|  |
| --- |
| EPAM Systems, RD Dep. |
| MTN.BI.07 Oracle Join Methods |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | [Hanna Klimovich](mailto:Kiryl_Bucha@epam.com) | 8-NOV-2017 |  |  |

**Contents**

[1. Join Methods 3](#_Toc497904028)

[1.1. Task 2: Nested Loop Join 3](#_Toc497904029)

[1.2. Task 3: Sort-Merge Join 3](#_Toc497904030)

[1.3. Task 4: Hash Join 3](#_Toc497904031)

[1.4. Task 5: Cartesian Join 3](#_Toc497904032)

[1.5. Task 6: Left/Right Outer Joins 3](#_Toc497904033)

[1.6. Task 7: Full Outer Join 3](#_Toc497904034)

[1.7. Task 8: Results 3](#_Toc497904035)

# Join Methods

The main task is to try to get different types of different join methods. You can create your own needed objects, or use existing (in sample schemas). If you are creating tables or indexes, please, attach a script.

## Task 2: Nested Loop Join

Example:

SELECT \*

FROM scott.emp e, scott.dept d

WHERE e.deptno = d.deptno

AND d.deptno = 10

**Note:** You can instruct optimized to use nested loop with the hint: USE\_NL.

A nested loops join is particularly effective if the outer input is small and the inner input is preindexed and large. In many small transactions, such as those affecting only a small set of rows, index nested loops joins are superior to both merge joins and hash joins. In large queries, however, nested loops joins are often not the optimal choice.

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.

SQL:

select empno, ename, dname, loc

from emp, dept

where emp.deptno = dept.deptno;

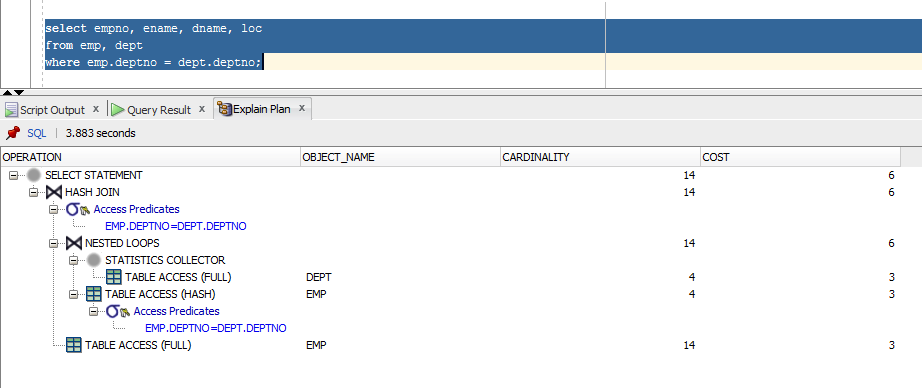


Figure 1 Nested loop join

## Task 3: Sort-Merge Join

**Note:** You can instruct optimized to use sort-merge join with the hint: USE\_MERGE.

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 row sets. 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 row sets 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.

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

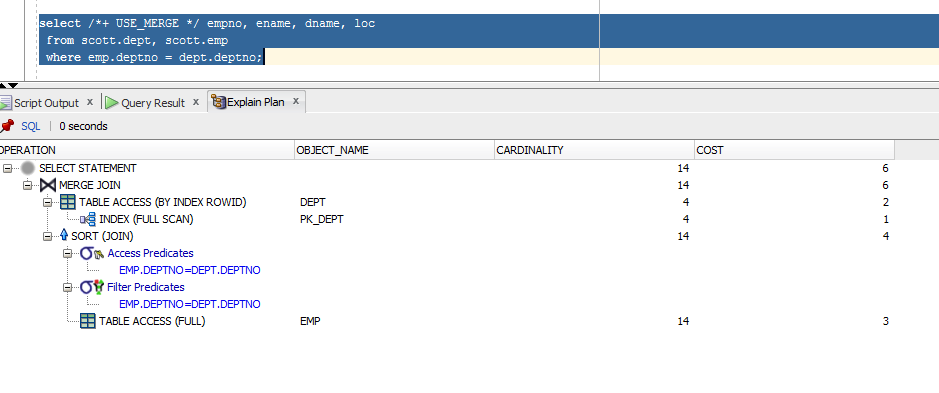


Figure 2 Sort-Merge Join

## Task 4: Hash Join

**Note:** You can instruct optimized to use hash join with the hint: USE\_HASH.

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. 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.

SQL:

explain plan for

select /\*+ USE\_HASH (emp) \*/ empno, ename, dname, loc

from scott.dept, scott.emp

where emp.deptno = dept.deptno;

select \* from table(dbms\_xplan.display);

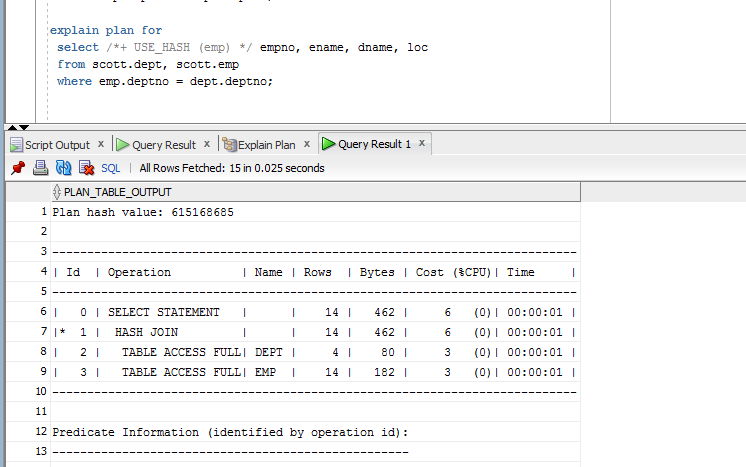


Figure 3 Hash Join

## Task 5: Cartesian Join

Implement Cartesian join with:

* ORDERED hint and wrong predicate order.
* W/o join conditions.

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(A) multiplied by the number of rows in the other table (B) such that A x B = total rows in the result set. 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.

SQL:

explain plan for

select /\*+ USE\_HASH \*/ empno, ename, dname, loc

from scott.dept, scott.emp

select \* from table(dbms\_xplan.display);

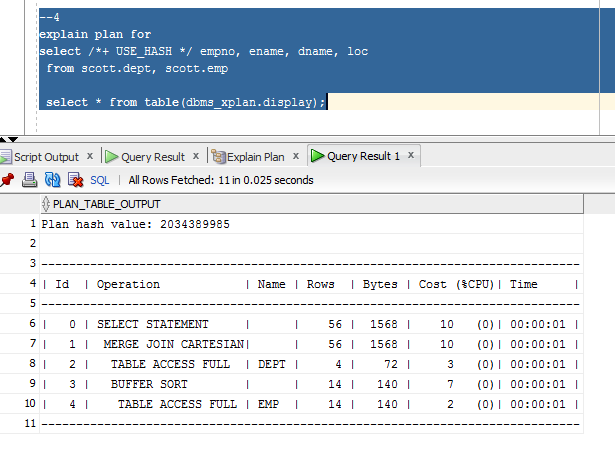


Figure 4 Cartesian Join

## Task 6: Left/Right Outer Joins

Implement Left/Right outer joins with:

* ANSI syntax (left/right join)
* Oracle syntax (+)

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

SCRIPT:

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

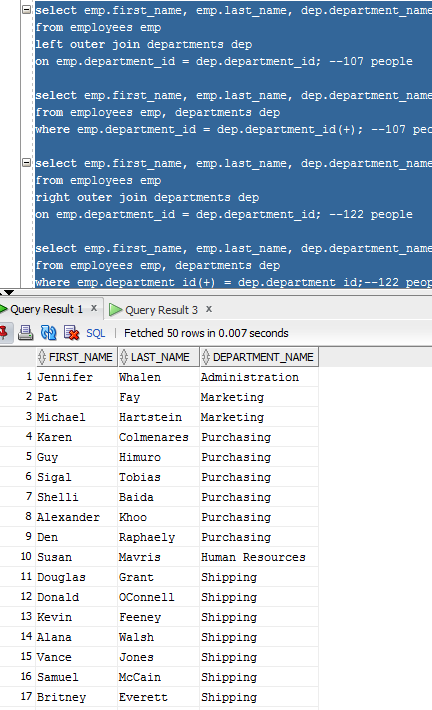
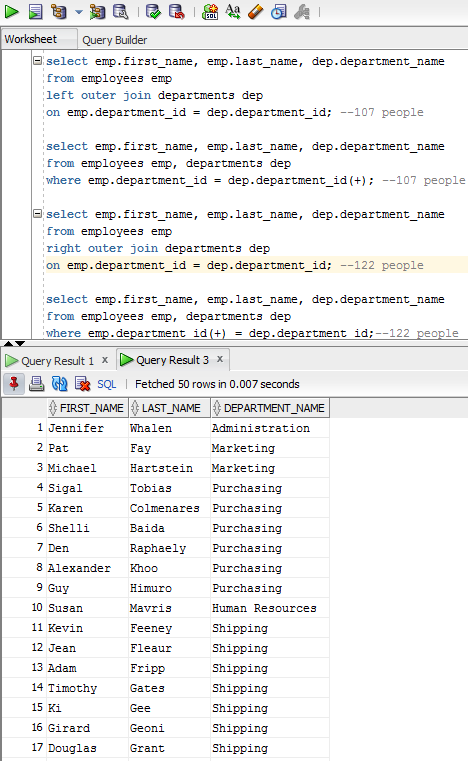
 

Figure 5 Left/Right Outer Join

## Task 7: Full Outer Join

Implement outer join with:

* ANSI syntax (outer join)
* Oracle syntax (+)

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.

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(+);

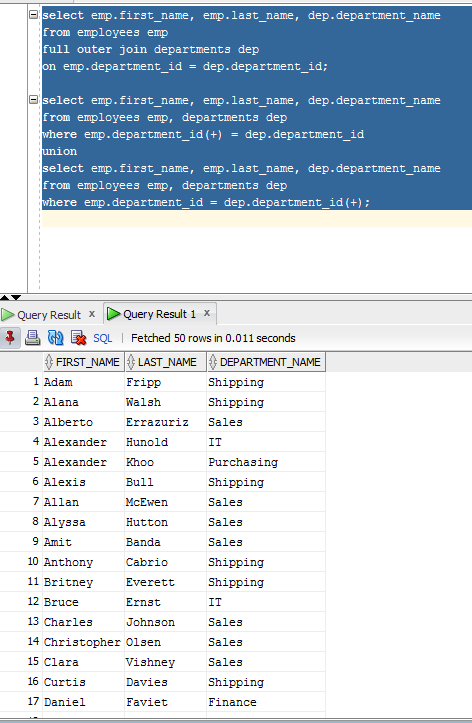
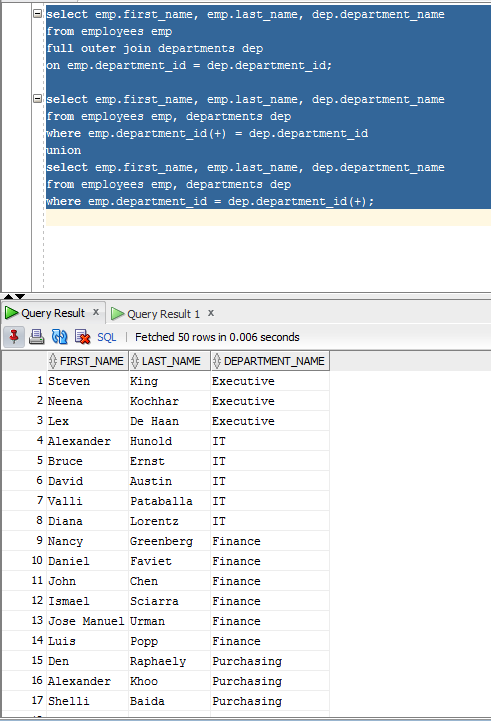


Figure 6 Full Outer Join

## Task 8: Results

* Screenshots, SQLs, descriptions for Tasks 2-7.
* Make a list of combination of different tables (see examples below) and write a description for the join between them. Express your opinion on why the selected join type was chosen by an optimizer. Create at least 10 variants.

|  |  |  |  |
| --- | --- | --- | --- |
| № | 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. |