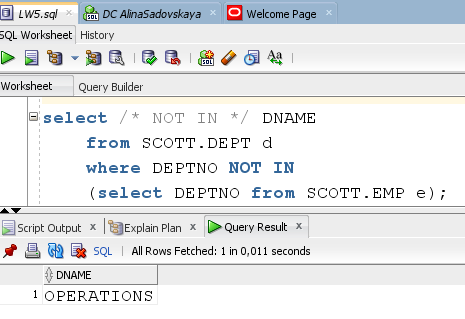
The COST remains the same as when using IN(EXISTS).

## 2.8. 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. However, they are the opposite of semi-joins in terms of the data they return. Those mathematician types familiar with relational algebra would say that antijoins can be defined as the complement of semi-joins. An anti-join returns rows from the left (right) side of the predicate for which there are no corresponding rows on the right (left) side of the predicate. It returns rows that fail to match (NOT IN) the subquery on the right(left) side.

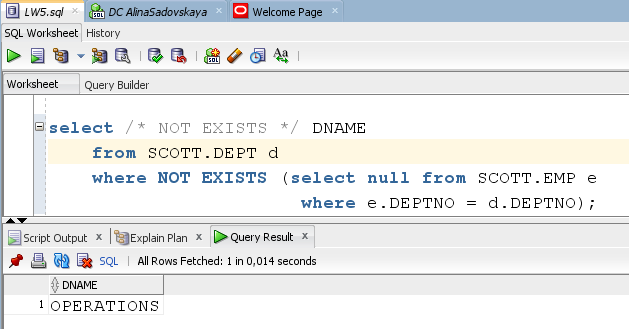
As with semi-joins, there is no specific SQL syntax that invokes an ANTI join. They are one of several choices that the optimizer may use when the SQL statement contains the key words NOT IN or NOT EXISTS (by the way, NOT IN is much more common than NOT EXISTS, probably because it is easier to understand).

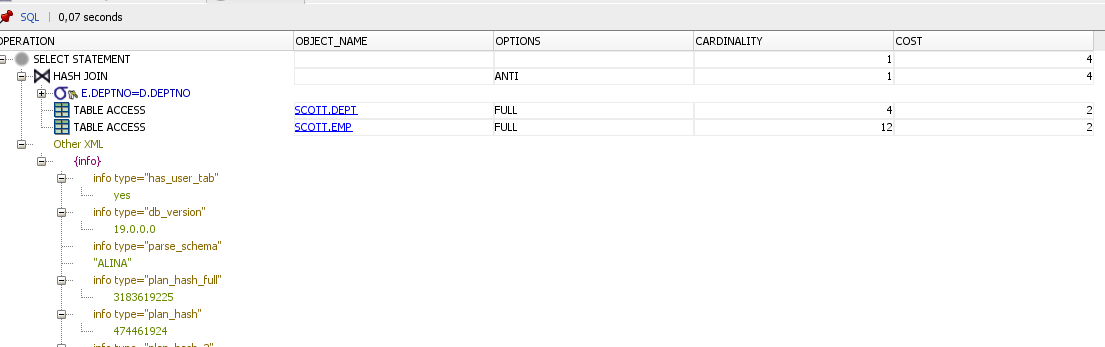
* Apply ANTY Join using NOT IN:





* Apply ANTY Join using NOT EXISTS:

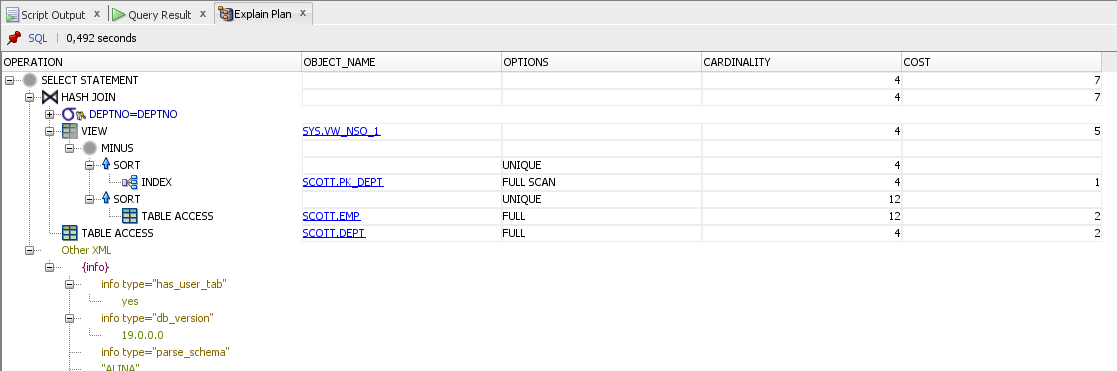
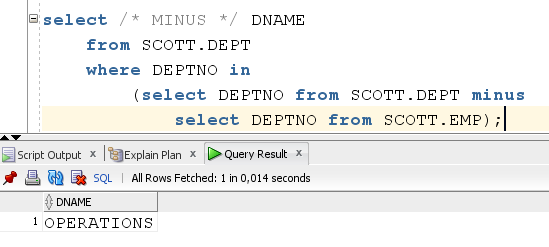




If a null value is returned to a NOT IN operator, then no records will be returned by the overall query. In the first place, the NOT IN operator is just another way of saying !=ANY. So you we think of it as a loop comparing values. If it finds a match, the record is discarded. If it doesn’t, the record gets returned to the user. We know that null is not equal to anything, even another null. In this case, Oracle will return a value of false, even though the theoretical answer is unknown. C.J. Date would probably argue that this is a shortcoming of Oracle’s implementation of relation theory, as it should provide for all three potential answers. At any rate, this is the way it works in Oracle.

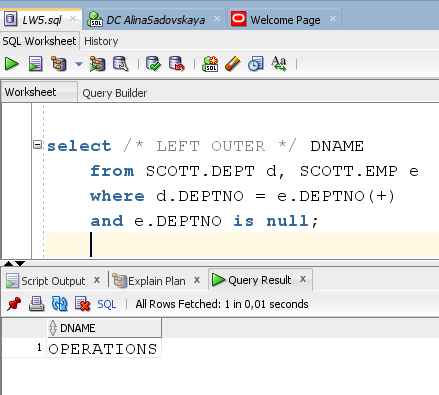
* *Alternative Syntax to NOT IN and NOT EXISTS:*

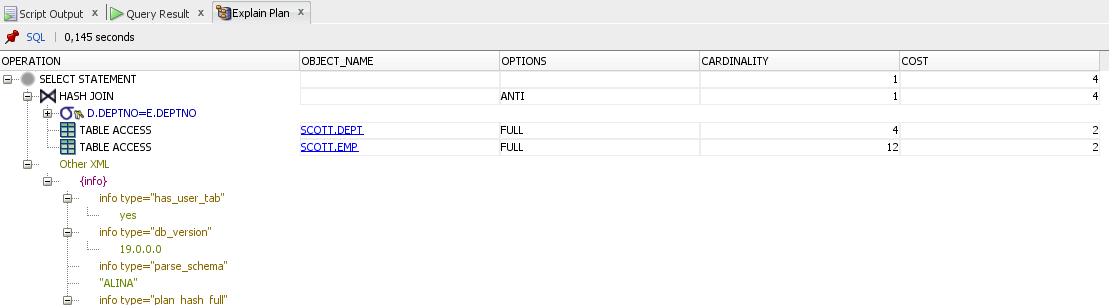
1. *Use MINUS*



As we can see, the cost of the operation has increased significantly.

1. Use LEFT OUTER





If we look at the cost of this method, we will see that it is the same as when using NOT\_IN/NOT\_EXISTS. Even in EXPLAIN PLAN, we see exactly the same form of data traversal.

If you would like change in execution plan the type of join method use oracle performance hints.

1. ANTIJOIN – perform an anti-join (the optimizer gets to pick which kind)
2. USE\_ANTI – older version of ANTIJOIN hint

## 2.9. Task 10: Prepare summary table

 - BEST CASE

 - AVERAGE CASE THAT DEPENDS ON CERTAIN CONDITIONS

- WORST CASE



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Join Access “A” | Join Access “B” | Nested Loop | Hash Join | Sort-Merge Join |
| Small Table | Small Table | Флажок | Флажок  (It will work not bad on small tables, although the cost of building a Map-table is unnecessary) | Флажок  (It will work not bad on small tables, but it will sort the data first) |
| Small Table | Indexed Small Table | Флажок | Флажок  (It will work not bad on small tables, although the cost of building a Hash-table is unnecessary) | Флажок  (It will work not bad on small tables, but it will sorts table A first) |
| Indexed Small Table | Indexed Small Table | Флажок | Флажок  (the cost of building a Hash-table is unnecessary) | Флажок |
| Big Table | Big Table | Флажок | Флажок | Флажок  (it all depends on the sorting costs) |
| Big Table | Indexed Big Table | Флажок | Флажок | Флажок  (it all depends on the sorting costs) |
| Indexed Big Table | Indexed Big Table | Флажок | Флажок | Флажок  (best choice) |
| Small Table | Big Table | Флажок | Флажок  (table A – hash-table) | Флажок  (if the data is sorted) |

Anti-/Semi- Joins are applied depending on the desired result(without repetition) and the selected Join type.