Star Join Query Processing
(using Cartesian Product)

1. Introduction

A Star Join query is a query joining a Fact table with more than one dimension tables. If table *F* is the Fact, and tables D_1, D_2, …, D_n, are the dimension tables, a Star Join is to join *F* with D_1, and then the results are joined with D_2, and then D_3, etc.

We will use the USELOG case study to explain Star Join queries. The following are the tables of the fact and the four dimensions in the USELOG case study:

```
SQL> desc usefact;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 SEMID                                              VARCHAR2(10)
 TIMEID                                             NUMBER
 CLASS_ID                                           VARCHAR2(6)
 MAJOR_CODE                                         VARCHAR2(8)
 TOTAL_USAGE                                        NUMBER

SQL> desc semester;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 SEMID                                              VARCHAR2(10)
 SEM_DESC                                           VARCHAR2(20)
 BEGIN_DATE                                         DATE
 END_DATE                                           DATE

SQL> desc labtime;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 TIMEID                                             NUMBER
 TIME_DESC                                          VARCHAR2(15)
 BEGIN_TIME                                         DATE
 END_TIME                                           DATE

SQL> desc majordim;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 MAJOR_CODE                                         VARCHAR2(8)
 MAJOR_NAME                                         CHAR(35)

SQL> desc classdim;
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 CLASS_ID                                           VARCHAR2(6)
 CLASS_DESCRIPTION                                  CHAR(50)
```

2. A Join Query between a Fact and TWO Dimensions

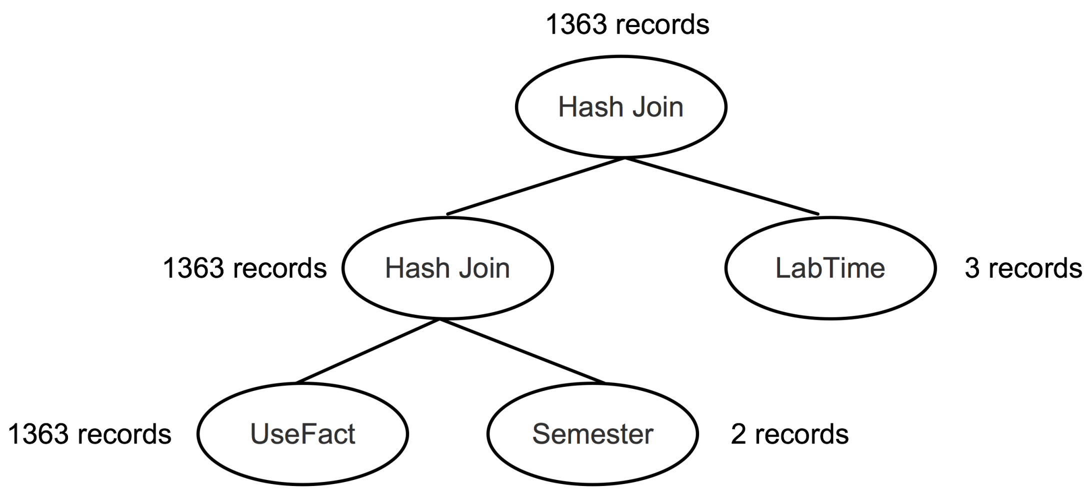
The following query joins the fact (usefact table) and two dimension tables (tables semester and labtime). Because the common attributes for the join (e.g. semid and timeid) are located in the usefact table, the usefact table must be join with the two dimensions. It is impossible to join the two dimensions by themselves because there is no common attribute between semester and labtime dimension tables. Everything has to go through the usefact table.

```
SQL> select *
  2  from usefact u, semester s, labtime l
  3  where u.semid = s.semid
  4  and u.timeid = l.timeid;
```

When the query is executed, the UseFact table needs to be joined with Semester and LabTime tables. There is no particular order, which join needs to be executed first. So, it can be UseFact to be joined with Semester, and then the result is joined with LabTime. Or it is also possible that the UseFact to be joined with LabTime first, and then with Semester. Both ways are correct and will produce the same results.

The query tree below shows the execution of the query whereby UseFact joins with Semester first, and then with LabTime. The numbers of records of the three input tables are: 1363 records in UseFact, 2 records in Semester, and 3 records in LabTime.

When UseFact joins with Semester, the result is 1363 records. This is then joined with 3 LabTime records, and also produces 1363 records, which is the final result of the query.



There are several lessons from this example:

1. Because the records in UseFact are not filtered, all records will need to be joined with the dimensions one-by-one. As a result, the 1363 records will be carried through out the process. The 1363 records are the result of the first join with table Semester, and then 1363 records are joined with the second dimension table LabTime, which will also produce 1363 records.
2. If there are more dimensions used in the query, these 1363 records will be joined with the third dimension table, and then with the fourth dimension table, etc. Consequently, all UseFact records are being used repeatedly in the query tree. It is like carrying a big baggage in the entire query processing.

It would be a good idea, if we could "join" the small dimension tables first, before joining with the fact table. For example, join Semester and LabTime first, before joining with UseFact table, with an aim that we will not carry the heavy UseFact table in the entire query processing.

But the problem is that Semester and LabTime do not have a common join attribute. Therefore, it is impossible to "join" Semester and LabTime table. However though, we could do a "**Cartesian Product**" between Semester and LabTime. Cartesian Product has never been used as a sole operator in query processing because Cartesian Product will find all records combination from the two tables, and as a result, the result of this Cartesian Product will not have any meaning. But Cartesian Product could be used meaningfully in **Star Join** queries.

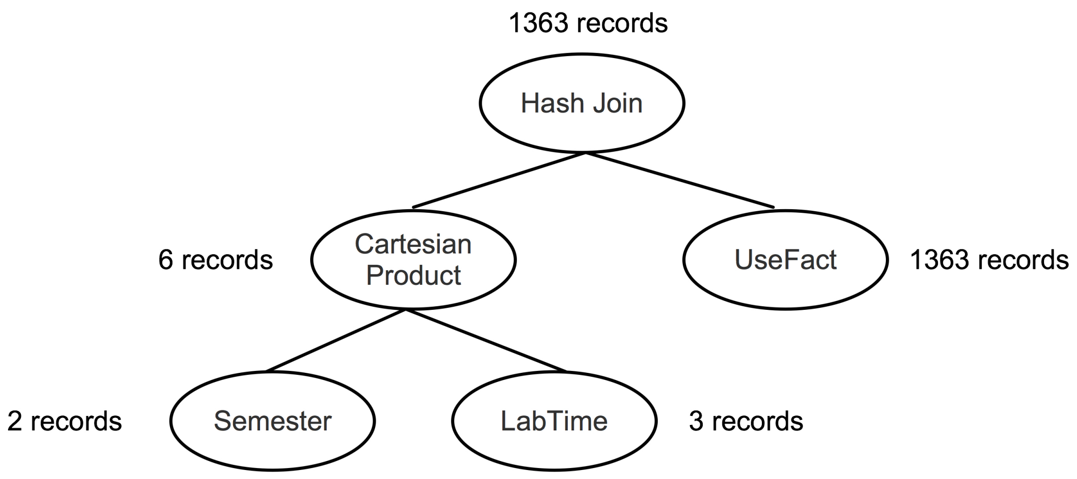
If we do a Cartesian Product between Semester and LabTime, here are the results:

```
SQL> select * from semester;

SEMID      SEM_DESC             BEGIN_DAT END_DATE
---------- -------------------- --------- ---------
S1         Semester1            01/JAN/12 15/JUL/12
S2         Semester2            16/JUL/12 31/DEC/12

SQL> select * from labtime;

    TIMEID TIME_DESC       BEGIN_TIM END_TIME
---------- --------------- --------- ---------
         1 morning         01/MAR/12 01/MAR/12
         2 afternoon       01/MAR/12 01/MAR/12
         3 night           01/MAR/12 01/MAR/12

SQL> select * from semester, labtime;

SEMID SEM_DESC   BEGIN_DATE END_DATE   TIMEID TIME_DESC BEGIN_TIME END_TIME
----- ---------- ---------- ---------- ------ --------- ---------- ---------
S1    Semester1  01/JAN/12  15/JUL/12       1 morning   01/MAR/12  01/MAR/12
S1    Semester1  01/JAN/12  15/JUL/12       2 afternoon 01/MAR/12  01/MAR/12
S1    Semester1  01/JAN/12  15/JUL/12       3 night     01/MAR/12  01/MAR/12
S2    Semester2  16/JUL/12  31/DEC/12       1 morning   01/MAR/12  01/MAR/12
S2    Semester2  16/JUL/12  31/DEC/12       2 afternoon 01/MAR/12  01/MAR/12
S2    Semester2  16/JUL/12  31/DEC/12       3 night     01/MAR/12  01/MAR/12

6 rows selected.
```

Note that Semester has 2 records, and LabTime has 3 records. The Cartesian Product will simply produce 6 records (2 x 3 records). Also see that the Cartesian Product results do not have any meaning (e.g. the 6 records do not have any meaning at all – it simply combines the records from the two tables).

Once the Cartesian Product of Semester and LabTime is produced, we can then join with UseFact table – that is to join 6 records and 1363 records to produce the final result of the query. The result of this join will be the same as the joining between UseFact and Semester and then LabTime. The reason is because the join between 6 records (Cartesian Product) and 1363 records (UseFact) will use the common attributes between the Cartesian Product and UseFact, which are semid and timeid. Therefore, the join result between the Cartesian Product and UseFact will be meaningful, and correct.

The query tree of this execution plan is shown as follows:



Using the Cartesian Product method to "join" the dimension tables would be more efficient, as the large fact tables would not need to be carried throughout the entire query processing. However, if the dimension tables are relatively big, resulting in a large Cartesian Product table, then the Cartesian Product method may not give benefits.

Comparing the Cartesian Product method with the normal join ordering, the order of the process of each method is as follows:

The order of processes using the Cartesian Product method:
1. Cartesian Product Semester and LabTime tables
2. Join the result of the Cartesian Product with the UseFact table

The order of processes using the normal join:
1. Join UseFact with Semester table
2. And the result of #1 is joined with LabTime table

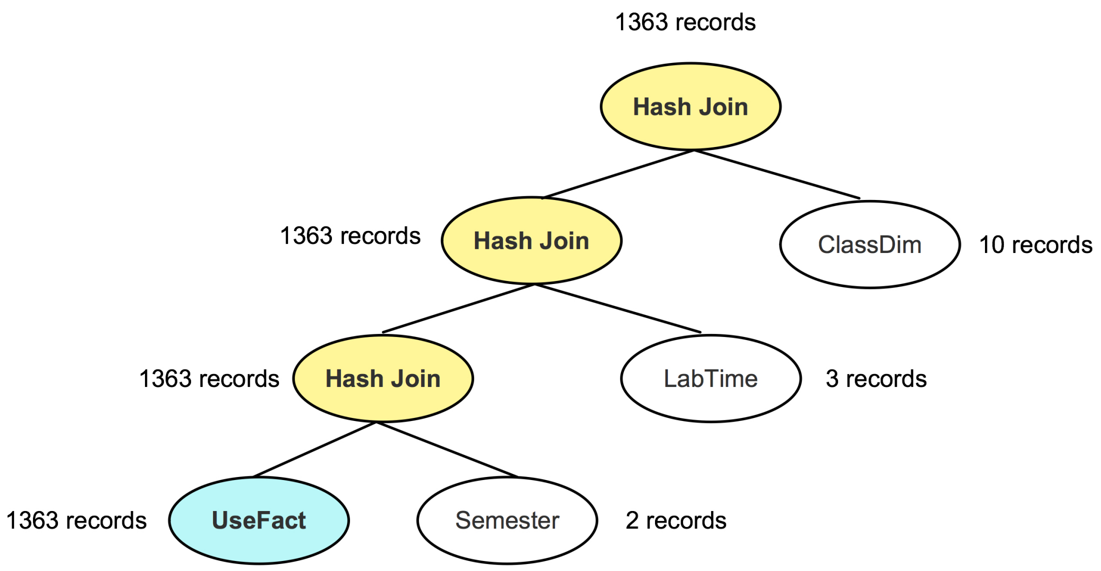
Which one is better? It depends on the size of the dimension tables and the fact table. So, you need to decide on a case-by-case basis.

3. Now, with THREE Dimensions

Let's consider the following SQL, which joins UseFact with three dimensions (e.g. Semester, LabTime, and ClassDim).

```
SQL> select *
  2  from usefact u, semester s, labtime l, classdim c
  3  where u.semid = s.semid
  4  and u.timeid = l.timeid
  5  and u.class_id = c.class_id;
```

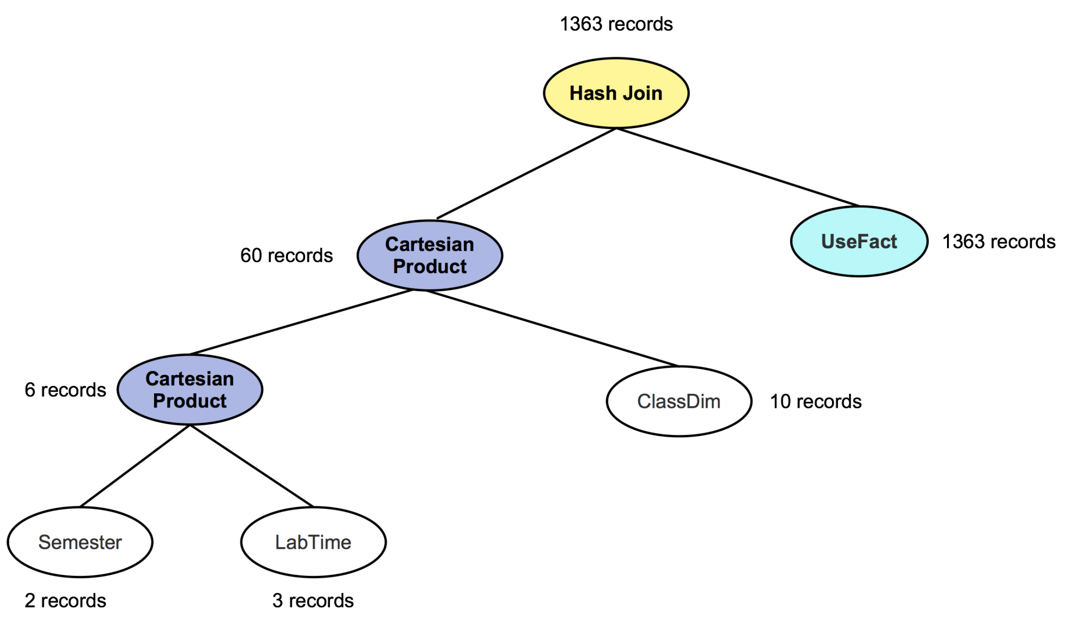
The original query tree is shown as follows.



It shows that the fact UseFact is first joined with Semester, and then with LabTime, and then with ClassDim. The join operation uses a Hash Join operation, because the Hash Join operation is the most efficient join operation. However, the main drawback of the above execution plan (query tree) is that the number of records during the three Hash join is never reduced – that is the number of records from UseFact (e.g. 1363 records).

An optimized version of the query tree is shown as follows, where Cartesian Product operation is performed first, before the join with UseFact. The Cartesian Product between Semester and LabTime produces 6 records, and then another Cartesian Product with ClassDim produces 60 records. Only then a Hash Join is performed with UseFact, resulting in 1363 records.

This Cartesian Product method is certainly more efficient than the original Hash Join method, because of the less number of records being processed.



4. with FOUR Dimensions

If we join UseFact with all the four dimensions (e.g. Semester, LabTime, ClassDim, and MajorDim), it is not a good idea to do a Cartesian Product of all of the four dimensions. The reason is as follows:

Note that the use of Cartesian Product combines dimension tables before a join with the fact table. The aim is to reduce number of records in the processing pipeline. Using the original join, the number of records that the fact table has needs to carry along the processing pipeline since there is no reduction on number of records; fact joins with dimension-1 will result in the same number of records as the fact, then the result is joined with dimension-2, the same number of records is produced, etc.

With the Cartesian Product operation, we combine smaller dimensions, but the number of records as a result of the Cartesian Product operation must be smaller than the number of records in the fact. If the Cartesian Product produces more number of records than that of the fact, then the process will be worst than the original query plan.

So, if we want to join UseFact with all the four dimensions: Semester, LabTime, ClassDim, and MajorDim, do we want to do the Cartesian Product for all the four dimensions or just some of them. Number of records in Semester, LabTime, ClassDim, and MajorDim are 2, 3, 10, and 172 records, whereas UseFact has 1363 records.

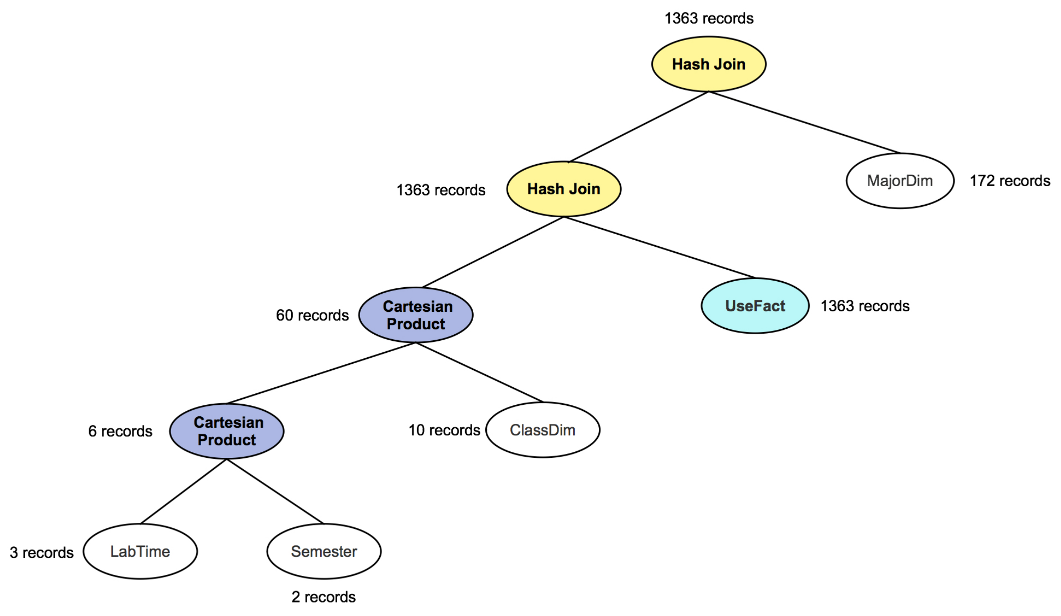
Now, let's examine the number of records if we do a Cartesian Product. A Cartesian Product between Semester and LabTime will produce only 6 records, so a Cartesian Product between Semester and LabTime is desirable.

Next, we do a Cartesian Product involving Semester, LabTime, and ClassDim. The result will be 60 records (2 x 3 x 10 records), which is also far below the fact table that has 1363 records. Hence, Cartesian Product operations for these three dimensions are desirable.

Finally, let's involve the fourth dimension, which is MajorDim. MajorDim has 172 records. If we do a Cartesian Product between the first three dimensions and MajorDim, the result will be 60 x 172 records = 10320 records, which is far more than the number of records of UseFact which is only 1363. In other words, we should not do a Cartesian Product with MajorDim. The Cartesian Product should only involve the three dimensions.

So, the process should be as follows: Do Cartesian Product operations for the first three dimensions (Semester, LabTime, and ClassDim), and then a Hash Join with the fact table (UseFact), and then another Hash Join with MajorDim.

The optimized query tree is as follows:


1363 records
Hash Join
1363 records
Hash Join
MajorDim
172 records
60 records
Cartesian Product
UseFact
1363 records
6 records
Cartesian Product
10 records
ClassDim
3 records
LabTime
Semester
2 records