Lab 9: Join Types

In a previous lab, you did the standard SQL types of joins which all work in Hive QL. However, the problem with executing any type of join in controlling the amount of computational resources required to execute the join. Since Hive queries are translated into underlying map reduce jobs, one of the computation bottlenecks is the amount of shuffling of the data required during a standard type of join.

In this lab, you will explore several different sorts of joins that are designed to improve the efficiency of executing joins in Hive

1. Map Join or Side Join

This can be used when one of the tables in the join is a small table and can be loaded into memory. This means that the join can be done completely within a mapper without using a a reducer.

Create the Data

For this, you are using two tables, one of which is quite small. Assuming you have uploaded the data trans_1000.csv and vendors.csv into the /user/maria_dev/data directory, you can create the tables by doing:

```
create table trans1000(
      id          int,
      account_id  int,
      vendor_id   int,
      transtime   string,
      city        string,
      state       string,
      amount      double)
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;

load data inpath'/user/maria_dev/data/trans_1000.csv' into table trans1000;
```

```
create table vendors (
      id          int,
      name        string,
      city        string,
      state       string ,
      category    string ,
      swipe_rate  double)
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;

load data inpath'/user/maria_dev/data/vendors.csv' into table vendors;
```

Since Hive does map side join as a default optimization, turn this feature off and then run a select query.

```
set hive.auto.convert.join=false;
select trans1000.*,  vendors.* from trans1000 join vendors on
       (trans1000.vendor_id = vendors.id) limit 10;
```

Observe the execution carefully, you should see that a reducer is used in computing the final result. Now turn the map join optimization back on and rerun the query.

```
set hive.auto.convert.join=true;
select trans1000.*,  vendors.* from trans1000 join vendors on
       (trans1000.vendor_id = vendors.id) limit 10;
```

The output should be the same but you should notice that the join was done entirely in the mapper and no reducer was involved.

2. The Left Semi Join

In a traditional RDBMS, the IN and EXISTS clauses are widely used whereas in Hive where we want one table to act as a sort of filter. The left semi join is used to get the equivalent functionality without actually performing any join operations.

Because the right-hand side table is only used to select rows from the left hand table, columns from the right hand table can only be used in the join clause but not in the WHERE or the SELECT clause.

Use INNER JOIN if you want to repeat the matching record from the left hand side table multiple times for each matching record in the right hand side but use LEFT SEMI JOIN if you want to list the matching record from the left hand side table only once for each matching record in the right hand side.

LEFT SEMI JOIN performs far more efficiently compared to the INNER JOIN.

Using the same data as the previous example, run an inner join:

```
select * from trans1000 inner join vendors on
          (trans1000.vendor_id = vendors.id) limit 10;
```

Notice that the output contains columns from both tables, just like a proper inner join should. Now convert this to a left semi join/

```
select * from trans1000 left semi join vendors on
          (trans1000.vendor_id = vendors.id) limit 10;
```

Notice that none of the columns from the right hand table are in the result. The right hand table has been used to filter rows in the left hand table.

3. The Bucket Join

When the tables are large to be joined are large and both tables used in the join are bucketed on the same oin columns, then we can do a bucket map join feature. However, in order for this to work, the number of buckets in one table should be a multiple of the buckets in the other table.

The basic idea is that each mapper can be loaded only with the buckets of data rather than the whole table. The join can then be done on matching buckets in the mapper without needing a reducer.

The data for this lab is small enough that it is hard to demonstrate the efficiency of the bucket map join, but you should get the idea of how it works

Bucket the Data

Using the data already loaded from previous sections, create two new bucketed versions of the trans1000 and vendors tables.

```
create table tb(
      id          int,
      account_id  int,
      vendor_id   int,
      transtime   string,
      city        string,
      state       string,
      amount      double)
clustered by (vendor_id) into 4 buckets
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;

insert into tb select * from trans1000;
```

```
create table vb (
      id          int,
      name        string,
      city        string,
      state       string ,
      category    string ,
      swipe_rate  double)
clustered by (id) into 2 buckets
row format delimited fields terminated by ','
lines terminated by '\n' stored as textfile;

insert into vb select * from vendors;
```

Run the Joins

Turn off the automatic map join optimization and run the join.

```
set hive.auto.convert.join=false;

select * from tb inner join vb on (tb.vendor_id = vb.id) limit 10;
```

You will notice the join uses the reducer. Set the value to true and rerun the query and you should see no reducer used in the join.

```
set hive.auto.convert.join=false;

select * from tb inner join vb on (tb.vendor_id = vb.id) limit 10;
```

Although there is no observable performance with this small amount of data, the idea is that we can extend the upper limit of the use of the map side join by bucketing data that would otherwise be too large for a straight map side join.