SQL Aint So Simple – Part 2

In Part 1 of this topic I proposed that while SQL is not an overly complex language in theory the SELECT command syntax permits numerous equivalent ways to write queries. Therefore the SQL language really is not as simple as we generally believe.

Here's the simple query I utilized as the starting point for discussion:

```
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a, movierental b, rentalitem c, moviecopy d
where a.EMPLOYEEID = b.EMPLOYEEID
  and b.RENTALID = c.RENTALID
  and c.MOVIECOPYID = d.MOVIECOPYID
  and a.LASTNAME = 'Smith'
  and b.CUSTOMERID IN (
          select customerid
          from customer e
          where e.address like 'A%' or
                e.address like 'F%' or
                e.address like 'T%' or
                e.address like 'Z%'
          )
   group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

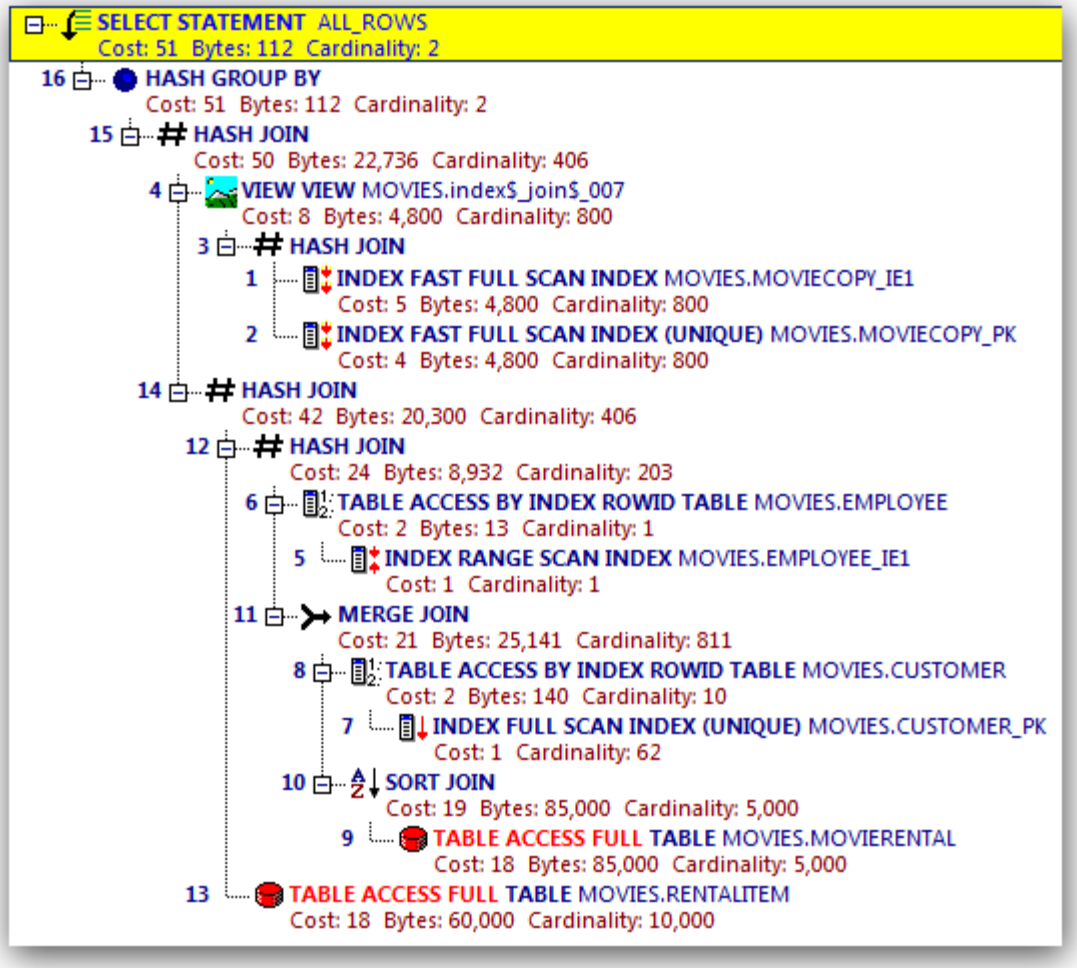
In Part 1 I showed four additional distinct yet equivalent ways to code the above culminating in the ideal solution (i.e. using all the latest and greatest syntax features) shown here:

```
with f as ( select customerid
             from customer e
              where e.address like 'A%' or
                    e.address like 'F%' or
                    e.address like 'T%' or
                    e.address like 'Z%'
          )
select  a.LASTNAME, a.FIRSTNAME, b.RENTALDATE,
        d.COPYFORMAT, sum(b.TOTALCHARGE) REVENUE
from employee a JOIN movierental b using(EMPLOYEEID)
                JOIN rentalitem c using (RENTALID)
                JOIN moviecopy d using(MOVIECOPYID),
```

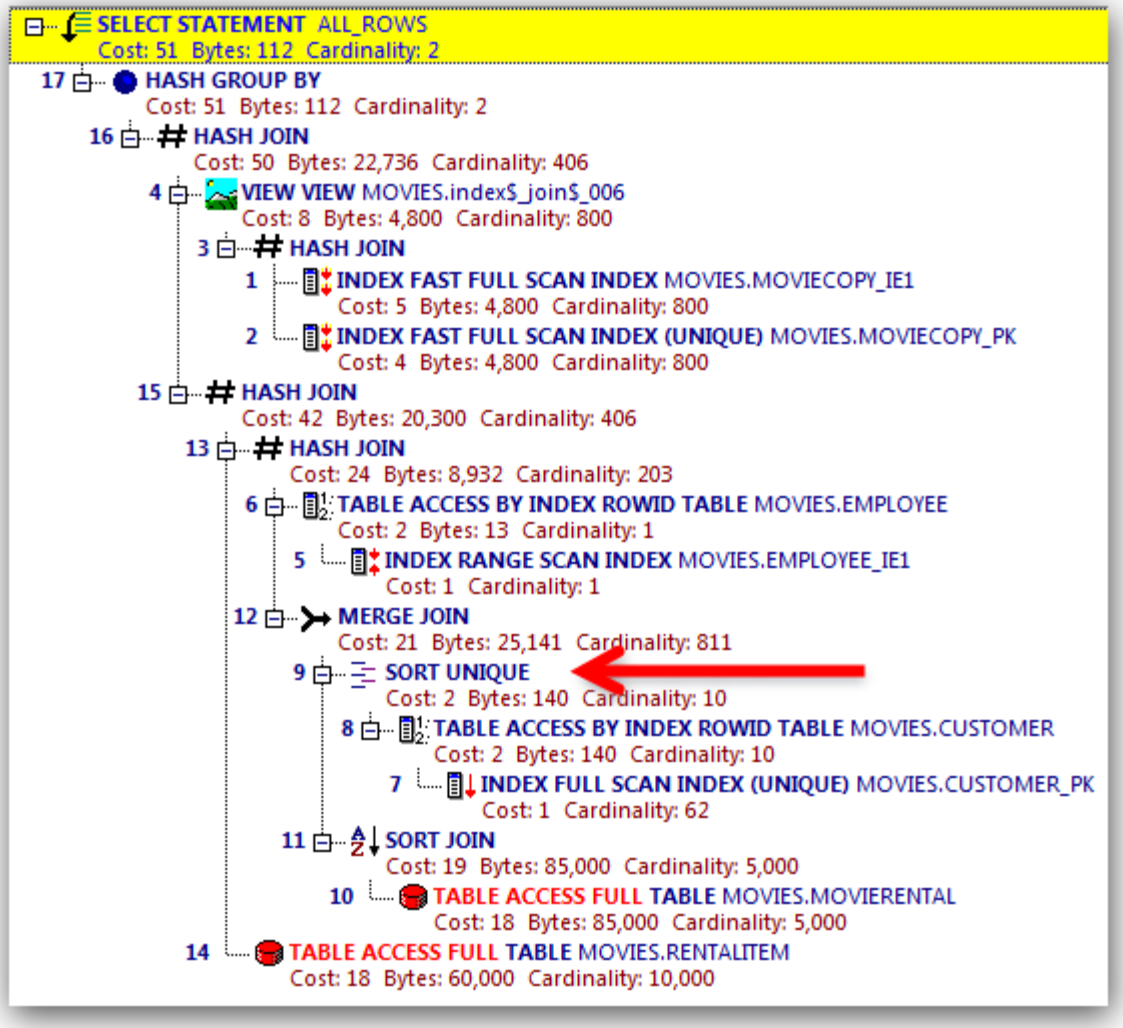
```
                    f
where a.LASTNAME = 'Smith'
  and b.CUSTOMERID = f.CUSTOMERID

   group by a.LASTNAME, a.FIRSTNAME, b.RENTALDATE, d.COPYFORMAT
```

The key point is that the SELECT syntax supports multiple ways to accomplish the same results. Add in that SQL is a set oriented language and that various Oracle versions' optimizers work differently, and the true number of combinations is even greater. Thus the first way one codes a SELECT might capture the correct solution concept (effectiveness) but might require additional thought for what will yield the fastest speed (efficiency).

For example using Oracle 11g R2 with current statiscs gathered on both the tables and indexes the explain for query styles #1, #2, #4 and #5 is:

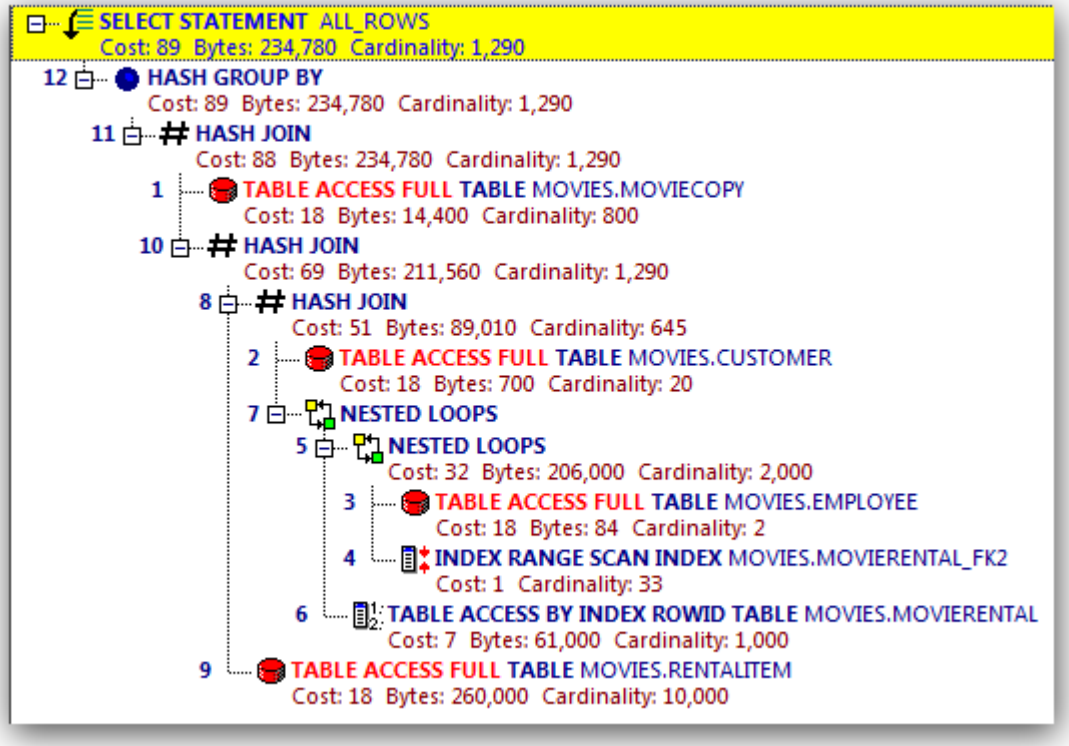


Whereas the explain for query style #3 is almost the same – with just the addition of an extra sort unique added by the EXISTS clause (and highlighted by a red arrow):



For the test data loaded the run times are identical. So even though query style #3 has an extra step, all five coding styles essentially result in the exact same performance in this case. But that might not always be true. It may well happen that the coding style best leveraging newer features could result in a more efficient explain plan and better run time. You'll need to verify the results for yourself. As President Reagan used to say "Trust but verify", in this case meaning trust the Oracle optimizer to do its job but verify that we provide it the best query code to optimize.

Of course when you don't have current statistics collected, all five query styles result in a highly inefficient explain plan as shown below (note all the red highlighted FULL TABLE SCANS). So always make sure you have current statistics gathered for tables and indexes.


SELECT STATEMENT ALL_ROWS
Cost: 89 Bytes: 234,780 Cardinality: 1,290
12 HASH GROUP BY
Cost: 89 Bytes: 234,780 Cardinality: 1,290
11 HASH JOIN
Cost: 88 Bytes: 234,780 Cardinality: 1,290
1 TABLE ACCESS FULL TABLE MOVIES.MOVIECOPY
Cost: 18 Bytes: 14,400 Cardinality: 800
10 HASH JOIN
Cost: 69 Bytes: 211,560 Cardinality: 1,290
8 HASH JOIN
Cost: 51 Bytes: 89,010 Cardinality: 645
2 TABLE ACCESS FULL TABLE MOVIES.CUSTOMER
Cost: 18 Bytes: 700 Cardinality: 20
7 NESTED LOOPS
5 NESTED LOOPS
Cost: 32 Bytes: 206,000 Cardinality: 2,000
3 TABLE ACCESS FULL TABLE MOVIES.EMPLOYEE
Cost: 18 Bytes: 84 Cardinality: 2
4 INDEX RANGE SCAN INDEX MOVIES.MOVIERENTAL_FK2
Cost: 1 Cardinality: 33
6 TABLE ACCESS BY INDEX ROWID TABLE MOVIES.MOVIERENTAL
Cost: 7 Bytes: 61,000 Cardinality: 1,000
9 TABLE ACCESS FULL TABLE MOVIES.RENTALITEM
Cost: 18 Bytes: 260,000 Cardinality: 10,000