**Task 4 – Aliaksandr\_Kudzelka**

**Ex1 – AutoTrace Configuration**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Auto Trace Configuration Options** | **Expected Results** | **Description** |
| **1** | autotrace OFF | Disables all autotrace. |  |
| **2** | autotrace ON | Shows the [execution plan](http://www.adp-gmbh.ch/ora/sql/execution_plan.html) as well as [statistics](http://www.adp-gmbh.ch/ora/tuning/statistics.html) of the statement. | This is the default. |
| **3** | autotrace traceonly | Displays the [execution plan](http://www.adp-gmbh.ch/ora/sql/execution_plan.html) and the [statistics](http://www.adp-gmbh.ch/ora/tuning/statistics.html) (as *set autotrace ON* does), but doesn't print a query's result. |  |
| **4** | autotrace ON explain | Displays the [execution plan](http://www.adp-gmbh.ch/ora/sql/execution_plan.html) only. | EXPLAIN shows the query execution path by performing an EXPLAIN PLAN. |
| **5** | autotrace ON statistics | Displays the [statistics](http://www.adp-gmbh.ch/ora/tuning/statistics.html) only. | SQL\*Plus produces a STATISTICS report, a second connection to the database is automatically created. |
| **6** | autotrace ON explain statistics | Displays the [execution plan](http://www.adp-gmbh.ch/ora/sql/execution_plan.html) & the [statistics](http://www.adp-gmbh.ch/ora/tuning/statistics.html) |  |
| **7** | autotrace traceonly explain | Displays the execution plan, but does not print query data. |  |
| **8** | autotrace traceonly statistics | Displays the statistics, but does not print query data. |  |
| **9** | autotrace traceonly explain statistics | Shows the [execution plan](http://www.adp-gmbh.ch/ora/sql/execution_plan.html) as well as [statistics](http://www.adp-gmbh.ch/ora/tuning/statistics.html) of the statement, but does not print query data. |  |
| **10** | autotrace OFF explain | Displays query result. | Autotrace disabled. |
| **11** | autotrace OFF statistics | Displays query result. | Autotrace disabled. |
| **12** | autotrace OFF explain statistics | Displays query result. | Autotrace disabled. |

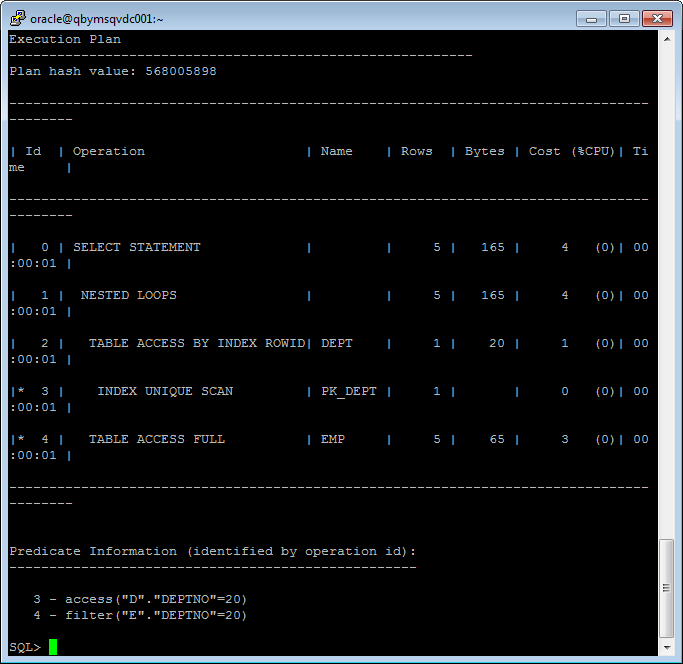
**JOIN methods**

**Ex2 – Nested Loop Joins**

SELECT /\*USE\_NL(dept emp)\*/empno, ename, dname, loc

FROM scott.emp e, scott.dept d

WHERE e.deptno = d.deptno AND d.deptno = 20;

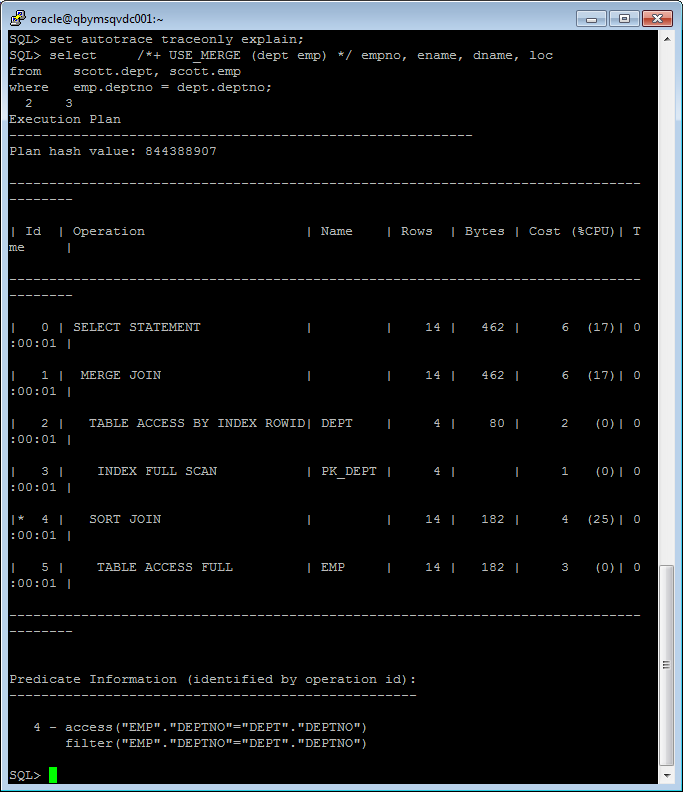


**Ex3 – Sort-Merge Joins**

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

from scott.dept, scott.emp

where emp.deptno = dept.deptno;

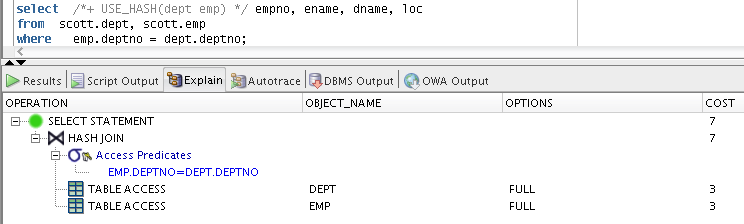


**Ex4 – Hash Joins**

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

from scott.dept, scott.emp

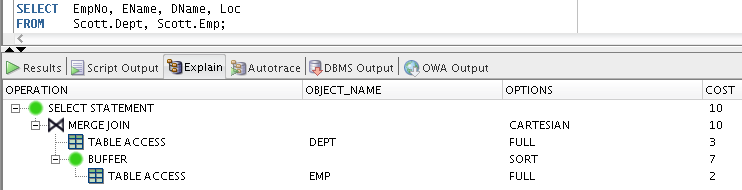
where emp.deptno = dept.deptno;



**Ex5 – Cartesian Join**

SELECT EmpNo, EName, DName, Loc

FROM Scott.Dept, Scott.Emp;

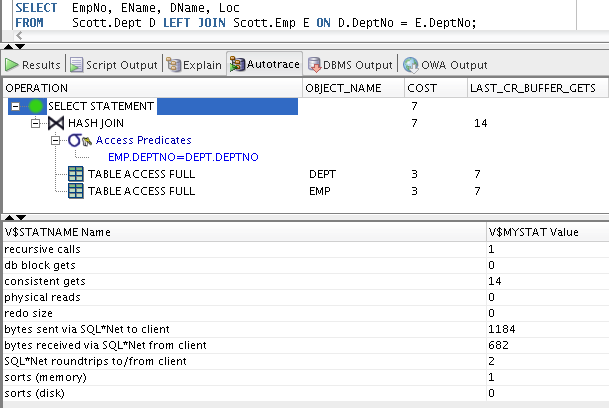
****

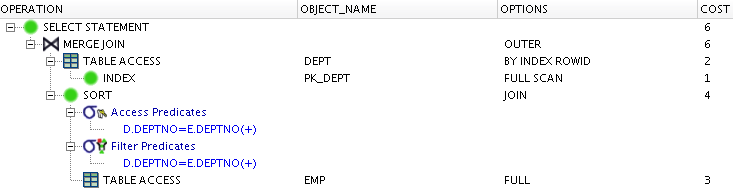
**Ex6 – LEFT/RIGHT Outer Join**

**LEFT Outer Join**

SELECT EmpNo, EName, DName, Loc

FROM Scott.Dept LEFT JOIN Scott.Emp ON Emp.DeptNo = Dept.DeptNo;

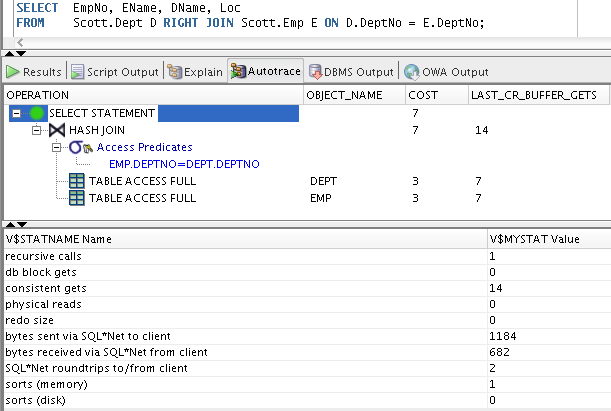


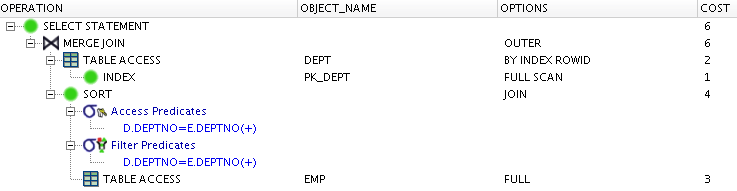


**RIGHT Outer Join**

SELECT EmpNo, EName, DName, Loc

FROM Scott.Dept RIGHT JOIN Scott.Emp ON Emp.DeptNo = Dept.DeptNo;



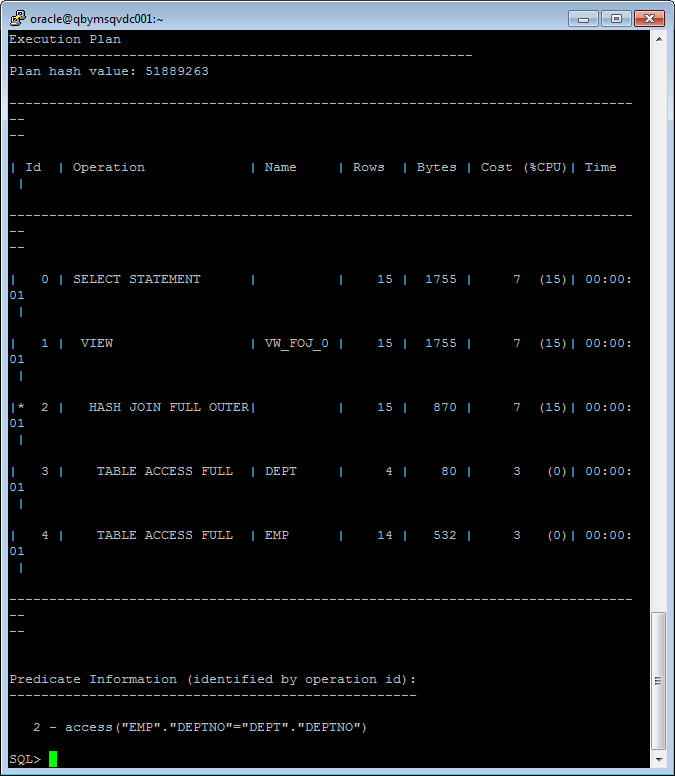


**Ex7 – FULL Outer Join**

SELECT \*

FROM Scott.Emp FULL OUTER JOIN Scott.Dept

ON (Emp.DeptNo = Dept.DeptNo);



**Ex8 – Semi Joins**

SELECT Dept.DName

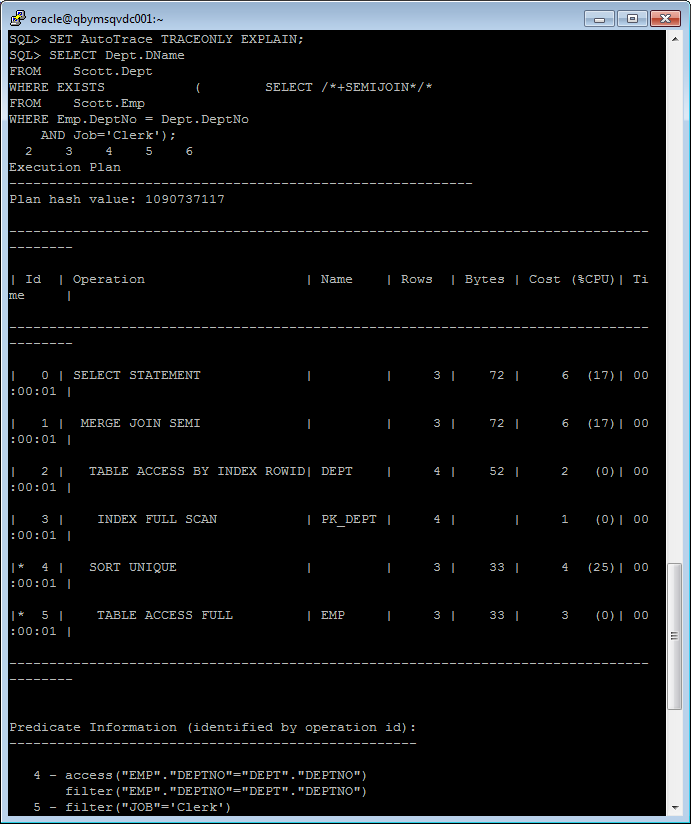
FROM Scott.Dept

WHERE EXISTS ( SELECT /\*+SEMIJOIN\*/\*

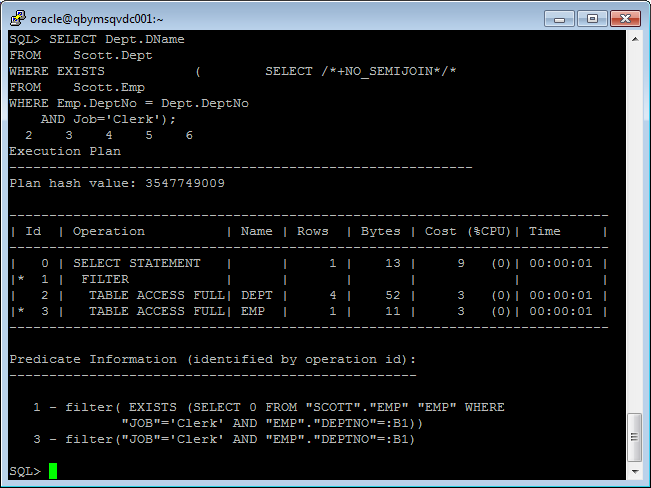
FROM Scott.Emp

WHERE Emp.DeptNo = Dept.DeptNo

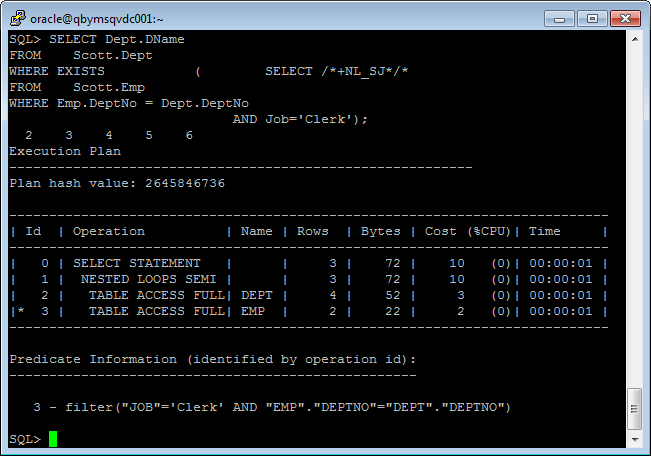
AND Job='Clerk');



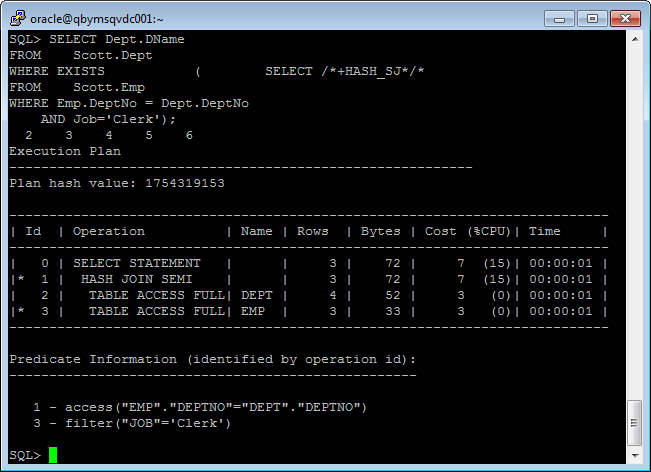
NO\_SEMIJOIN



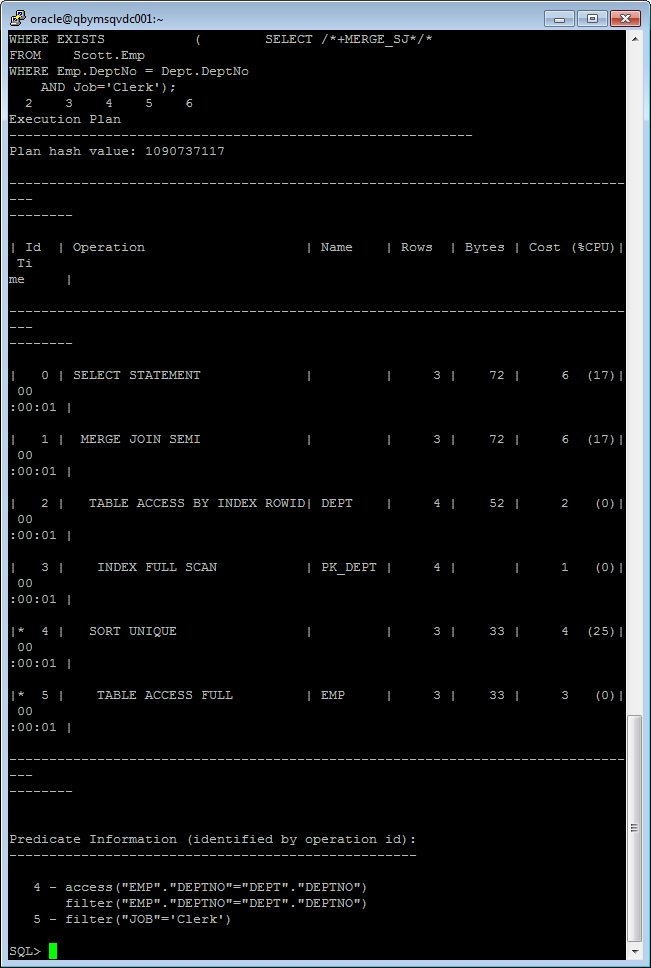
NL\_SJ



HASH\_SJ



MERGE\_SJ



**Ex9 – Anti Joins**

SELECT Dept.DName

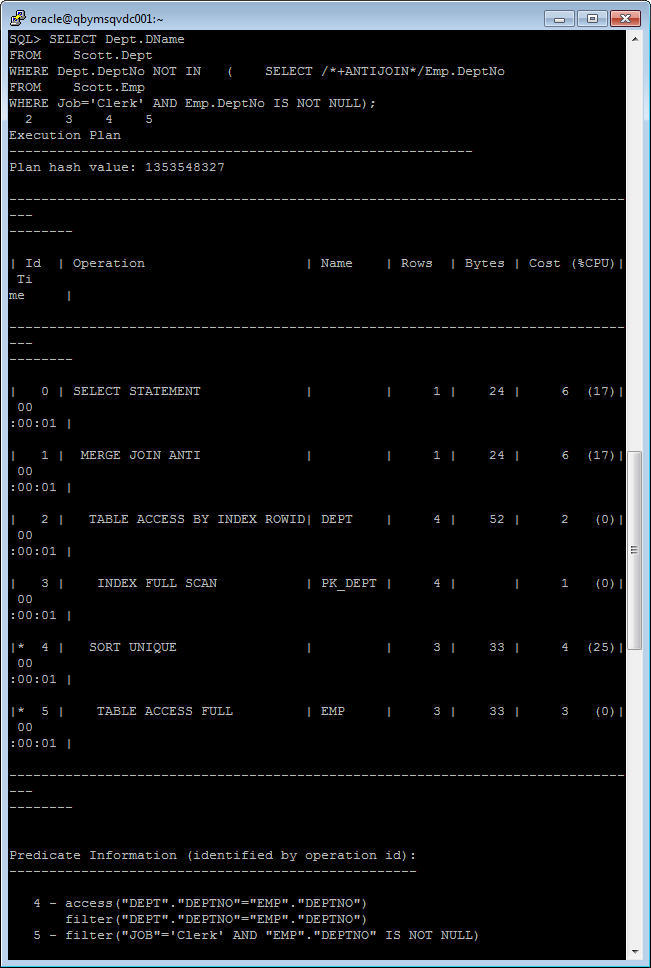
FROM Scott.Dept

WHERE Dept.DeptNo NOT IN

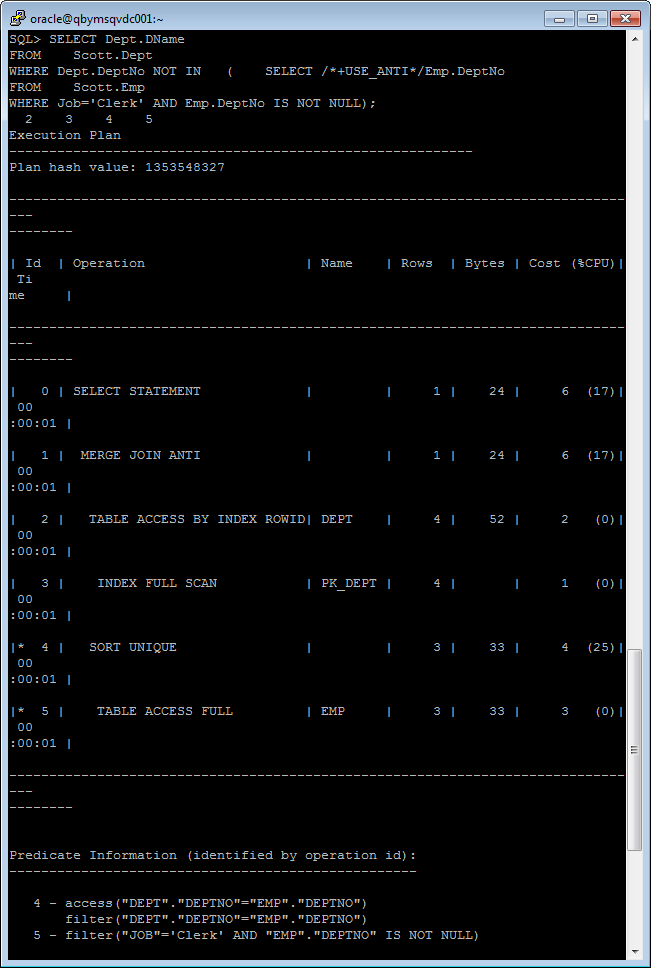
(SELECT /\*+ANTIJOIN\*/Emp.DeptNo

FROM Scott.Emp

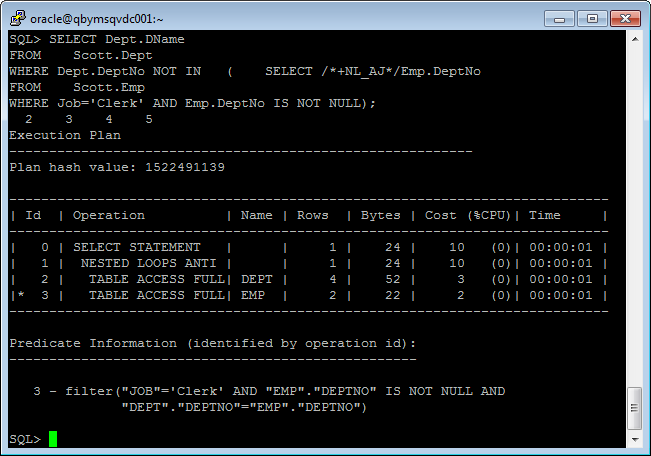
WHERE Job='Clerk' AND Emp.DeptNo IS NOT NULL);



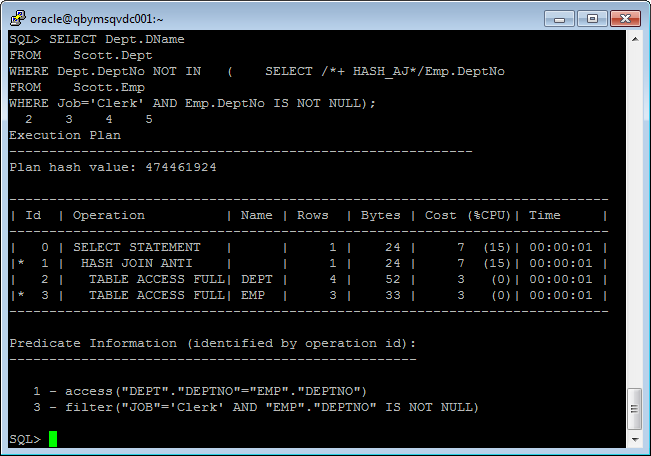
USE\_ANTI



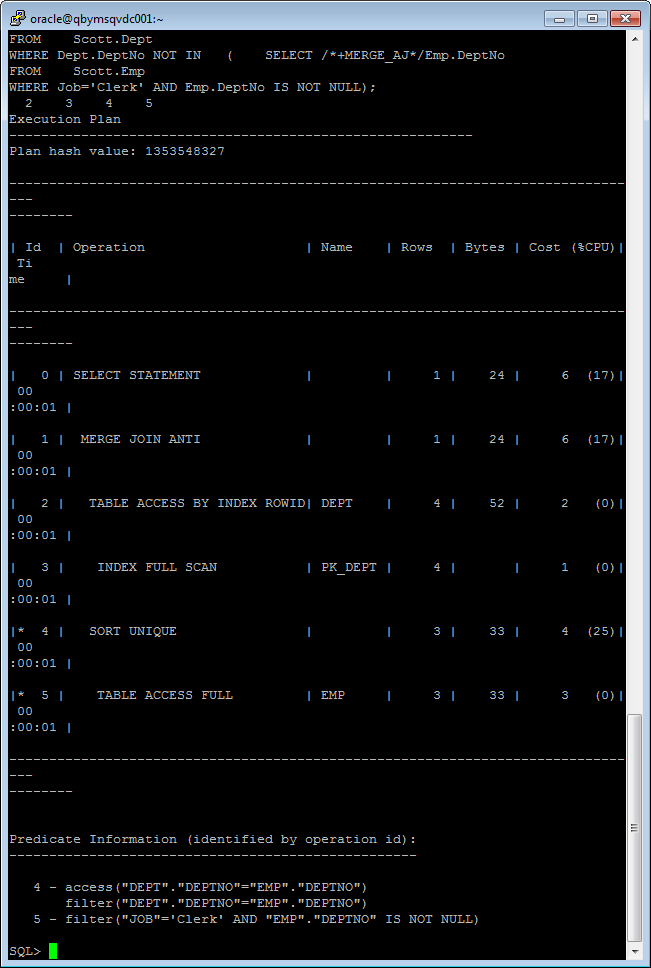
NL\_AJ



HASH\_AJ



MERGE\_AJ



**Ex10 – Summary**

With a **NESTED JOIN**, the optimizer chooses a driving (outer) table and a driven (inner) table. To reduce the number of scans of the inner table, the driving (outer) table is typically a **small table** (or a filter on the driving table reduces it to a small number of rows).

To reduce the cost of repeated scans of the driven (inner) table, the inner table will typically have a **unique** or **selective index** that can be used on the join column(s).

Nested loops starts returning rows very quickly - this is useful in situations where you intend to retrieve only **the first "n" number of rows** from the result or a process can start consuming rows before the entire set is constructed. However, if memory is not in short supply, or if the **tables are large**, or **selective indexes are not available**, this can be a poor join choice. In fact, CBO will probably choose a *hash join instead*.

With an **OUTER JOIN**, Oracle must use the table whose rows are being preserved as the driving table. For each row read from the driving table, it searches the other table (possibly via an index) looking for matches. If a match is found, the joined row is returned. If no match is found, the driving table's row is returned anyway, with NULL for all fields in the other table.

If Oracle were to choose the **SMALL table** as the driving table in this example, then after processing all rows in the SMALL table, how would Oracle know which rows in the BIG table had not been matched so those rows could then be returned? **Oracle must use the BIG table** as the driving table in the join, even though performance suffers. So, an outer join can hurt the performance of a nested loop.

The **HASH JOIN** is a very common join technique, especially on servers with lots of PGA memory dedicated to Oracle.

Unlike the nested loops join, the hash join has a pre-processing step that delays the return of the 1st row. But once it starts returning rows, this is a **very fast join technique**.

Since a hash join can spend considerable time creating a hash table, this join technique is typically useful when **you intend to read the entire result set.** If your intention is **to read just the first row or first 100 rows**, a *nested loops* join is probably a better choice.

The hash join process involves the following steps:

1. Read the driving table (CBO will designate **the smaller table as the driving table**) and for each row read, hash the join key value. The hash value is used as an index into the hash table.

2. Create a private memory-based table (array). Each entry in the array contains the hashed value and the row from the driving table. If this array is too large for the private memory available to the session, Oracle will use temp disk space. This will raise the cost of the hash join and lower the chances that CBO will choose this technique. This pre-processing step causes an initial delay; i.e. the **client will not start receiving rows as fast as a nested loops join would produce them**. However, the cost and elapsed time to return the entire result set is often far less than the nested loops.

A **SORT-MERGE JOIN** can sometimes be best for overall throughput when processing **large sets**. Note that usually the hash join performs better than the sort merge, but is not always an available path (perhaps the join is a range join).

Sort merge joins can - if no index exists on the join key - require significant pre­processing of the data (sorting). Therefore, the invoker will not start receiving rows **until the preprocessing has completed which can require a long time**. Because of this, **the sort-merge join is usually *not* good in an on-line environment**. It is much better suited in a batch environment.

The sort-merge join **can be useful for range joins (e.g. > or <).**

**ANTI JOIN**s **are useful when there is no match in the NOT IN subquery**.

An efficient method of handling an **EXISTS correlated subquery** is the use of the **SEMI-JOIN** join technique. With this join method Oracle probes the inner table looking for a match for the outer row. However, the difference is that once a match is found, a single outer row is returned; even if there might be more matches, the search stops and the single outer row is returned.

**CARTESIAN JOIN**s are often the result of a missing predicate. So we should beware of this situation.