**Report “Lab 5”**

**Anton Tserakhau**

# Auto Trace & Explain Plan

## Task 1: Auto Trace configuration training

Summary table with all result and text description of analyses this results.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Auto Trace Configuration Options** | **Expected Results** | **Description** |
| 1 | set autotrace off | Result of query | Show only table with result of query |
| 2 | set autotrace on | Result of query + execution plane + statistics | Show table with result of query, execution plan with costs of operations, statistics of query |
| 3 | set autotrace traceonly | Execution plane + statistics | Show table with result of query and statistics of query |
| 4 | set autotrace on explain | Result of query + execution plane | Show table with result of query and execution plan with costs of operations |
| 5 | set autotrace on statistics | Result of query + statistics | Show table with result of query and statistics of query |
| 6 | set autotrace on explain statistics | Result of query + execution plane + statistics | Show table with result of query, execution plan with costs of operations, statistics of query |
| 7 | set autotrace traceonly explain | Execution plane | Show only execution plan with costs of operations |
| 8 | set autotrace traceonly statistics | Statistics | Show only statistics of query |
| 9 | set autotrace traceonly explain statistics | Execution plane + statistics | Show execution plan with costs of operations and statistics of query |
| 10 | set autotrace off explain | Result of query | Show only table with result of query |
| 11 | set autotrace off statistics | Result of query | Show only table with result of query |
| 12 | set autotrace off explain statistics | Result of query | Show only table with result of query |

# Join Methods

I have created tables (2 small indexed tables and 2 big indexed tables – 1 000 000 rows on each) and indexes:

Create table emp\_small as

SELECT \*

FROM scott.emp e;

Create index idx\_sm\_emp on emp\_small (empno);

Create table dept\_small as

SELECT \*

FROM scott.dept d;

Create index idx\_sm\_dept on dept\_small (deptno);

Create table emp\_big as

SELECT t1.empno,

t2.ename,

t2.job,

t2.mgr,

t2.hiredate,

t2.sal,

t2.comm,

t2.deptno

FROM (Select ROW\_NUMBER () OVER (order by 1 ) as empno, MOD(ROW\_NUMBER () OVER (order by 1 )-1, 14)+1 id FROM dual CONNECT BY rownum <= 1000000) t1

LEFT JOIN (SELECT e.\*, ROW\_NUMBER () OVER (order by 1 ) as n FROM scott.emp e) t2

ON t1.id=t2.n

order by t1.empno;

Create index idx\_bg\_emp on emp\_big (empno);

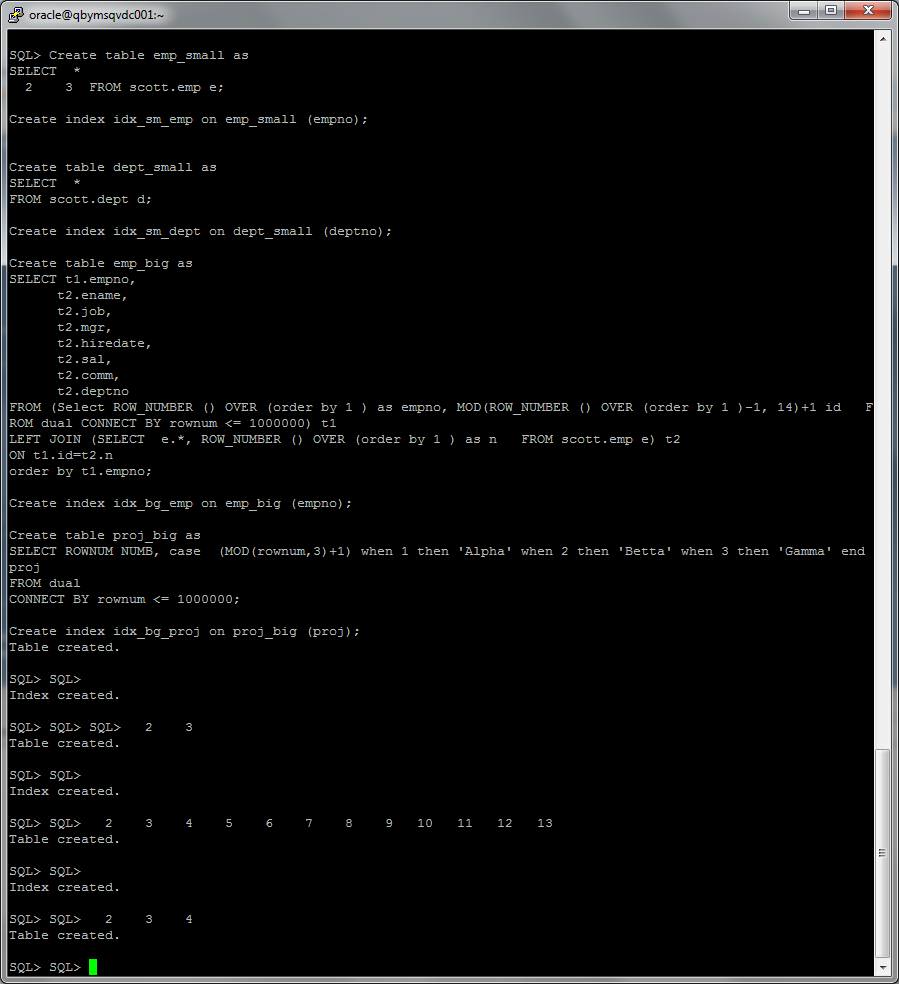
Create table proj\_big as

SELECT ROWNUM NUMB, case (MOD(rownum,3)+1) when 1 then 'Alpha' when 2 then 'Betta' when 3 then 'Gamma' end proj

FROM dual

CONNECT BY rownum <= 1000000;

Create index idx\_bg\_proj on proj\_big (proj);



## 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.

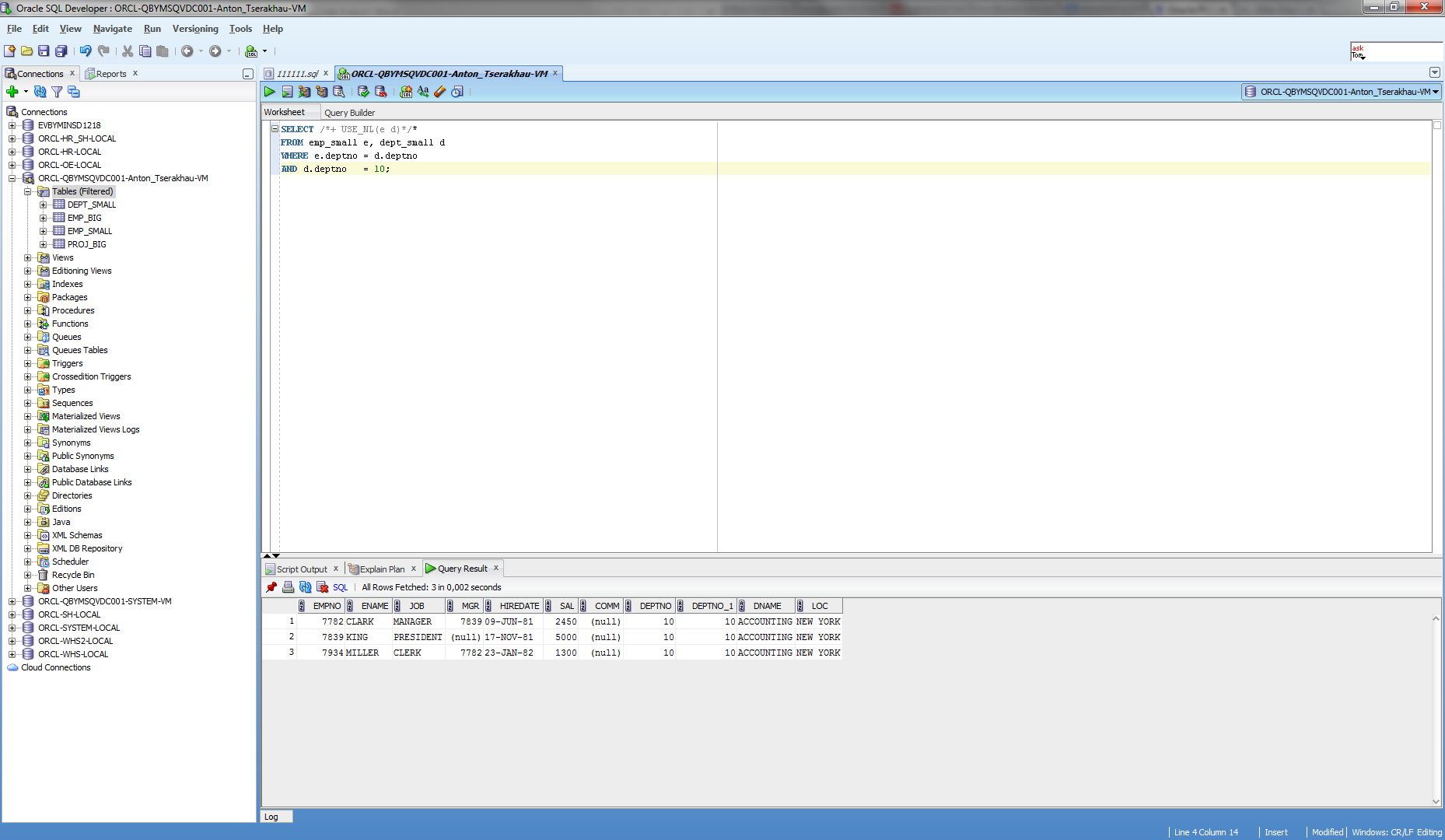
These kinds of joins are quite robust in that they use very little memory. Since row sets are built one row at a time, there is little overhead required. For that reason, they are actually good for huge result sets except for the fact that building a huge result set one row at a time can take quite a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

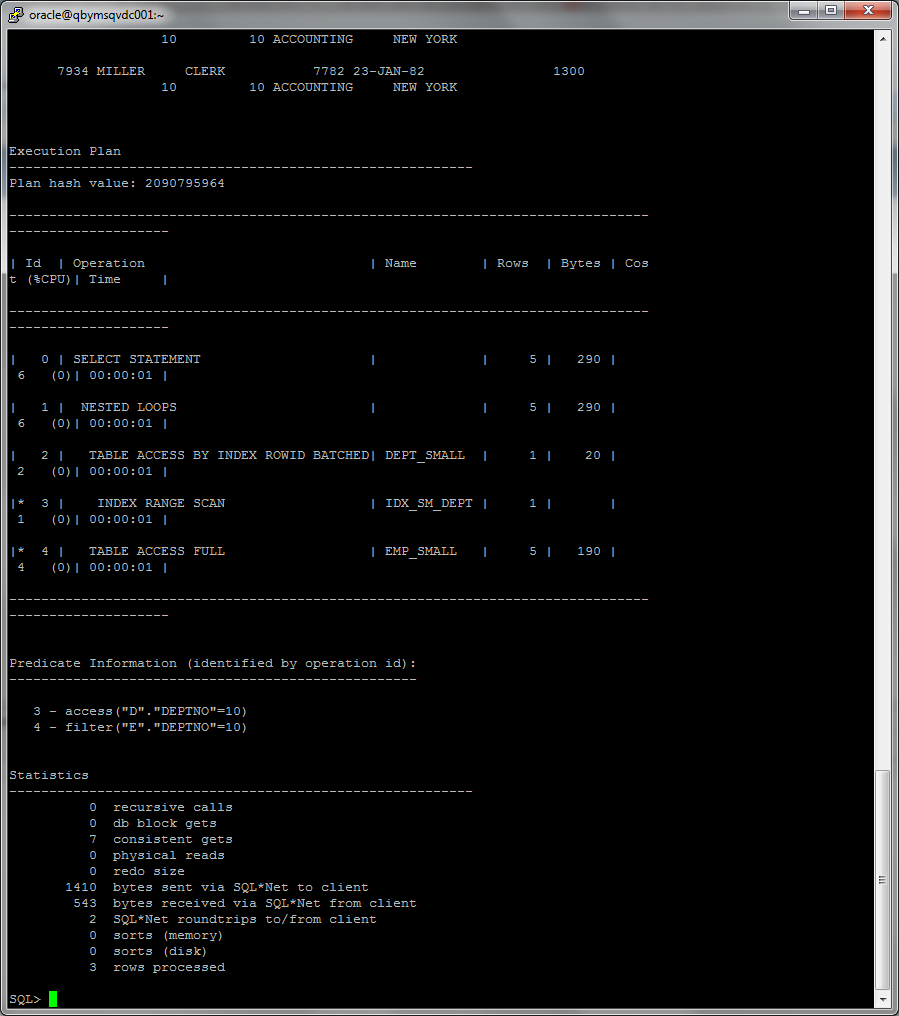
SELECT /\*+ USE\_NL(e d)\*/\*

FROM emp\_small e, dept\_small d

WHERE e.deptno = d.deptno

AND d.deptno = 10;





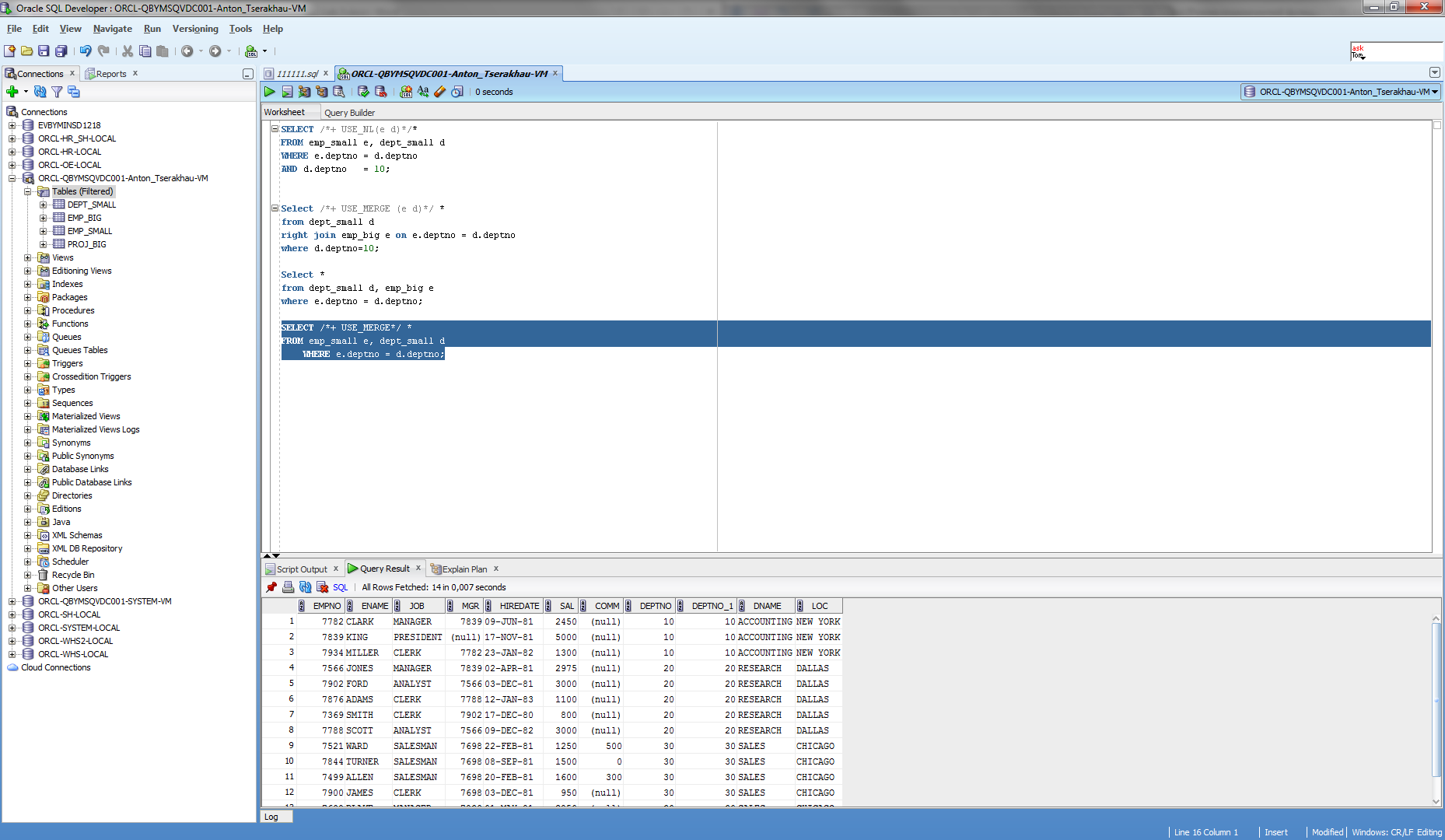
## Task 3: Sort-Merge Joins

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.

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

FROM emp\_small e, dept\_small d

WHERE e.deptno = d.deptno;





## Task 4: Hash Joins

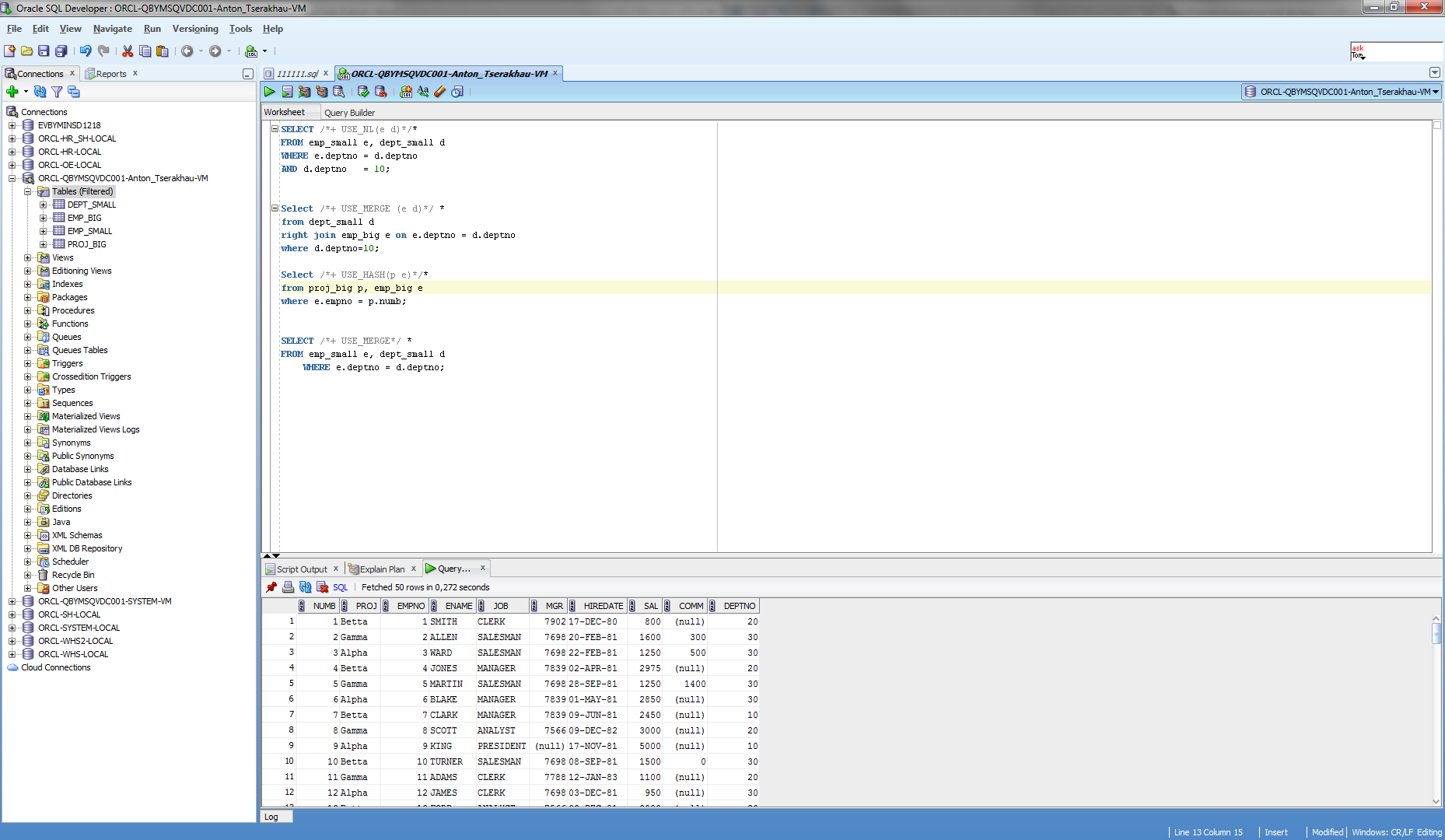
Hash joins, like sort-merge joins, first reads the two tables to be joined independently and applies the criteria in the WHERE clause. Based on table and index statistics, the table that is determined to return the fewest rows will be hashed in its entirety into memory. This hash table includes all the 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.

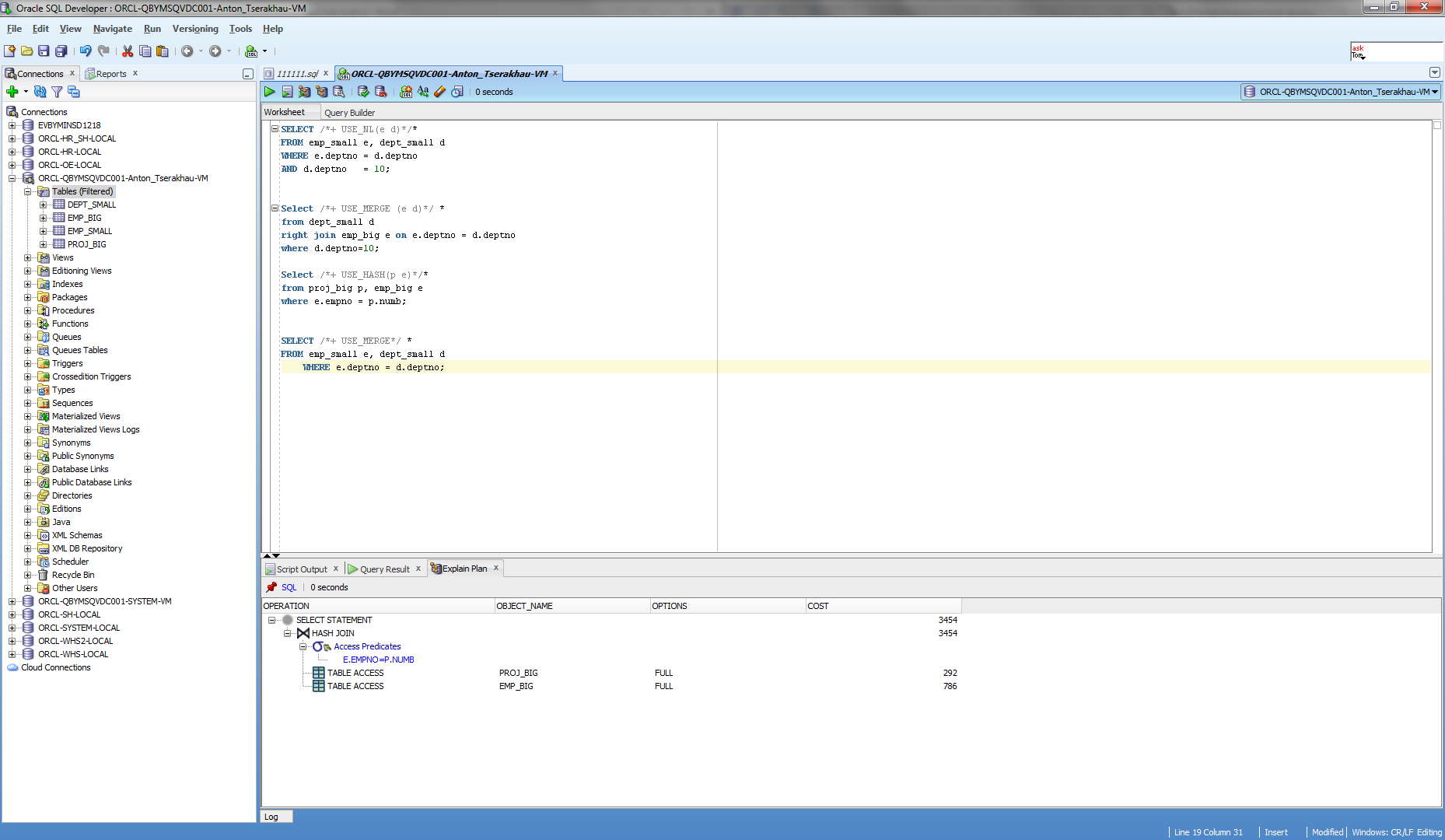
The next step is for the other larger table to be read and the hash function is applied to the join key column. That hash value is then used to probe the smaller in memory hash table for the matching hash bucket where the row data for the first table resides. Each bucket has a list (represented by a 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.

Select /\*+ USE\_HASH(p e)\*/\*

from proj\_big p, emp\_big e

where e.empno = p.numb;



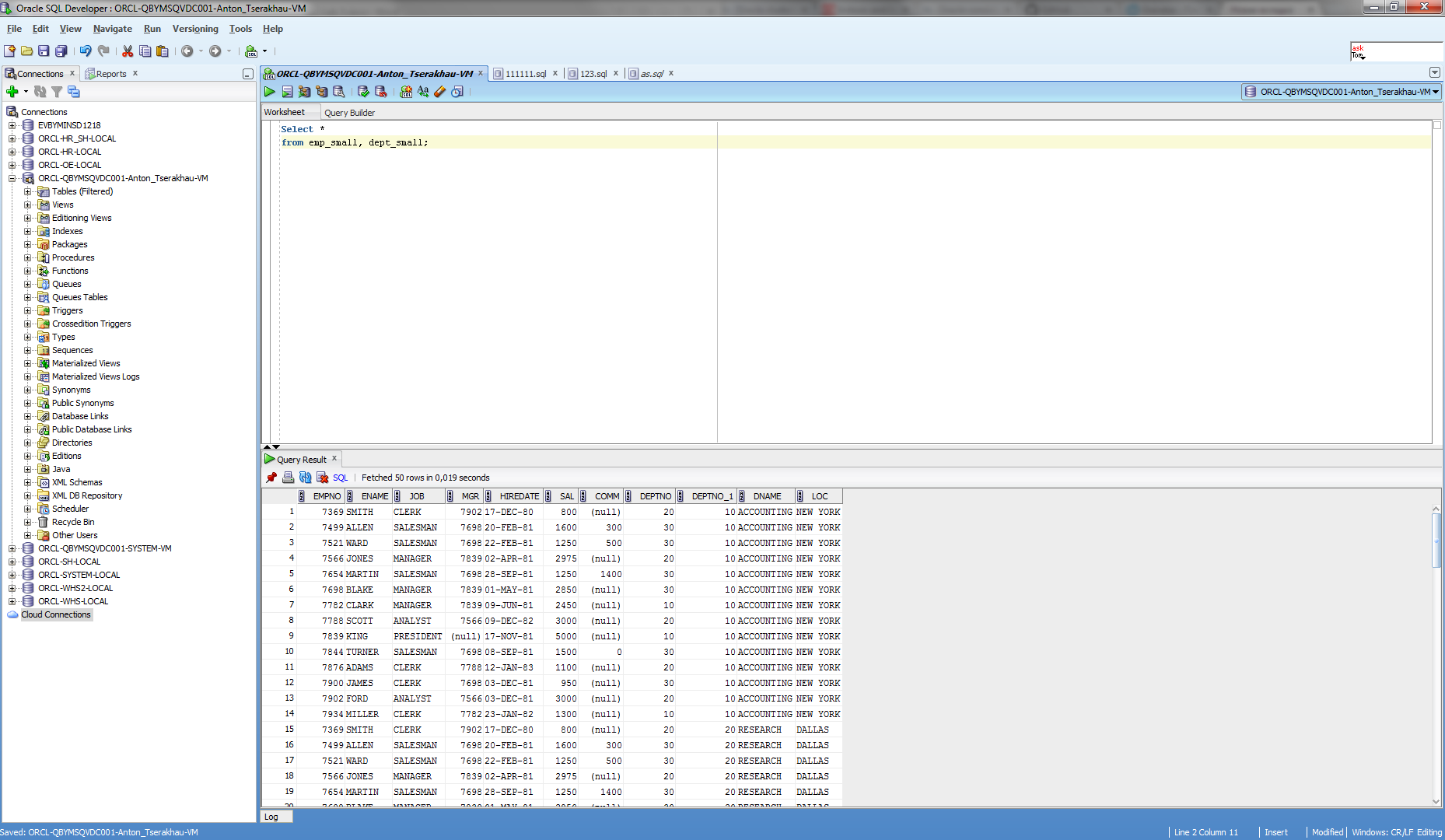


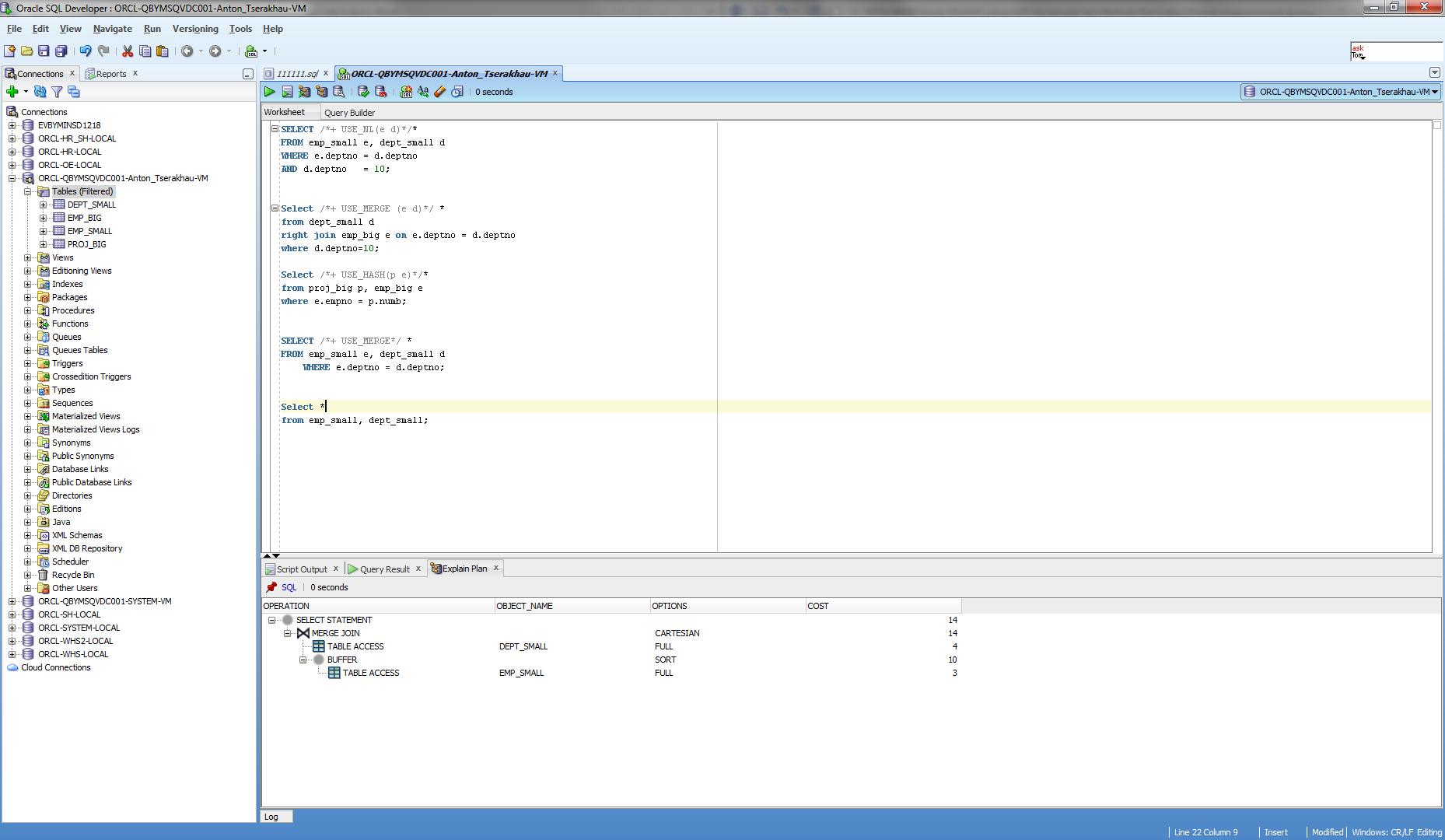
## Task 5: Cartesian Joins

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.

Select \*

from emp\_small, dept\_small;





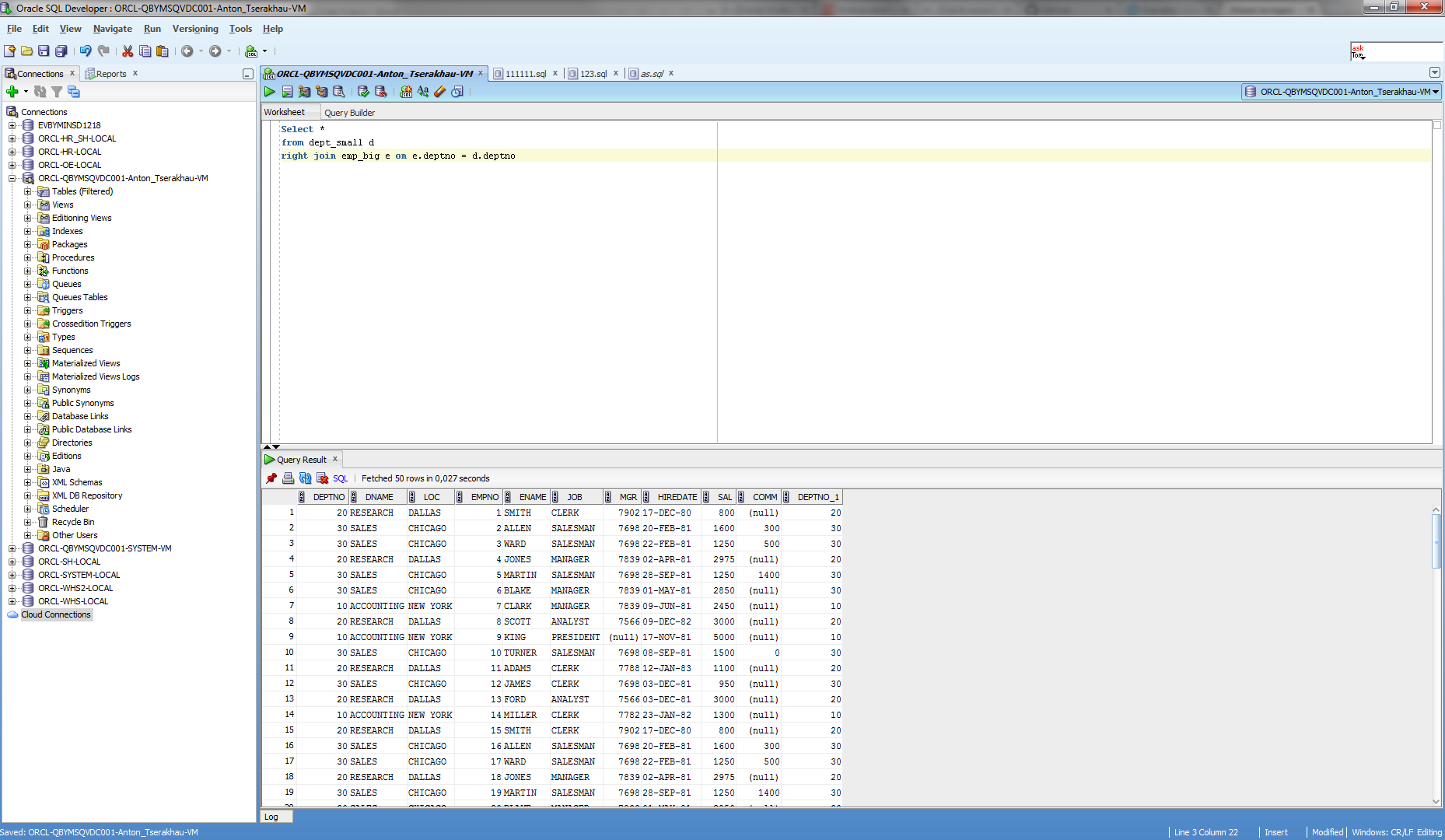
## Task 6: Left/Right Outer Joins

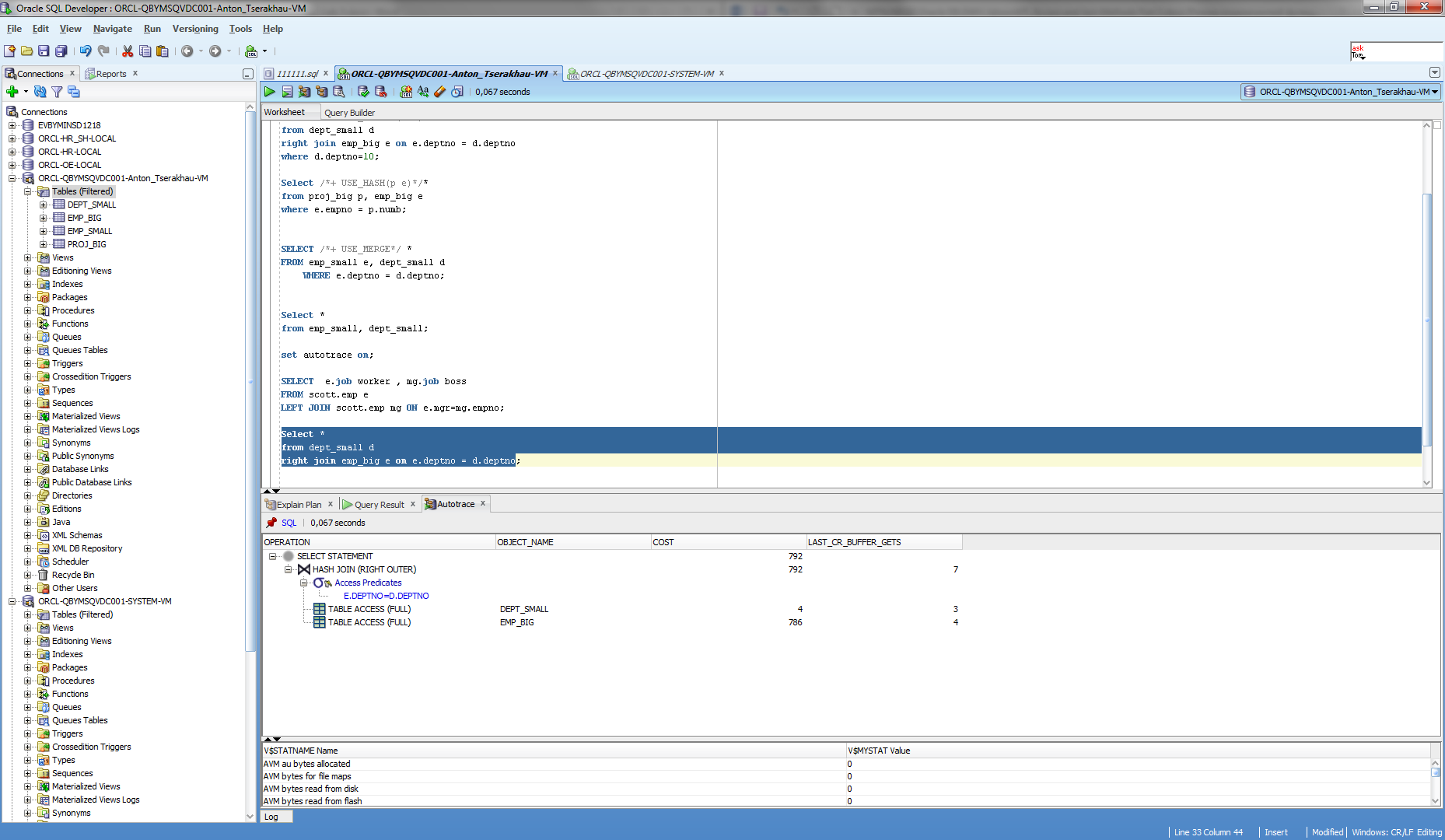
An outer join returns all rows from one table and only those rows from the joined table where the join condition is met.

Select \*

from dept\_small d

right join emp\_big e on e.deptno = d.deptno





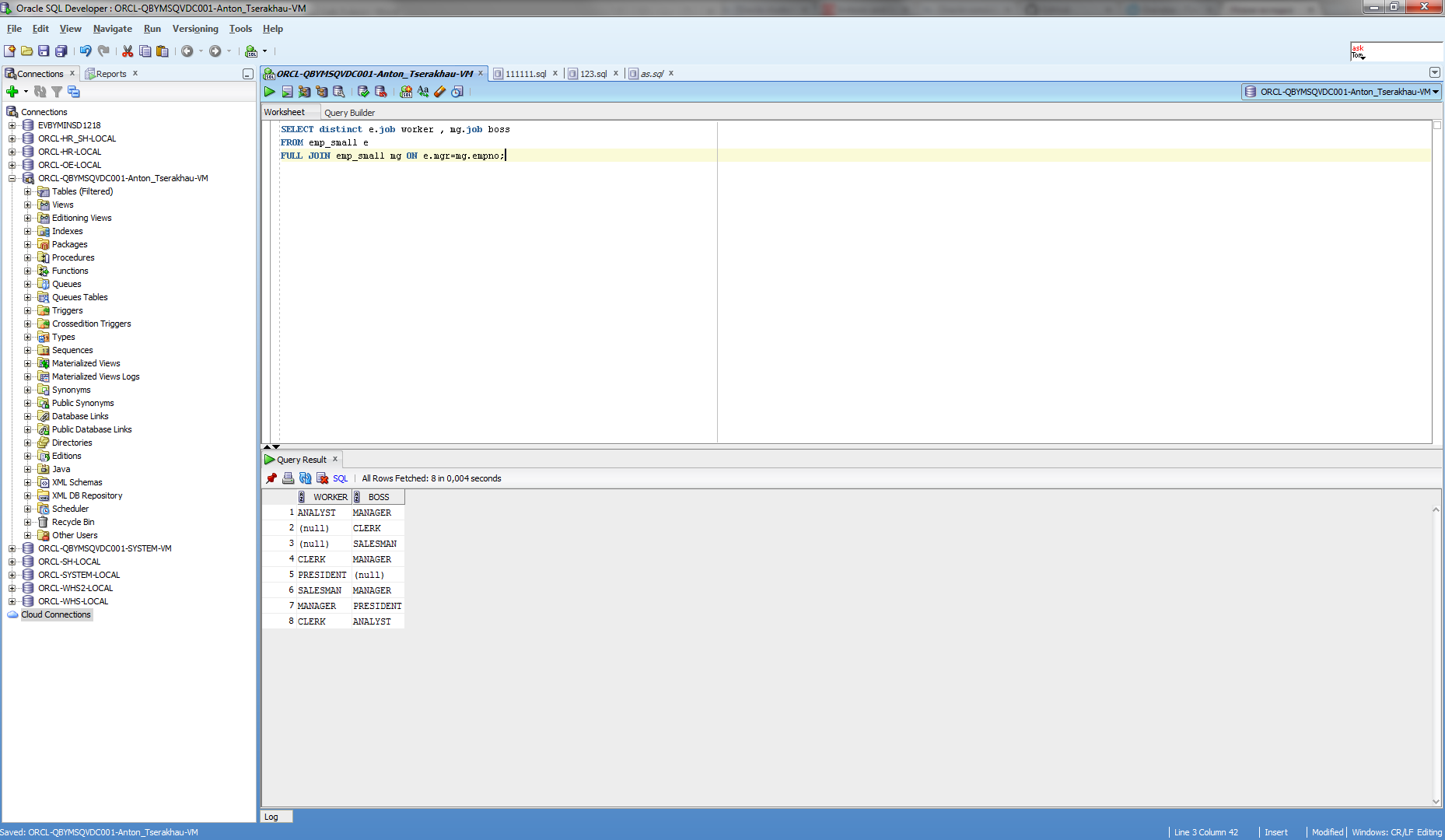
## Task 7: Full Outer Join

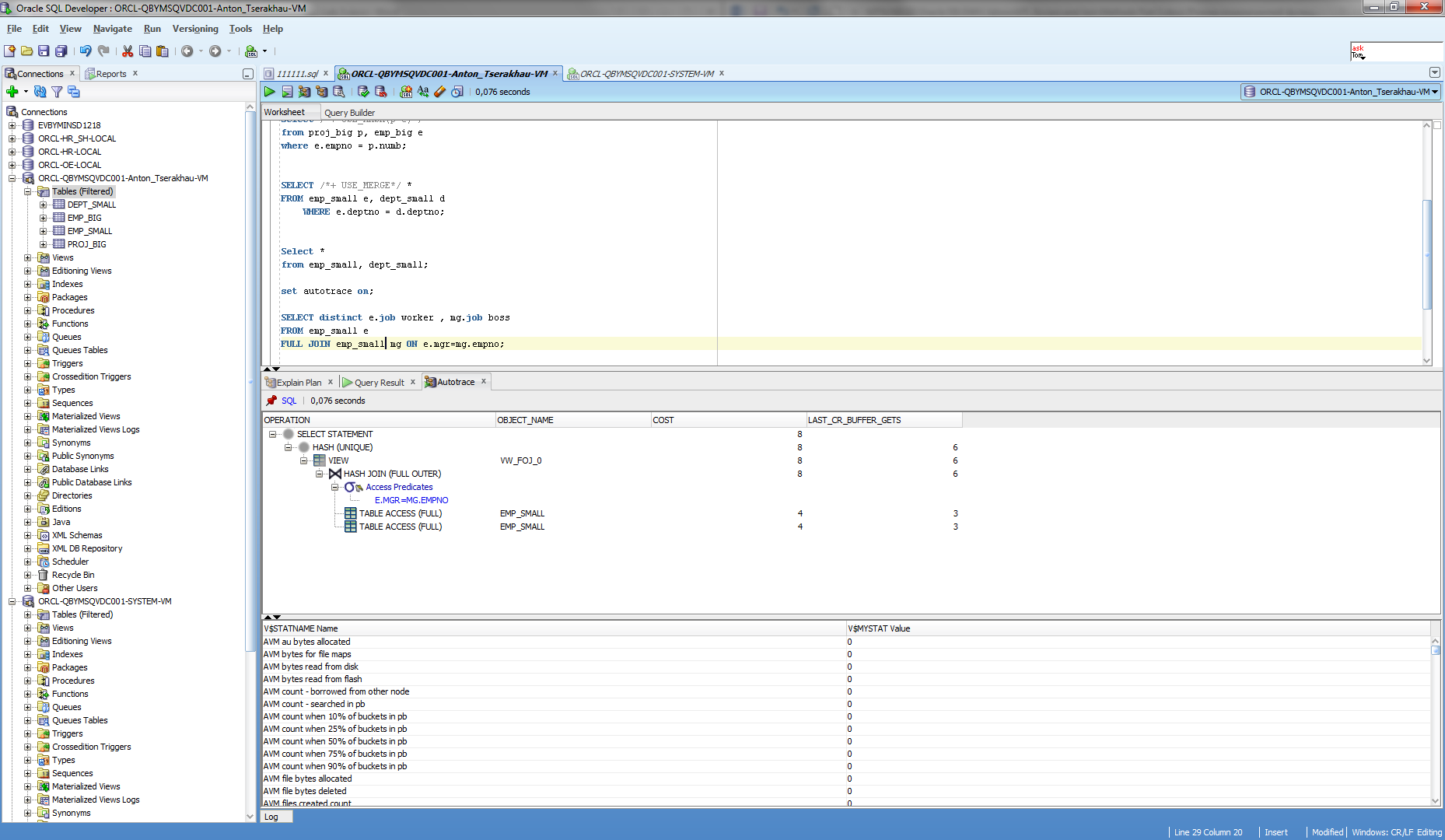
A full outer join will join two tables from left-to-right and right-to-left. Records that join in both directions are output once to avoid duplication. The full outer join will return all the rows from both tables that match plus the rows that are unique to each table.

SELECT distinct e.job worker , mg.job boss

FROM emp\_small e

FULL JOIN emp\_small mg ON e.mgr=mg.empno;





## Task 8: Semi Joins

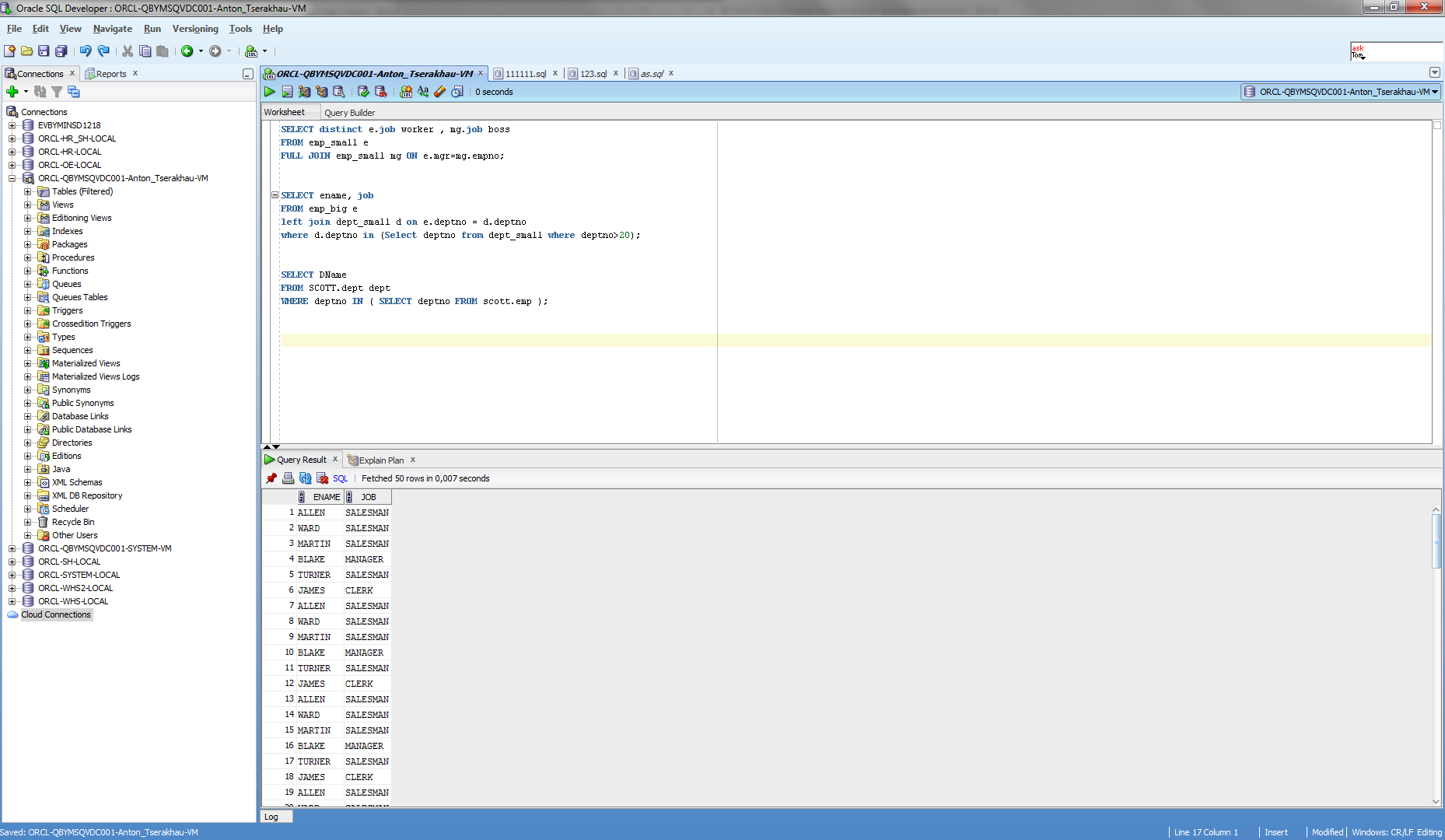
A semi-join is a join between two sets of data (tables) where rows from the first set are returned, based on the presence or absence of at least one matching row in the other set.

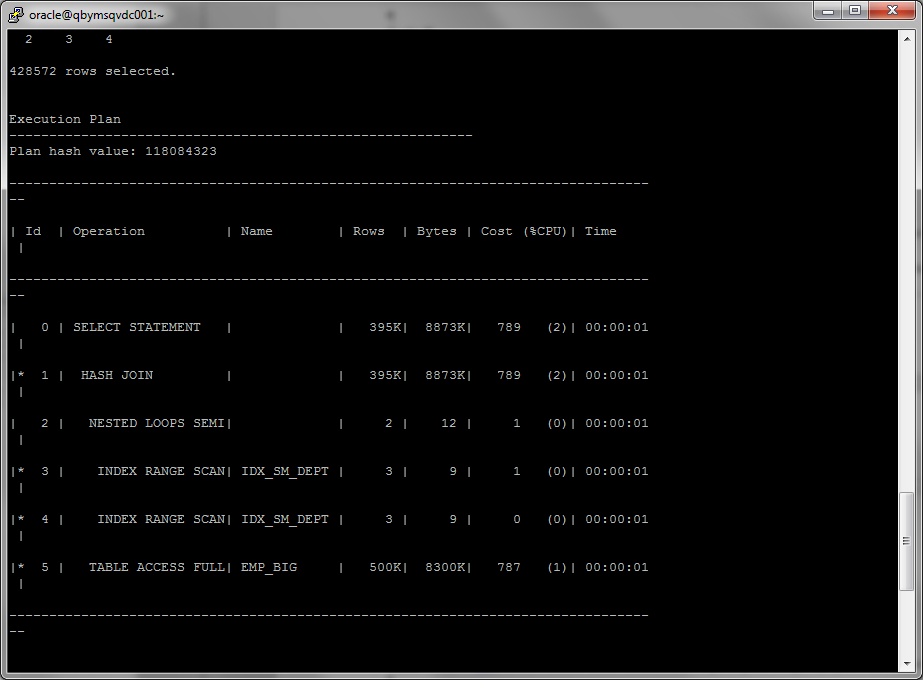
SELECT ename, job

FROM emp\_big e

left join dept\_small d on e.deptno = d.deptno

where d.deptno in (Select deptno from dept\_small where deptno>20);





## Task 9: Anti Joins

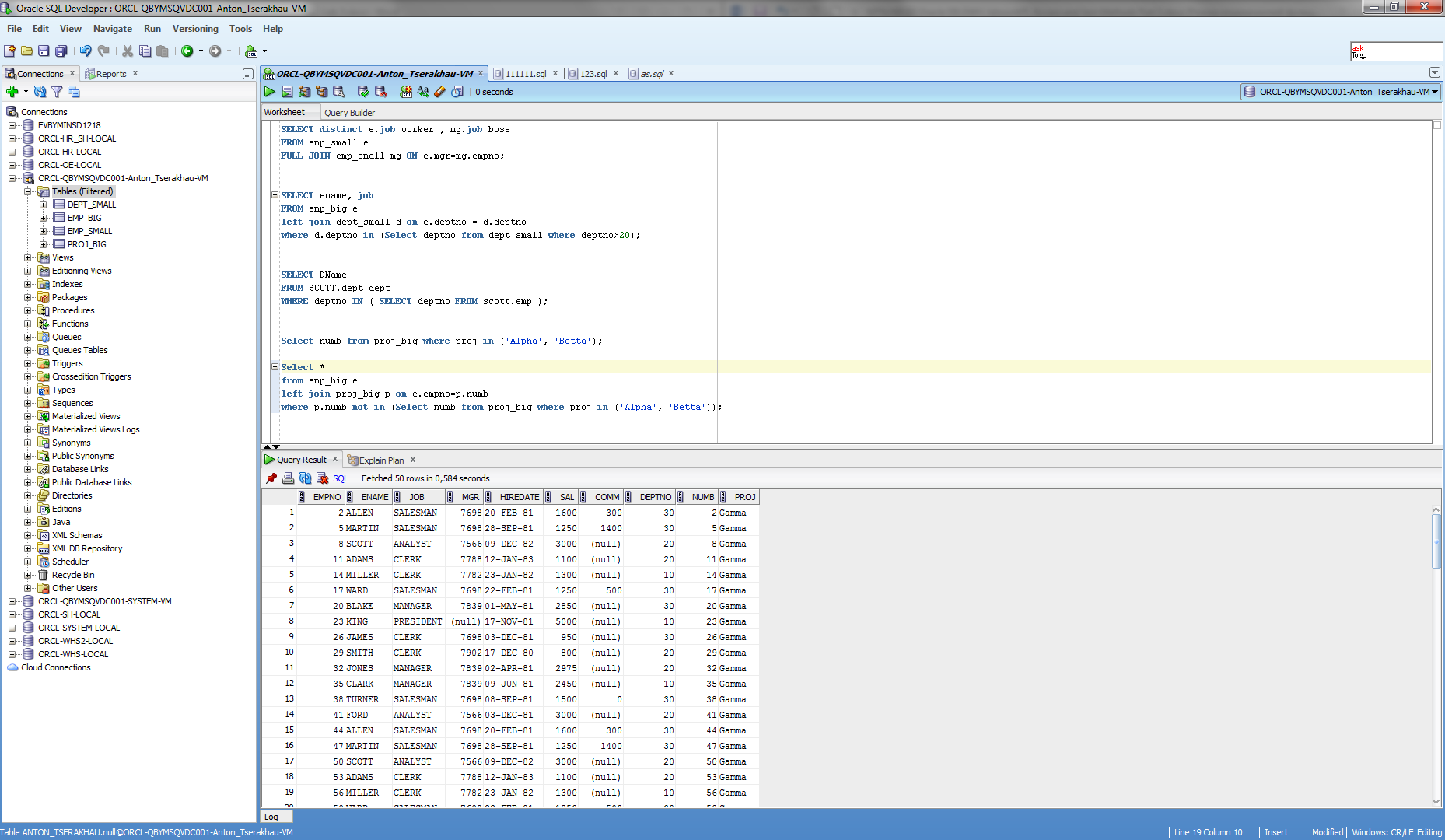
Anti-joins are basically the same as semi-joins in that they are an optimization option that can be applied to nested loop, hash, and merge joins. However, they are the opposite of semi-joins in terms of the data they return. Those mathematician types familiar with relational algebra would say that anti joins can be defined as the complement of semi-joins.

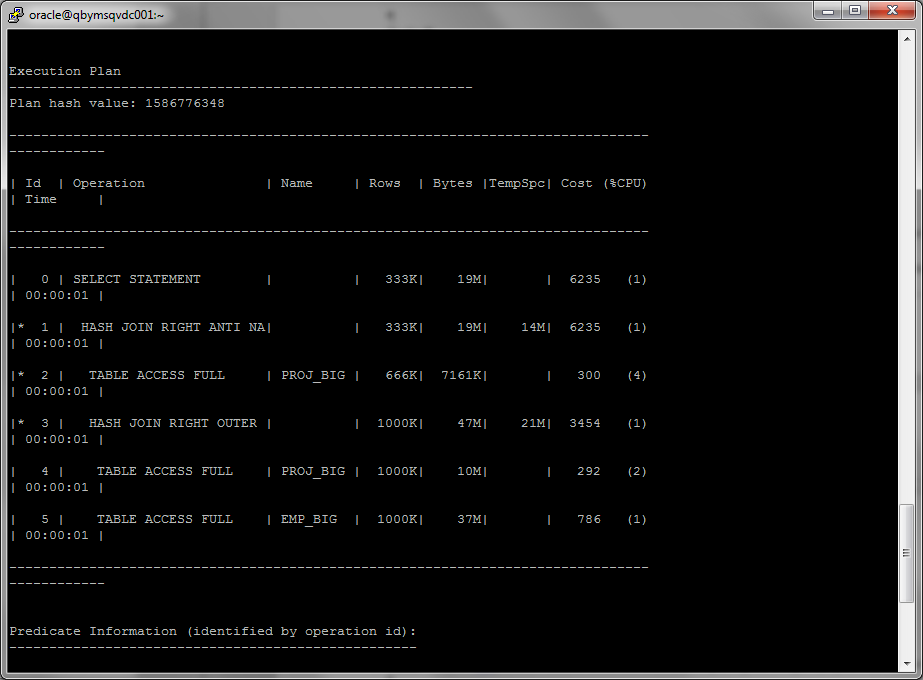
Select \*

from emp\_big e

left join proj\_big p on e.empno=p.numb

where p.numb not in (Select numb from proj\_big where proj in ('Alpha', 'Betta'));





## Task 10: Prepare summary table

Comparison of all possible variant of join methods and join access methods:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Join Access “A” | Join Access “B” | Nested Loop | Hash Join | Sort-Merge Join | Anti-Join | Semi-Join |
| Small Table | Small Table | 8 | 9 | 8 | 6 | 7 |
| Small Table | Indexed Small Table | 10 | 9 | 9 | 7 | 7 |
| Indexed Small Table | Indexed Small Table | 10 | 10 | 10 | 8 | 8 |
| Big Table | Big Table | 1 | 6 | 5 | 5 | 4 |
| Big Table | Indexed Big Table | 3 | 7 | 5 | 5 | 4 |
| Indexed Big Table | Indexed Big Table | 4 | 8 | 6 | 6 | 5 |
| Small Table | Big Table | 3 | 9 | 7 | 5 | 5 |
| Small Table | Indexed Big Table | 8 | 9 | 8 | 6 | 5 |
| Indexed Small Table | Big Table | 2 | 10 | 5 | 5 | 5 |
| Indexed Small Table | Indexed Big Table | 8 | 10 | 6 | 7 | 6 |