



# Hive & SQL

# JOINS AND TYPES

---

Type	Approach	Pros	Cons
Shuffle Join	Join keys are shuffled using map/reduce and joins performed join side.	Works regardless of data size or layout.	Most resource-intensive and slowest join type.
Broadcast Join	Small tables are loaded into memory in all nodes, mapper scans through the large table and joins.	Very fast, single scan through largest table.	All but one table must be small enough to fit in RAM.
Sort-Merge-Bucket Join	Mappers take advantage of co-location of keys to do efficient joins.	Very fast for tables of any size.	Data must be sorted and bucketed ahead of time.

# Shuffle Joins – the default

customer

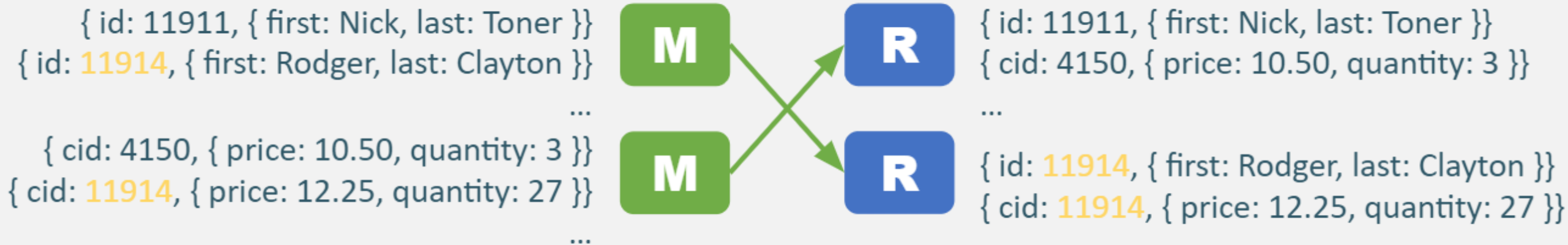
first	last	id
Nick	Toner	11911
Jessie	Simonds	11912
Kasi	Lamers	11913
Rodger	Clayton	11914
Verona	Hollen	11915

order

cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10



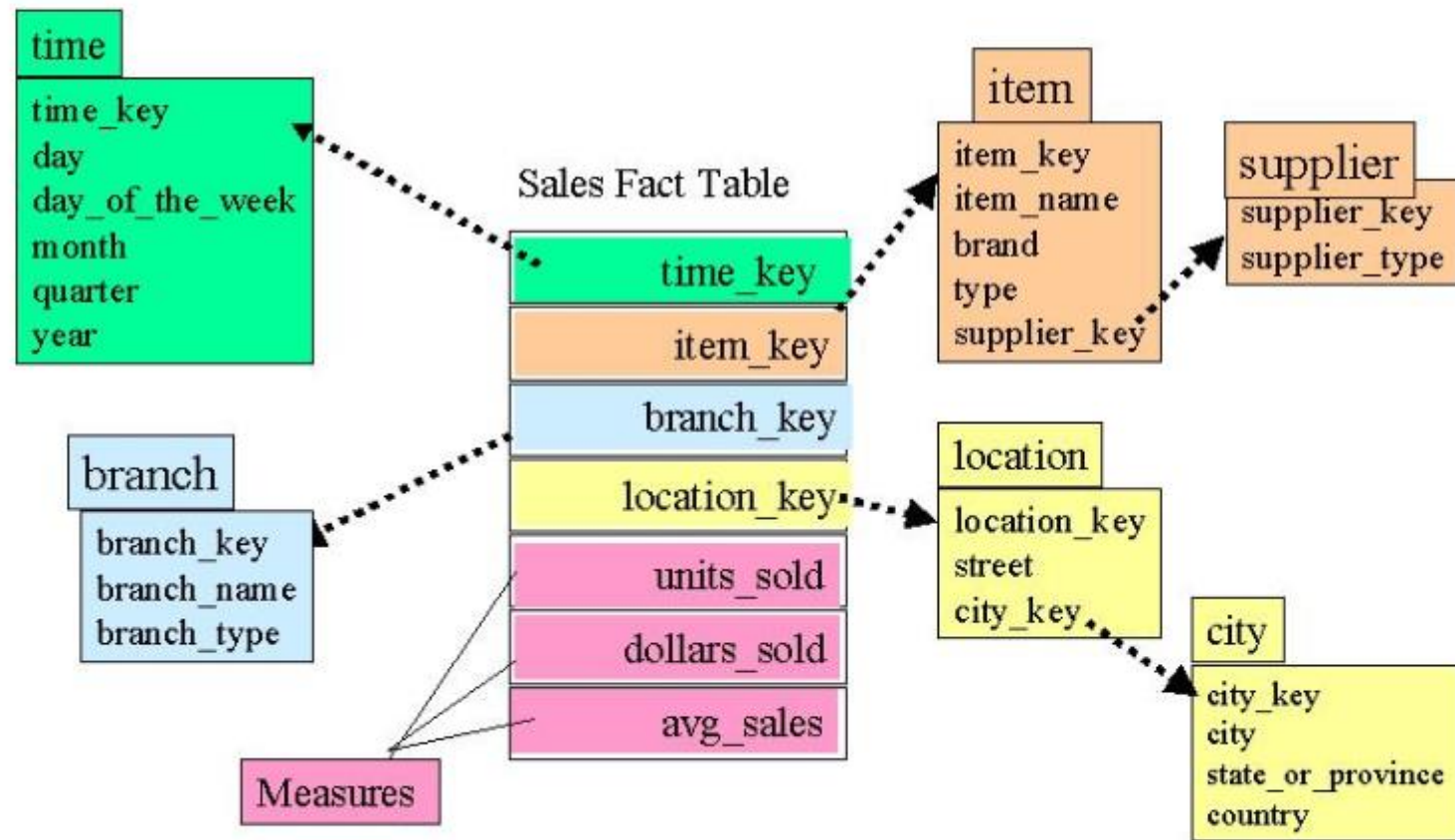
```
SELECT * FROM customer join order ON customer.id = order.cid;
```



Identical keys shuffled to the same reducer. Join done reduce-side.  
Expensive from a network utilization standpoint.

# Broadcast Join (aka Map-side Join)

- Star schemas (e.g. dimension tables)
- Good when table is small enough to fit in RAM



# USING BROADCAST JOIN

Set `hive.auto.convert.join = true`

HIVE then automatically uses broadcast join, if possible

- Small tables held in memory by all nodes
- Single pass through the large table

Used for star-schema type joins common in Data warehousing use-cases

`hive.auto.convert.join.noconditionaltask.size` (by default 10 MB) determines data size for automatic conversion to broadcast join:

- Default 10MB is too low (check your default)
- Recommended: 256MB for 4GB container

# Sort-Merge-Bucket join:

## When both are too large for memory

customer			order		
first	last	id	cid	price	quantity
Nick	Toner	11911	4150	10.50	3
Jessie	Simonds	11912	11914	12.25	27
Kasi	Lamers	11913	11914	40.50	10
Rodger	Clayton	11914	12337	39.99	22
Verona	Hollen	11915	15912	40.50	10

```
CREATE TABLE order (cid int, price float, quantity int)
CLUSTERED BY(cid) SORTED BY(cid) INTO 32 BUCKETS;
```

```
CREATE TABLE customer (id int, first string, last string)
CLUSTERED BY(id) SORTED BY(id) INTO 32 BUCKETS;
```

```
SELECT * FROM customer join order ON customer.id = order.cid;
```

# OPTIMIZATIONS IN HIVE

- As join requires data to be shuffled across nodes, use filtering and projection as early as possible to reduce data before join.

## TUNE CONFIGURATIONS

### •Compress map/reduce output

SET mapred.compress.map.output =true;

SET mapred.output.compress =true;

### •Parallel execution

- Applies to MapReduce jobs that can run in parallel, for example jobs processing different source tables before a join.

SET hive.exec.parallel =true;

### •Vectorization

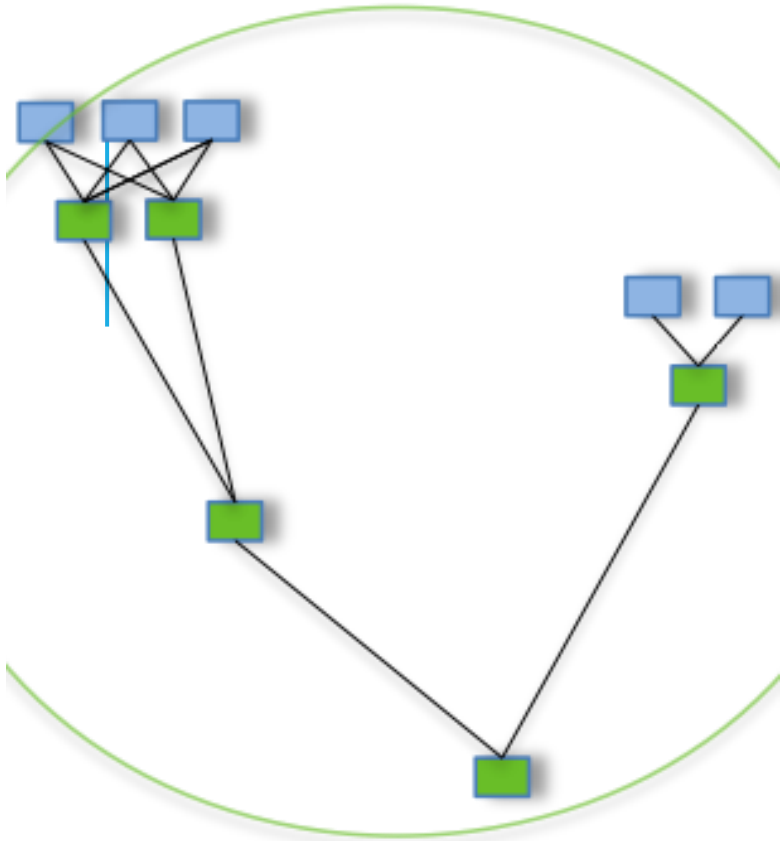
Vectorized query execution is a Hive feature that greatly reduces the CPU usage for typical query operations like scans, filters, aggregates, and joins

Vectorized query execution streamlines operations by processing a block of 1024 rows at a time (instead of 1 row at a time)

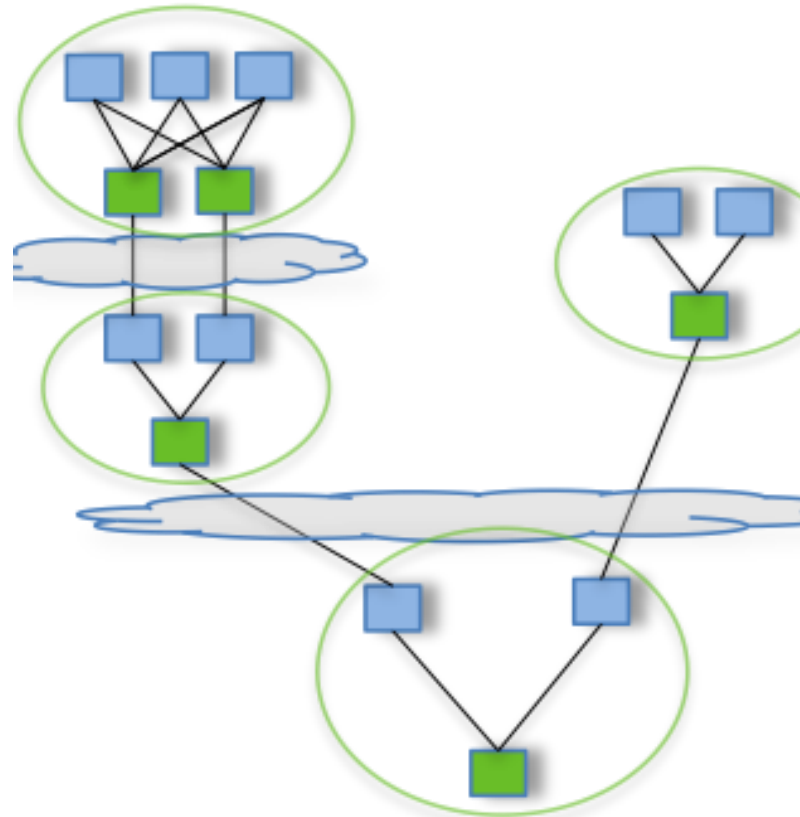
ONLY works with ORCFiles

SET hive.vectorized.execution.enabled = true;

SET hive.vectorized.execution.reduce.enabled=true;



Pig/Hive - Tez



Pig/Hive - MR

# OPTIMIZATIONS IN HIVE

## USE TEZ

- With Hadoop2 and Tez , the cost of job submission and scheduling is minimized.
- Also Tez does not restrict the job to be only Map followed by Reduce; this implies that all the query execution can be done in a single job without having to cross job boundaries.
- Creates a Directed Acyclic Graph to execute in shortest path possible.





# HIVE CBO



Cost based optimization using Apache Calcite project reduces the shuffle and does below optimizations-



- How to order Join



- What algorithm to use for a given Join



- Should the intermediate result be persisted or should it be recomputed



- The degree of parallelism at any operator (specifically number of reducers to use).



- Semi Join selection

# HIVE COST-BASED OPTIMIZATION (CBO)

**Cost-Based Optimization** (CBO) engine uses statistics within Hive tables to produce optimal query plans

Two types of stats used for optimization:

- Table stats
- Column stats

Uses an open-source framework called **Calcite** (formerly Optiq)

Stats are collected at the table level automatically when: `set hive.stats.autogather=true;`

If you have an existing table without stats collected:

CBO uses statistics about Hive tables, table partitions, and columns within a table to produce good query execution plans.

- More efficient query plans better utilize cluster resources and improve query latency.
- CBO is most useful for complex queries containing multiple JOIN statements and for queries on very large tables.
- CBO uses column level statistics to come up with better query plans.

Below parameters needs to be set to enable CBO:

```
set hive.cbo.enable=true;
```

```
set hive.compute.query.using.stats=true;
```

```
set hive.stats.fetch.column.stats=true;
```

```
set hive.stats.fetch.partition.stats=true;
```

# HIVE ACID

1. New type of table that supports Insert/Update/Delete/Merge SQL operations (SQL 2011)
- 2.
3. Concept of ACID transaction (Atomic, Consistent, Isolated, Durable)

```
CREATE TABLE T(a int, b int)
CLUSTERED BY (b) INTO 8 BUCKETS STORED AS ORC
TBLPROPERTIES ('transactional'='true');
```

1. Not all tables support transactional semantics
2. Non ACID tables must be re-created

## Structure limitations:

1. Table must be bucketed
2. Table cannot be sorted
3. Requires ORC File

```
DELETE FROM hello_acid WHERE key = 2;
UPDATE hello_acid SET value = 10 WHERE key = 3
```

# Syntax : Merge Statement – SQL Standard 2011

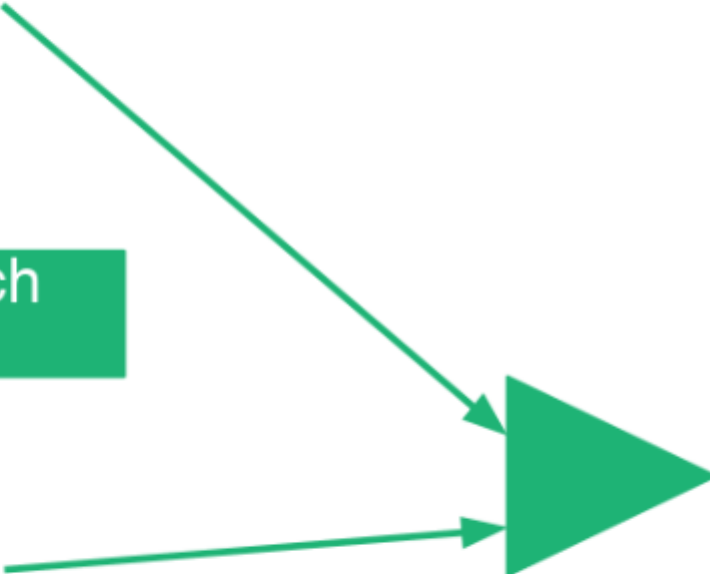
Target : table being modified

ID	State	County	Value
1	CA	LA	19.0
2	MA	Norfolk	15.0
7	MA	Suffolk	50.15
16	CA	Orange	9.1

Source : table from which  
modification are read

ID	State	Value
1		20.0
7		80.0
100	NH	6.0

```
MERGE INTO TARGET T
  USING SOURCE S ON T.ID=S.ID
    WHEN MATCHED THEN
      UPDATE SET T.Value=S.Value
    WHEN NOT MATCHED
      THEN INSERT (ID,State,Value)
        VALUES(S.ID, S.State, S.Value)
```



ID	State	County	Value
1	CA	LA	20.0
2	MA	Norfolk	15.0
7	MA	Suffolk	80.0
16	CA	Orange	9.1
100	NH	null	6.0

# SOLVE SMALL FILES PROBLEM

1. Sometimes there are 100s of small files of e.g. 100 KB which is very less than the block size
2. Unnecessary blocks will be occupied
3. Unnecessary shuffle and sort will happen
4. To avoid this we have to merge the files in a table on a periodic basis
5. Run query **ALTER TABLE TABLENAME CONCATENATE;**
6. This query will merge all the small files into 256 MB(Block size) file each.