Anton Slizh’s

U1M5.LW.Access and Join Methods Part 2

**Task 2**

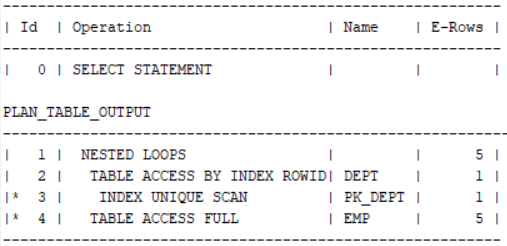
**Task 2.1 Nested Loops Joins**

SELECT \*

FROM scott.emp e, scott.dept d

WHERE e.deptno = d.deptno

AND d.deptno = 10



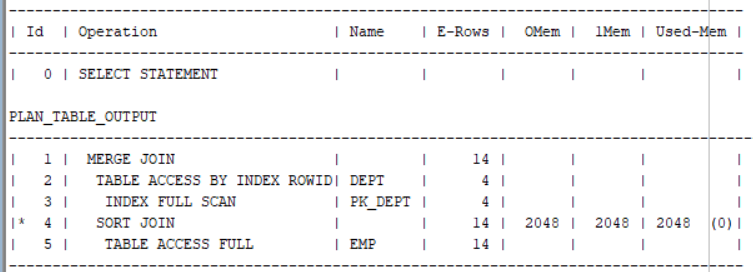
Nested Loops Join works like this: Oracle takes the first value from the first table (emp) and compares it with each value in the second internal table (dept) in search of a match. When each value from the internal table (dept) is checked, Oracle moves to the next value in the external table (emp), and the process repeats until each value from the external table is compared with each value from the internal table. Using indexes may significantly increase efficiency of join.

**Task 2.2 Sort-Merge Joins**

SELECT /\*+ USE\_MERGE\*/ \*

FROM scott.emp e, scott.dept d

WHERE e.deptno = d.deptno;



In a Sort-Merge join, Oracle sorts the first row source by its join columns, sorts the second row source by its join columns, and then merges the sorted row sources together.

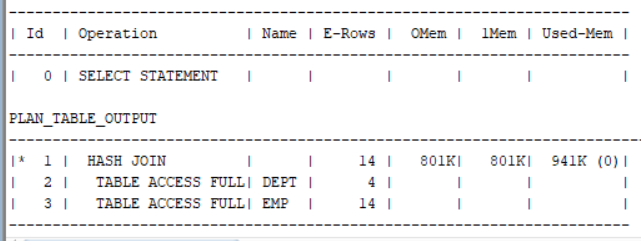
At prepared execution plan oracle didn’t sort DEPT table and used index to access. The data from EMP table should be sorted before merge, that’s we see at SORT JOIN operation. After data from two tables was sorted, Oracle performs MERGE operation.

**Task 2.3 Hash Joins**

SELECT /\*+ USE\_HASH(e, d)\*/ \*

FROM scott.emp e, scott.dept d

WHERE e.deptno = d.deptno;

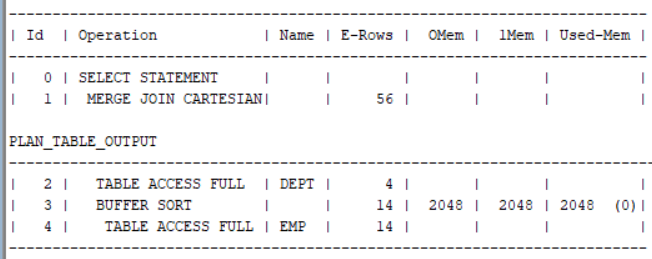


At hash join first step is to read the table that is determined to return the fewest rows and hash it into memory. The next step is for the other larger table to be read and the hash function is applied to the join key column. That hash value is then used to probe the smaller in memory hash table for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

**Task 2.4 Cartesian Joins**

SELECT \*

FROM scott.emp e, scott.dept d;



In other words cartesian join is the cross join, when all rows from one table are joined to all rows from another table.

At the prepared execution plan the SORT is misleading as no SORT takes place. The BUFFER SORT operation indicates that data is stored in the PGA rather than the buffer cache because the data is accessed multiple time and storing it in the buffer cache might cause contention issues with other processes that want to access the buffer cache as well.

**Task 2.5 Left / Right Outer Joins**

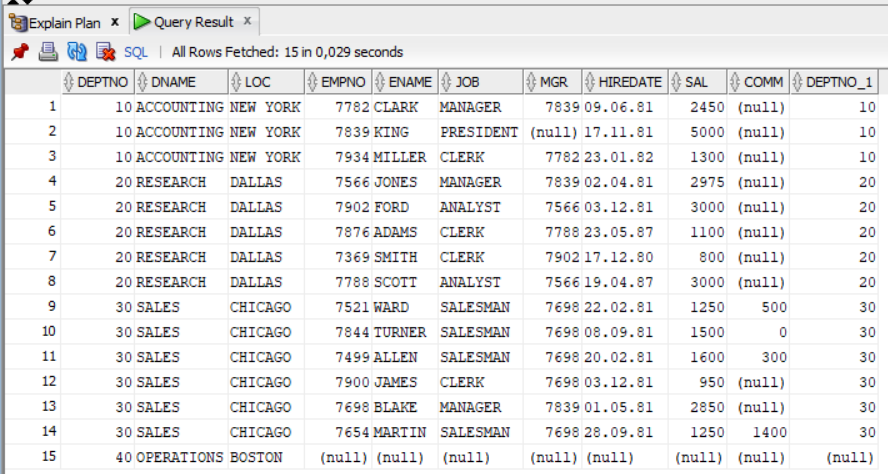
Left Outer Join

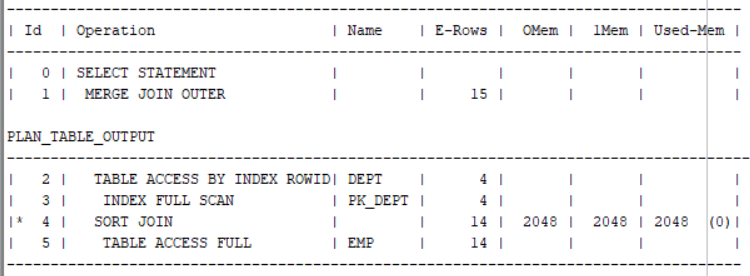
select \*

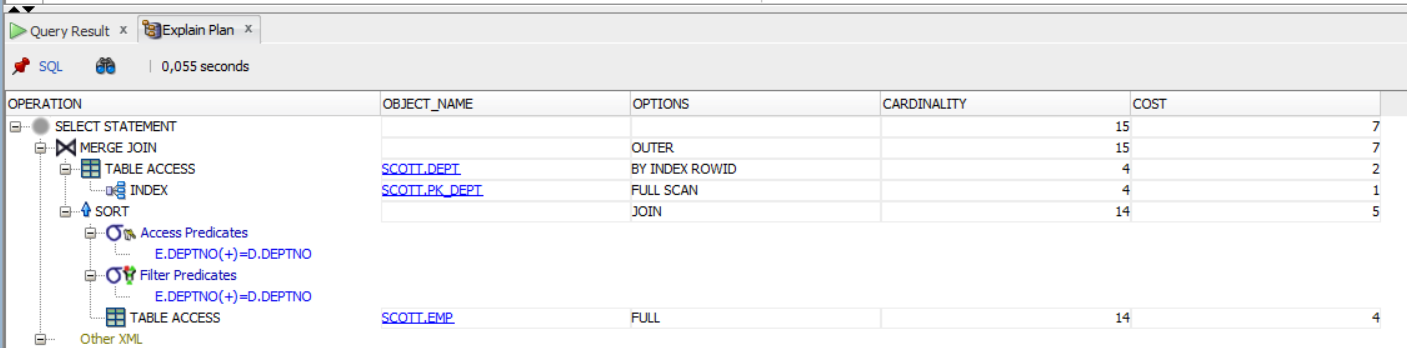
FROM scott.dept d

LEFT JOIN scott.emp e

ON e.deptno = d.deptno ;







Left join returns all rows from left table and only rows from joined table where the join condition is met.

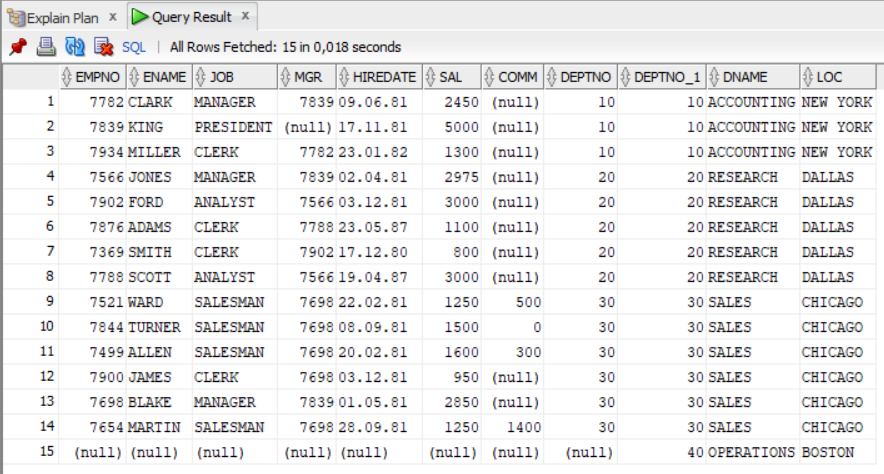
Right Outer Join

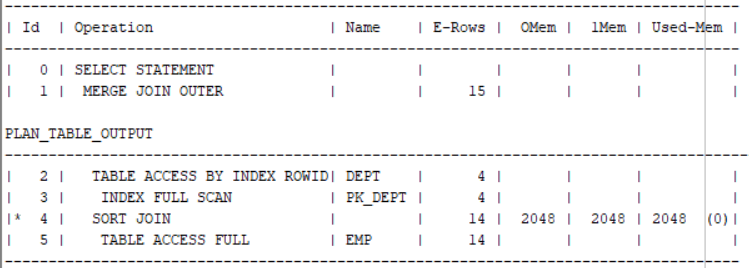
SELECT \*

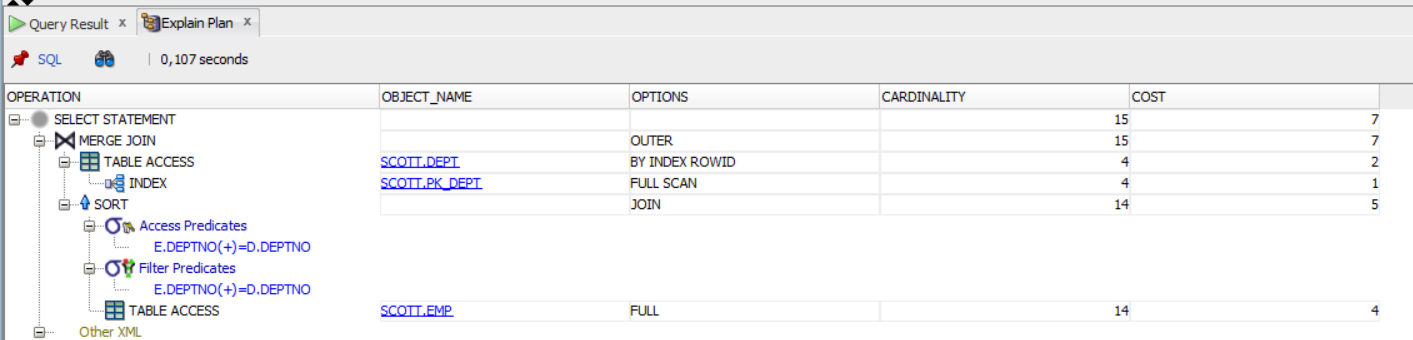
FROM scott.emp e

RIGHT JOIN scott.dept d

ON e.deptno = d.deptno ;







Right join returns all rows from joined table and only rows from left table where join condition is met.

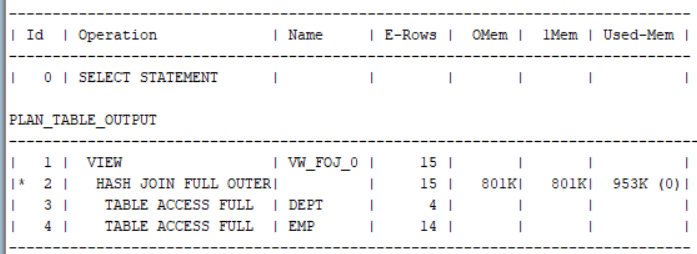
**Task 2.6 Full Outer Join**

SELECT \*

FROM scott.emp e

FULL OUTER JOIN scott.dept d

ON e.deptno = d.deptno;



The FULL OUTER JOIN  returns all matching records from both tables whether the other table matches or not.

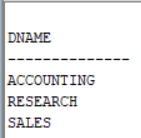
**Task 2.7 Seim Joins**

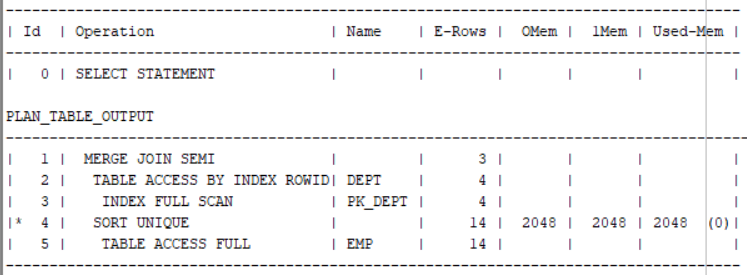
SELECT dname

FROM SCOTT.dept dept

WHERE deptno IN

(SELECT deptno FROM scott.emp );





A semi-join between two tables returns rows that match an EXISTS subquery without duplicating rows from the left side of the predicate when multiple rows on the right side satisfy the criteria of the subquery.

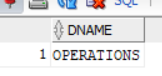
**Task 2.8 Anti Joins**

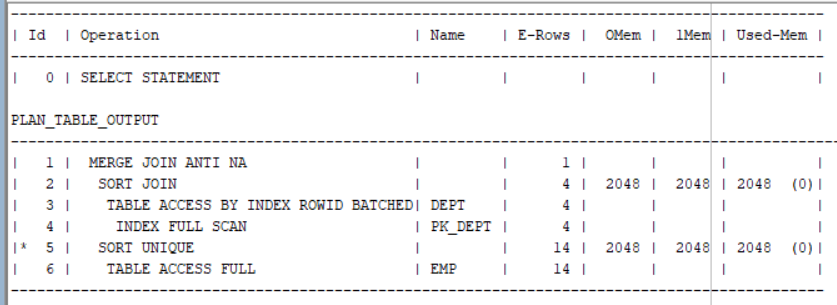
SELECT DName

FROM SCOTT.dept dept

WHERE deptno NOT IN

(SELECT deptno FROM scott.emp );





An antijoin between two tables returns rows from the first table where no matches are found in the second table. Anti-joins are written using the NOT EXISTS or NOT IN constructs. An anti-join is essentially the opposite of a semi-join. To find un-matched data between two tables Anti-Join is better than using not in or not exists.