|  |
| --- |
| EPAM Systems, RD Dep. |
| Oracle Join Methods |

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

Contents

[**1.** **Nested Loop Join** 3](#_Toc497936370)

[**2.** **Sort-Merge Join** 4](#_Toc497936371)

[**3.** **Hash Join** 5](#_Toc497936372)

[**4.** **Cartesian Join** 6](#_Toc497936373)

[**5.** **Left/Right Outer Joins** 7](#_Toc497936374)

[5\* Merge Cartesian 8](#_Toc497936375)

[**6.** **Full Outer Join** 9](#_Toc497936376)

[6\* Hash join 10](#_Toc497936377)

[**7.** **Results** 11](#_Toc497936378)

[7\* Nested Loop 12](#_Toc497936379)

[8\* Hash Join 13](#_Toc497936380)

[9\* Left Outer Join 14](#_Toc497936381)

[10\* Nested Loop 15](#_Toc497936382)

## **Nested Loop Join**

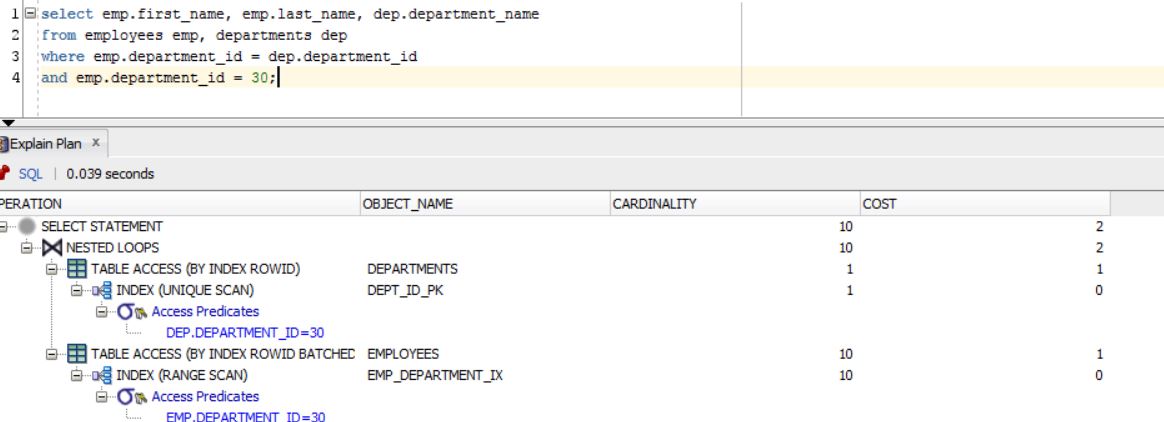
Nested loops joins are useful when the following conditions are true:

1. The database joins small subsets of data, or the database joins large sets of data with the optimizer mode set to FIRST\_ROWS.
2. The join condition is an efficient method of accessing the inner table.

The optimizer always tries to put the smallest row source first, making it the driving 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.

In this select, first Oracle goes to departments table by rowid. Then he understand that department\_id is a primary key so here is used index unique scan. So Oracle comes one by one rows and only ones in departments table

In employees table, Oracle has already knows department\_id and he is just go through the whole table in finding this rows. Then he returns to department table, choose another value and then return to employees and do the same thing as for previous row.



Select emp.first\_name, emp.last\_name. emp.department\_name

from employees emp, departments dep

where emp.department\_id=dep.department\_id

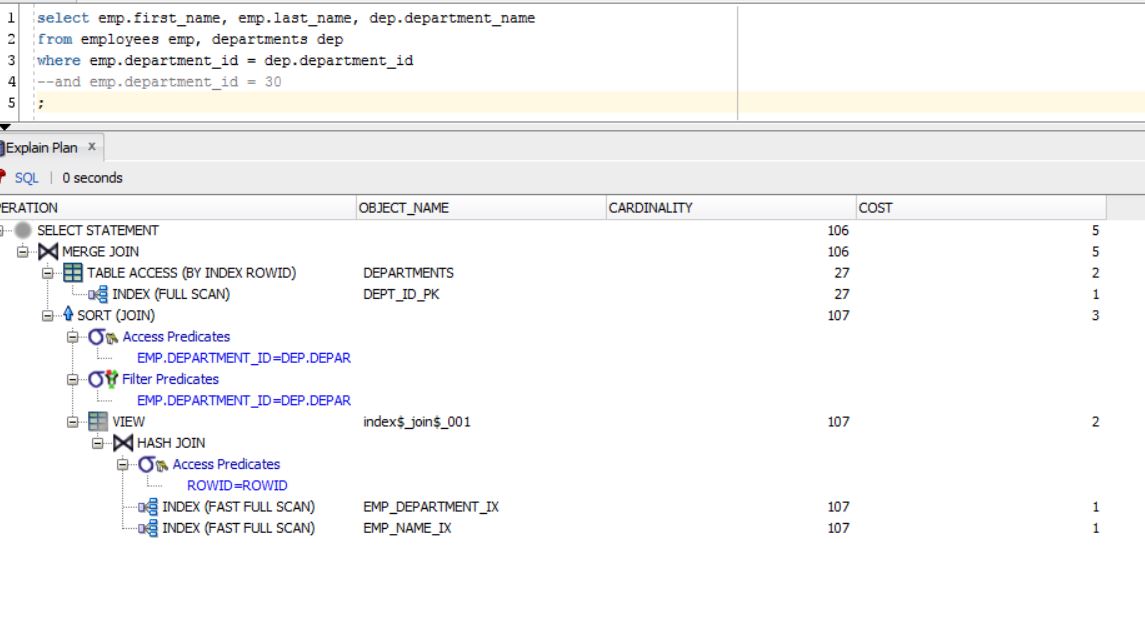
and department\_id=30;

## **Sort-Merge Join**

A sort merge join is a variation on a nested loops join. The database sorts two data sets, if they are not already sorted. For each row in the first data set, the database probes the second data set for matching rows and joins them, basing its start position on the match made in the previous iteration.

A hash join requires one hash table and one probe of this table, whereas a sort merge join requires two sorts.

In this selection, first of all Oracle sorting one table, other values he takes from another table by rowid. But then he did hash join for employees table because he needed to find the whole values which we try to find in select statement by equijoin, which is comparing rowid. And then he does merge join.



Select emp.first\_name, emp.last\_name. emp.department\_name

from employees emp, departments dep

where emp.department\_id=dep.department\_id;

## **Hash Join**

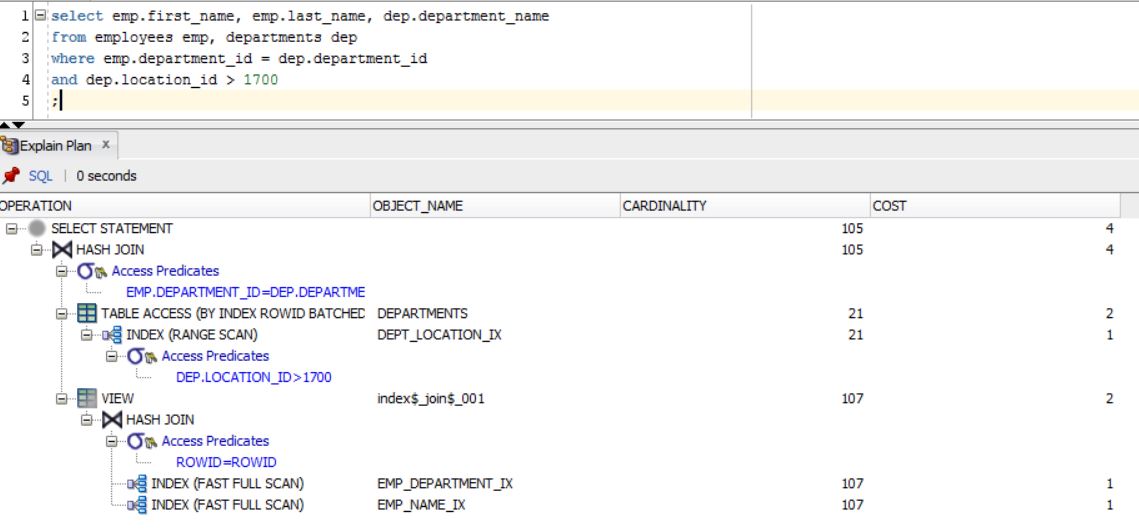
The optimizer considers a hash join when the following conditions are true:

1. A relatively large amount of data must be joined, or a large fraction of a small table must be joined.
2. The join is an equijoin.

A hash join is most cost effective when the smaller data set fits in memory. In this case, the cost is limited to a single read pass over the two data sets.

If the data sets do not fit in memory, then the database partitions the row sources, and the join proceeds partition by partition. This can use a lot of sort area memory, and I/O to the temporary tablespace. This method can still be the most cost effective, especially when parallel query servers are used

Here is an equijoin because we want to find data where location\_id >1700. The same things are going next. He trying to find whole values in select statements by comparing rowid.



Select emp.first\_name, emp.last\_name. emp.department\_name

from employees emp, departments dep

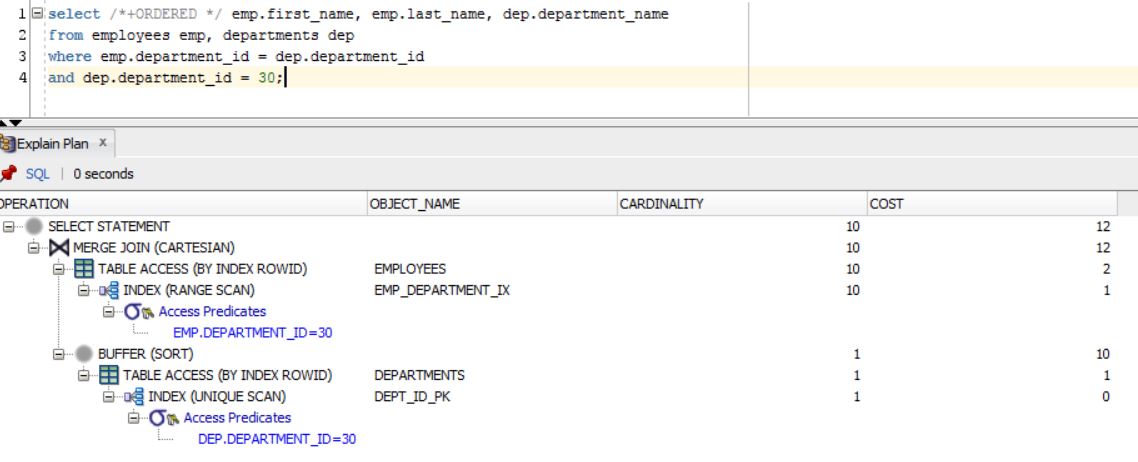
where emp.department\_id=dep.department\_id

and dep.location\_id>1700;

## **Cartesian Join**

The database uses a Cartesian join when one or more of the tables does not have any join conditions to any other tables in the statement. The optimizer joins every row from one data source with every row from the other data source, creating the Cartesian product of the two sets. Therefore, the total number of rows resulting from the join is calculated using the following formula, where rs1 is the number of rows in first row set and rs2 is the number of rows in the second row set.

Here I have done ordered hint and wrong predicate order to try Cartesian merge join.



Select /\*+ORDERED \*/ emp.first\_name, emp.last\_name. emp.department\_name

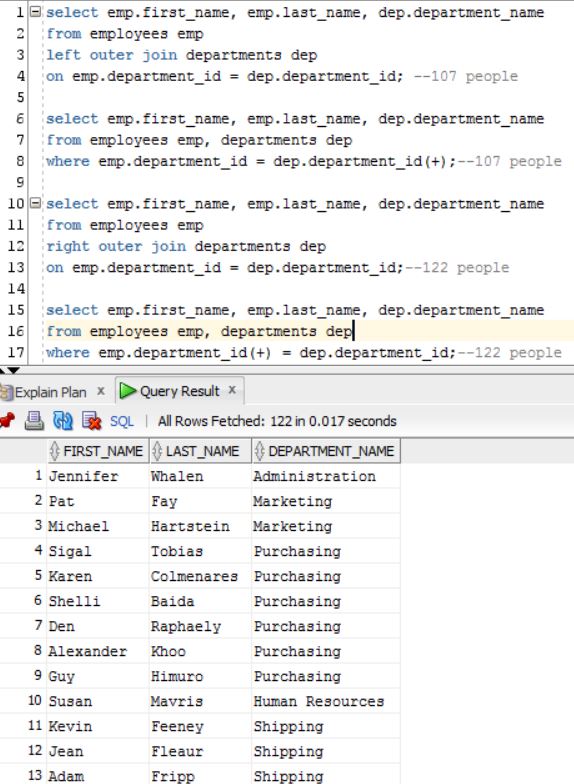
from employees emp, departments dep

where emp.department\_id=dep.department\_id

and dep.department\_id = 30;

## **Left/Right Outer Joins**

The left / right outer join returns all rows from one table, and only those rows from the merged table where the join condition is executed, if it is not matched, then returned null.



select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp

left outer join departments dep

on emp.department\_id = dep.department\_id; --107 people

select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp, departments dep

where emp.department\_id = dep.department\_id(+);--107 people

select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp

right outer join departments dep

on emp.department\_id = dep.department\_id;--122 people

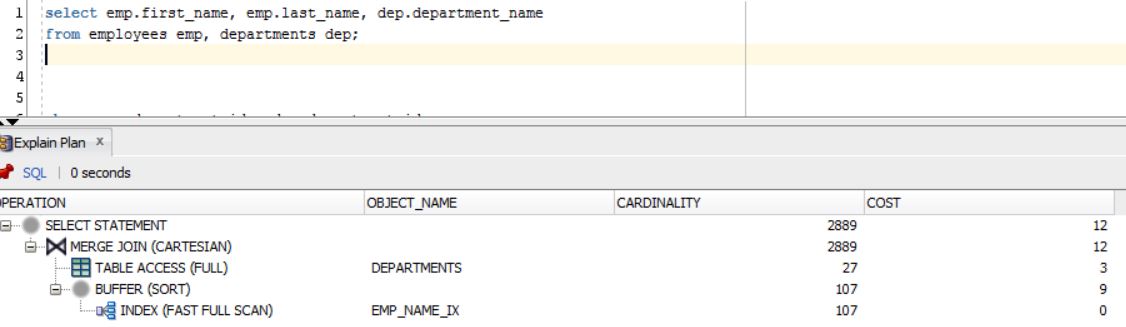
select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp, departments dep

where emp.department\_id(+) = dep.department\_id;--122 people

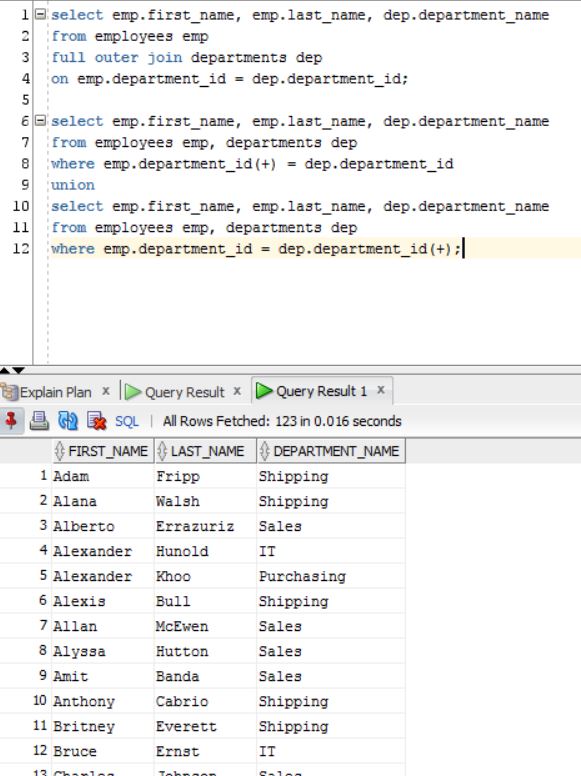
# **5\* Merge Cartesian**

There is no condition for merge



## **Full Outer Join**

A full outer join combines two tables from left to right and from right to left. Records, which are in two tables, are execute once to avoid duplication. Also, executing these rows, from two tables that do not have an equivalent for the fields to be joined in another table.



select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp

full outer join departments dep

on emp.department\_id = dep.department\_id;

select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp, departments dep

where emp.department\_id(+) = dep.department\_id

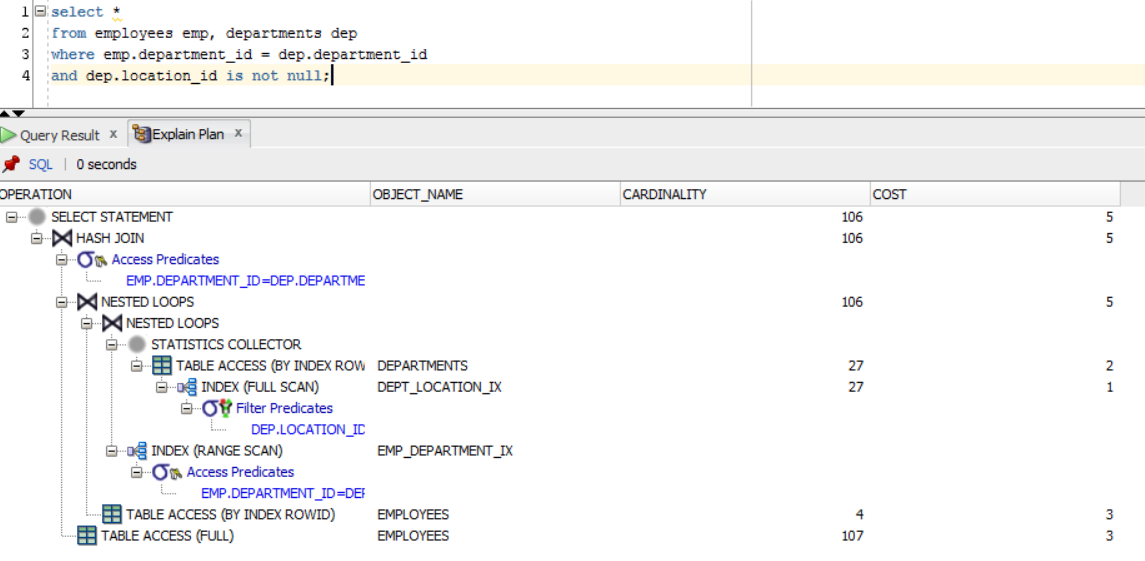
union

select emp.first\_name, emp.last\_name, dep.department\_name

from employees emp, departments dep

where emp.department\_id = dep.department\_id(+);

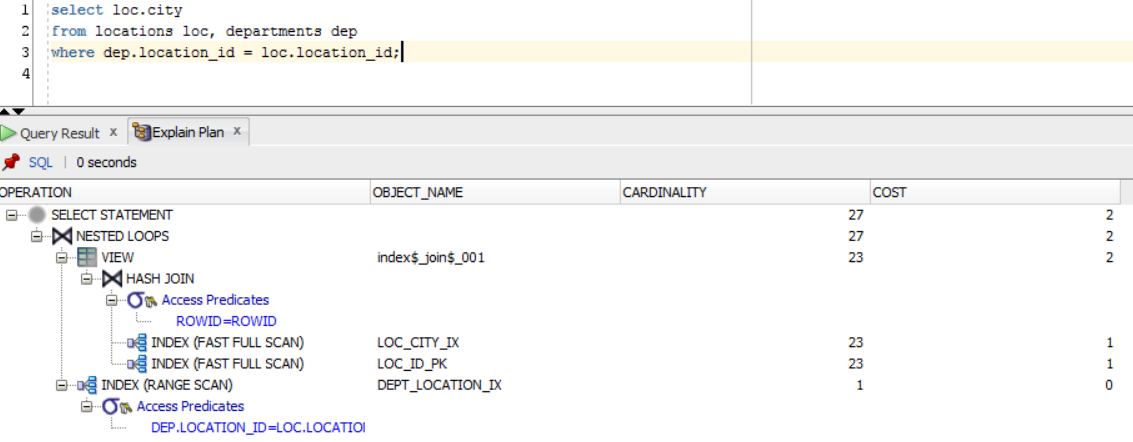
# **6\* Hash join**



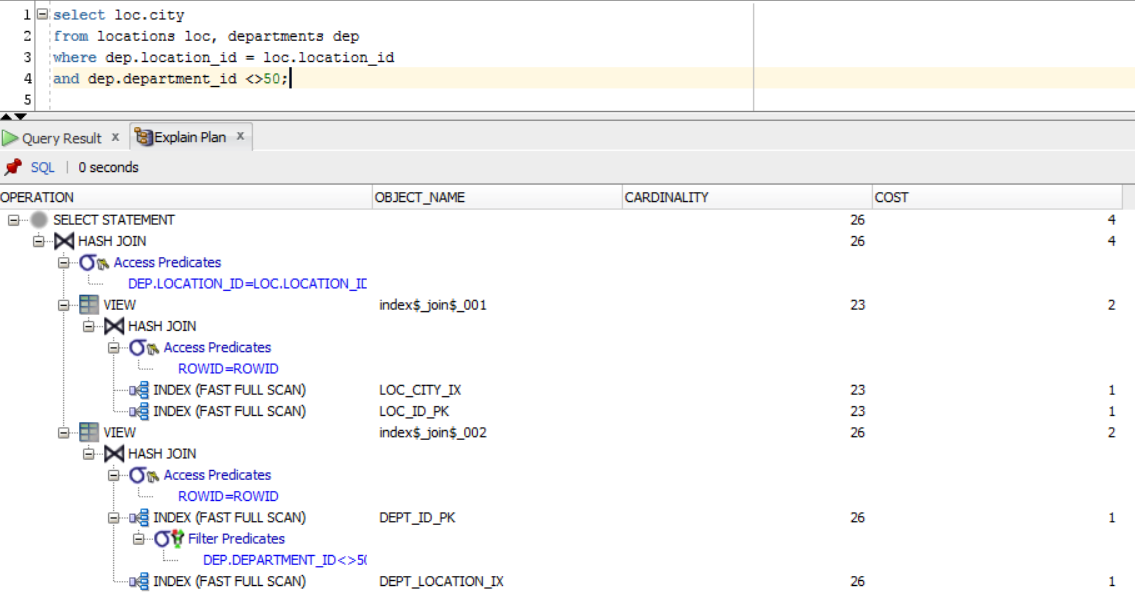
## **Results**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Table “A” | Table “B” | Join type description |
| **1** | Small Table with index on join field | Small Table with index on join field | It will be Nested Loop because there is equal condition and index keys. So we have two tables, where one is bigger than another so we will go through one table by rowid and another one just row by row |
| **2** | Table with index on join field | Table with index on join field | Sort Merge Join. Both tables are quite similar. Join by keys and it is not really important with table to sort |
| **3** | Table with index on join field | Table with index on join field | Hash Join. Because there is a values, which are needed to be > then something. So values from smaller table will be values for temp hash. |
| **4** | Middle table with index on join field | Table with index on join field | Merge Cartesian because we have hint /\* + ORDERED / which means that oracle cannot chouse Nested Loop for this merge. |
| **5** | Any table with any index | Any table with any index | Merge Cartesian because there is no condition for merge |
| **6** | Table with index on join field | Table with index on join field | Hash Join. Here we find values which are not null |
| **7** | Big table with index on join field | Small Table with index on join field | Nested Loop .One table is really big, another one is really small. |
| **8** | Big table with index on join field | Small Table with index on join field | Hash Join. Because there is a values, which are needed to be < > then something. So values from smaller table will be values for temp hash. |
| **9** | Table with index on join field | Table with index on join field | Left Outer Join is working like Hash Join because we use all values from one table and we need to find the same values from another one |
| **10** | Table with index on join field | Table with index on join field | Nested Loop. But it has bigger cost because we try to join by not equal values. |

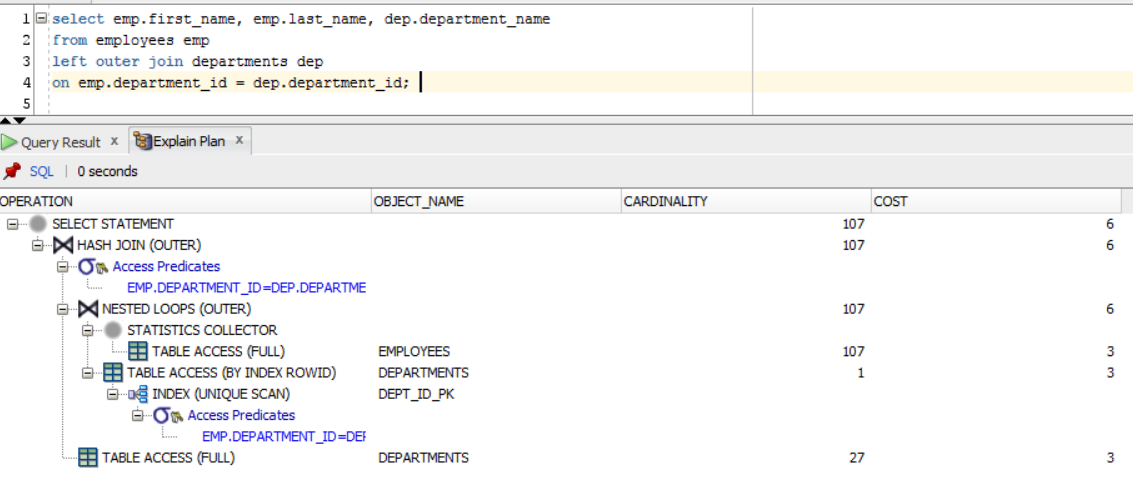
# **7\* Nested Loop**



# **8\* Hash Join**



# **9\* Left Outer Join**



# **10\* Nested Loop**

