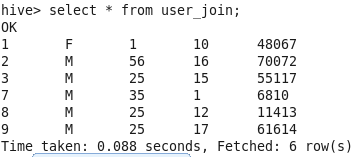
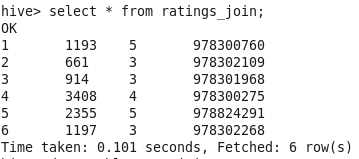
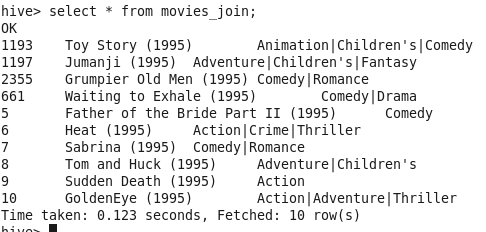
**User:**



**Rating:**



**Movies:**

****

**Cartesian Product Join:**

**A row from table A is taken and is combined with all the rows in table B** resulting in large number of resultant rows. A Cartesian product is a join where all the tuples in the left side of the join are paired with all the tuples of the right table. If the left table has 5 rows and the right table has 6 rows, 30 rows of output. Unlike other join types, **Cartesian products are not executed in parallel, and they are not optimized in any way using MapReduce.**

SELECT \* FROM ratings\_join;

SELECT \* FROM user\_join;

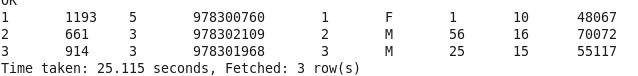
select \* from movies\_join;

SELECT \* FROM ratings\_join join user\_join; returned 36 rows(6X6)

select \* from movies\_join join user\_join; returned 60 rows(6X10)

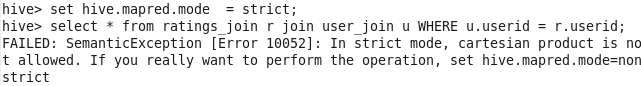
**Cartesian Join - Where Clause:**

select \* from ratings\_join r join user\_join u WHERE u.userid = r.userid;



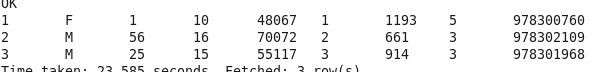
Returns same result as inner-join but performed without ON clause and is inefficient. In Hive, this query computes the full Cartesian product before applying the WHERE clause. It could take a very long time to finish.

When the property **hive.mapred.mode is set to strict**, Hive prevents users from inadvertently issuing a Cartesian product query.



**Inner Join - Same result as the above Cartesian Join:**

select \* from user\_join u join ratings\_join r on u.userid = r.userid; (Difference is table name is interchanged in this syntax)



**Note cartesian-join is different from inner-join, the latter join uses "ON" clause.**

**Map-Joins:**

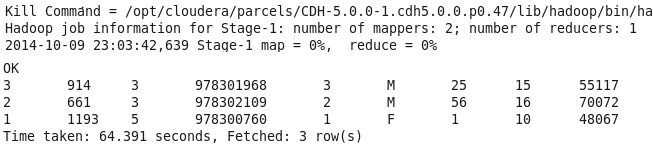
If one table is small, the largest table can be streamed through the mappers while the small tables are cached in memory. Hive can do all the joining map-side, since it can look up every possible match against the small tables in memory, thereby eliminating the reduce step. Even on smaller data sets, this optimization is noticeably faster than the normal join. Not only does it eliminate reduce steps, it sometimes reduces the number of map steps, too.

**Cartesian Join Example for Map-Joins:**

**Number of reducers is set to 1, which denotes map-join is not implemented in this step**

set hive.auto.convert.join=false;

select \* from ratings\_join r join user\_join u WHERE u.userid = r.userid;

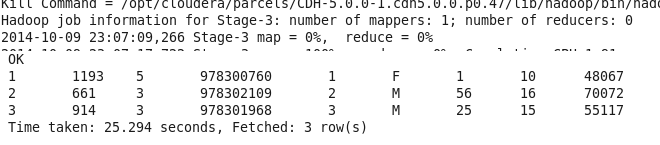


**Cartesian Join Example for Map-Joins:**

**Number of reducers is set to 0, which denotes map-join is not implemented in this step.** /\*+ MAPJOIN(r) \*/ denotes r table is the smallest and hints Hive to buffer it in memory.

set hive.auto.convert.join=true;

select /\*+ MAPJOIN(r) \*/ \* from ratings\_join r join user\_join u WHERE u.userid = r.userid;

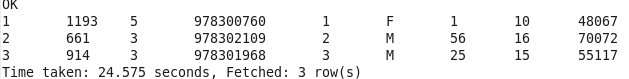


**Inner Join Example for Map-Joins:**

From HIVE 0.7

set hive.auto.convert.join=true;

select \* from ratings\_join r join user\_join u ON u.userid = r.userid;

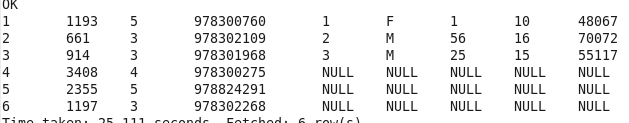


**Left outer Join Example for Map-Joins:**

From HIVE 0.7

set hive.auto.convert.join=true;

select \* from ratings\_join r left outer join user\_join u ON u.userid = r.userid;



One can also configure the threshold size for table files considered small enough to use this optimization. Here is the default definition of the property (in bytes):

hive.mapjoin.smalltable.filesize=25000000

Hive to attempt this optimization, set one or both of these properties in your $HOME/.hivercfile.

**Hive does not support the optimization for right- and full-outer joins.**

**Bucketed Map-Join:**

Hive can also join buckets between two tables without retrieving the content.

**set hive.optimize.bucketmapjoin=true;**

The conditions are Data must be bucketed on the keys used in the ON clause. The number of buckets for one table must be a multiple of the number of buckets for the other table. When these conditions are met, Hive can join individual buckets between tables in the map phase.

drop table user\_bucketed;

CREATE TABLE IF NOT EXISTS user\_bucketed (

userid INT COMMENT 'USER ID',

Gender CHAR(1) COMMENT 'User Gender',

Age INT COMMENT 'User Age',

Occupation INT COMMENT 'USER OCCUPATION',

Zipcode INT COMMENT 'User Zip-code'

)

CLUSTERED BY(userid) INTO 3 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\;'

STORED AS Textfile;

LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/input/users\_new.dat' INTO TABLE user\_bucketed;

select \* from user\_bucketed;

drop table ratings\_new\_temp;

create table ratings\_new\_temp

(

userid INT COMMENT 'USER ID',

movieid INT COMMENT 'Movie ID',

rating INT COMMENT 'Rating',

timestamp\_rating Bigint COMMENT 'Rated time'

)

CLUSTERED BY(userid) INTO 6 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\;'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/input/ratings\_new.dat' INTO TABLE ratings\_new\_temp;

select \* from ratings\_new\_temp;

**Without any Optimization: This INNER JOIN took 136s**

set hive.auto.convert.join=false;

set hive.optimize.bucketmapjoin=false;

select \* from user\_bucketed u join ratings\_new\_temp r on u.userid = r.userid;

**With Map-Side Optimization alone: This INNER JOIN took 116s**

Number of Reducers set to 0 denoting the map join is activated. Bucketed columns is userid and Variable in userid also ON clause. Bucketed column and On clause variable should be the same.

set hive.auto.convert.join=true;

select /\*+ MAPJOIN(u) \*/ \* from user\_bucketed u join ratings\_new\_temp r on u.userid = r.userid;

For the query above, the mapper processing joins 1 of Table A will with corresponding bucket of table B. Refrains itself from fetching all records.

**With both Map-Side Optimization and Bucket Join: This INNER JOIN took 106s**

If the tables being joined are sorted and bucketed, and the number of buckets are same, a sort-merge join can be performed and it will be even faster. Ensure these conditions and activate it by:

set hive.optimize.bucketmapjoin=true;

set hive.optimize.bucketmapjoin.sortedmerge=true;

select \* from user\_bucketed u join ratings\_new\_temp r on u.userid = r.userid;

drop table ratings\_new\_temp;

create table ratings\_new\_temp

(

userid INT COMMENT 'USER ID',

movieid INT COMMENT 'Movie ID',

rating INT COMMENT 'Rating',

timestamp\_rating Bigint COMMENT 'Rated time'

)

CLUSTERED BY(userid) INTO 3 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\;'

STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/input/ratings\_new.dat' INTO TABLE ratings\_new\_temp;

select \* from ratings\_new\_temp;