Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

**Learning & Development**

*Enabling development, Impacting growth…*

# Hive Programming_Module_1

# Module-1 Outline

- Motivation to Hive
- What is Hive?
- Why do we use Apache Hive?
- Comparing Hive with traditional database
- Hive Architecture
- Hive configuration property
- Basic SELECT Queries
- Submitting Hive Queries
- Query Processing in Hive

# Motivation to Hive

- MapReduce code is typically written in Java
  - Although it can be written in other languages using Hadoop Streaming API

- Requires:
  - A programmer
  - Who is a good Java programmer
  - Who understands how to think in terms of MapReduce
  - Who understands the problem they’re trying to solve
  - Who has enough time to write and test the code
  - Who will be available to maintain and update the code in the future as requirements change

# Motivation to Hive (cont..)

- Many organizations have only a few developers who can write good MapReduce code

- Meanwhile, many other people want to analyze data
  - Business analysts
  - Data scientists
  - Statisticians
  - Data analysts

- What’s needed is a higher-level abstraction on top of MapReduce
  - Providing the ability to query the data without needing to know MapReduce intimately
  - Hive address these needs

# What is Hive?

- Hive is an abstraction on top of MapReduce
- Allows users to query data in the Hadoop cluster without knowing Java or MapReduce
- Hive provides a SQL-like interface to data stored in an Apache Hadoop based data warehouse.
- Hive was originally developed by Facebook for data warehousing ,

  Now an open-source Apache Project

# What is Hive? (cont..)

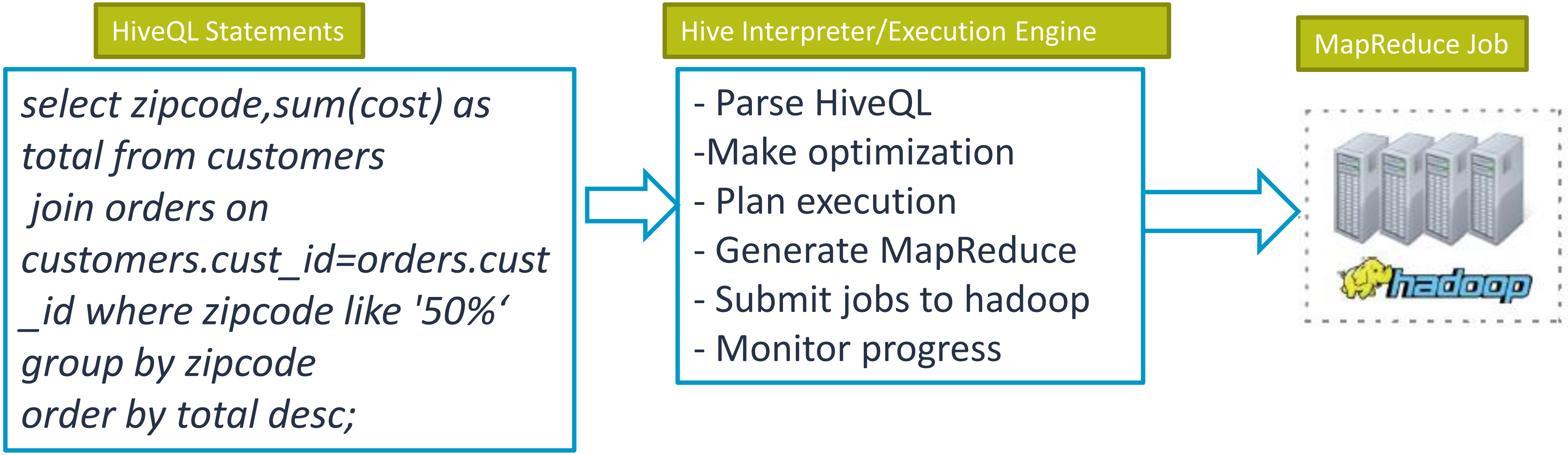
- **Intuitive**

  - Make the structured data looks like tables regardless how it is really stored in Hadoop File system
  - Hive enables analysis of large data sets using a HiveQL language very much similar

    to standard ANSI SQL
  - Generate specific execution plan for the query
  - Hive translates the query into Java Map Reduce code and runs the same on Hadoop cluster.

# What is Hive? (cont..)

- Hive interpreter runs on the client machine
  - Turns HiveQL  statements into MapReduce jobs
  - Submit those jobs and Execute across a Hadoop cluster



HiveQL Statements

*select zipcode,sum(cost) as total from customers join orders on customers.cust_id=orders.cust_id where zipcode like '50%' group by zipcode order by total desc;*

Hive Interpreter/Execution Engine

- Parse HiveQL
-Make optimization
- Plan execution
- Generate MapReduce
- Submit jobs to hadoop
- Monitor progress

MapReduce Job

# Why do we use Apache Hive?

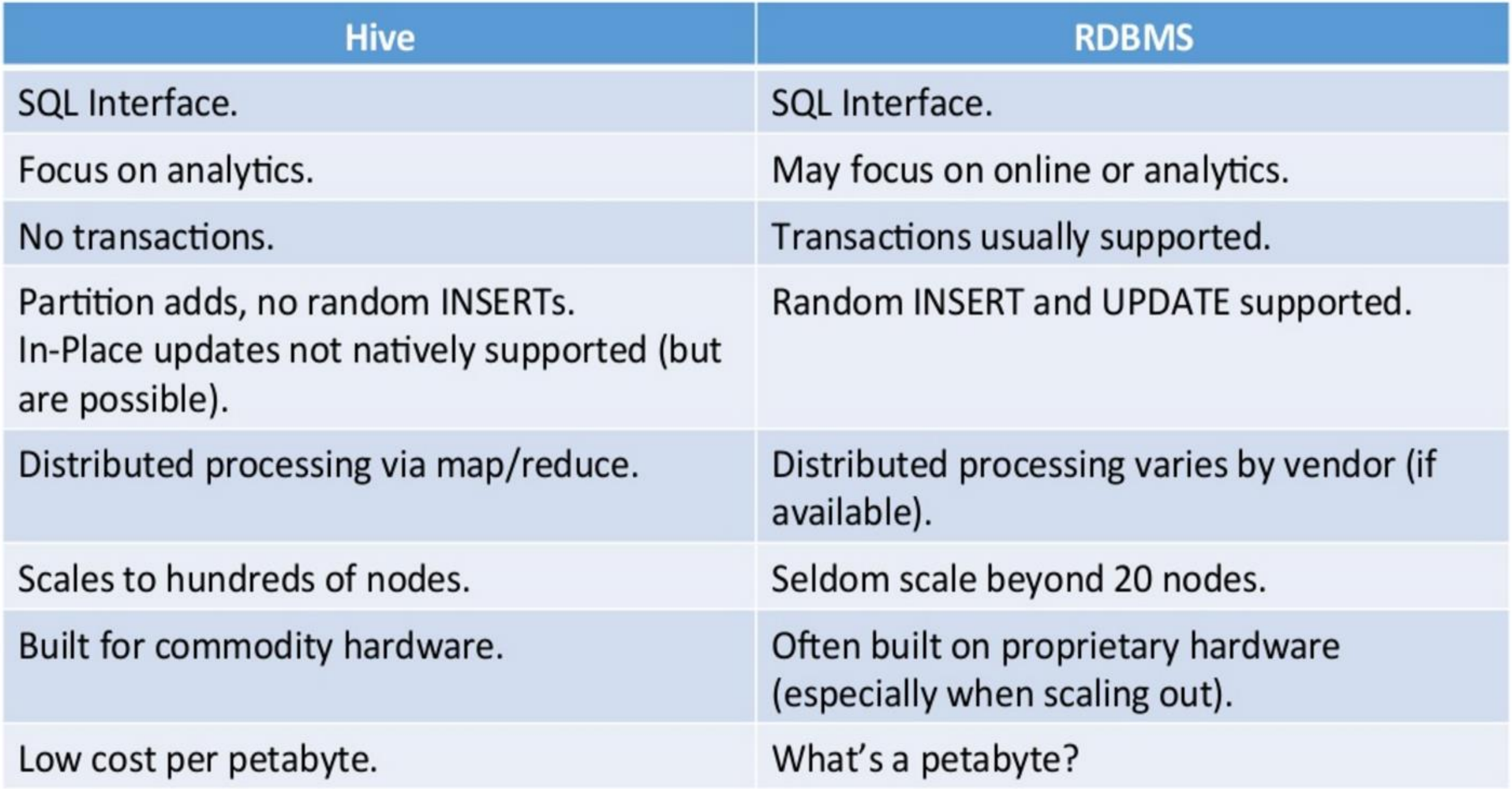
- More productive than writing complex MapReduce codes
  - Few lines of HiveQL query might be equivalent to 100 or more lines of Java code

- Brings large-scale data analysis to a broader audience
  - No Software development experience required
  - Leverage existing knowledge of SQL

- Offers interpretability with other systems
  - Extensible through java and external scripts
  - Many BI tools supports hive

# Comparing Hive with traditional database

# Hive Vs RDBMS

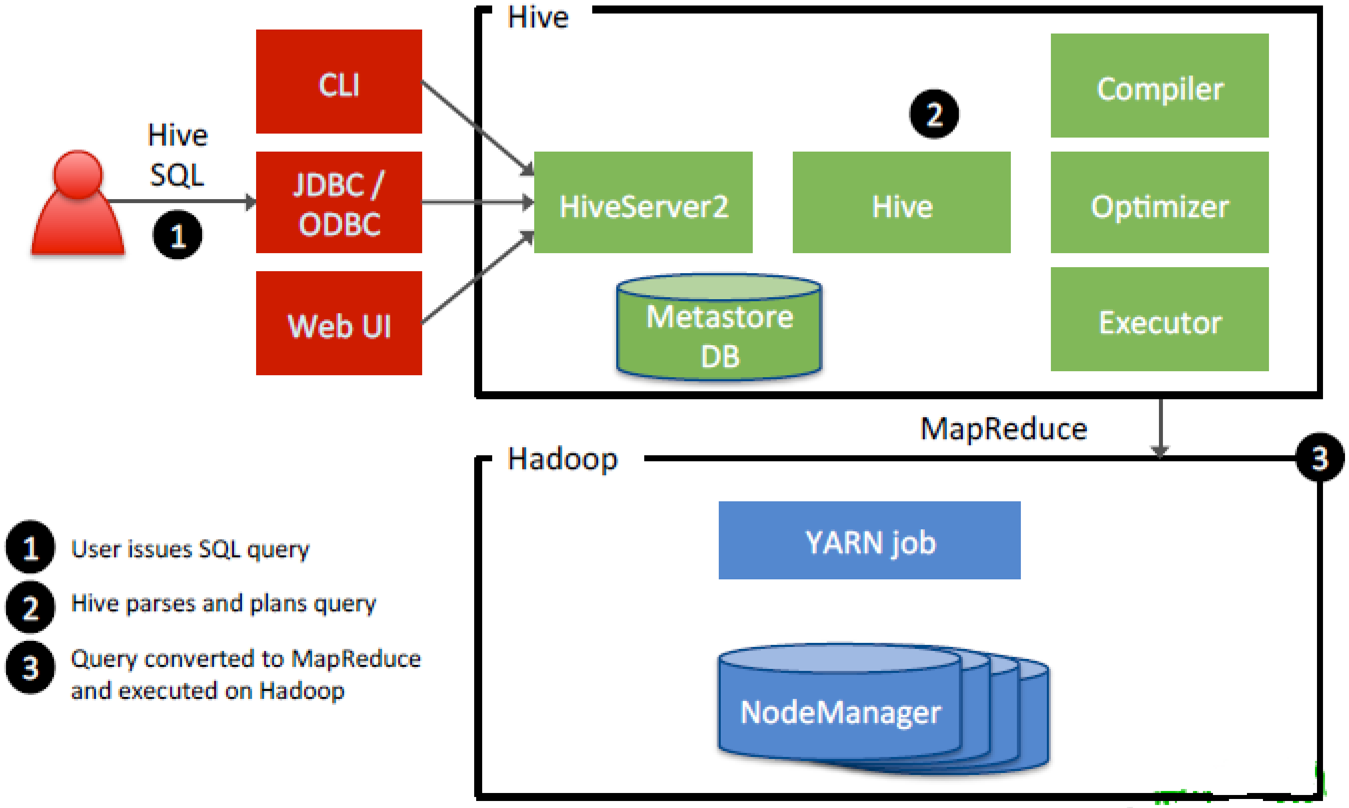


| Hive | RDBMS |
|---|---|
| SQL Interface. | SQL Interface. |
| Focus on analytics. | May focus on online or analytics. |
| No transactions. | Transactions usually supported. |
| Partition adds, no random INSERTs.<br>In-Place updates not natively supported (but are possible). | Random INSERT and UPDATE supported. |
| Distributed processing via map/reduce. | Distributed processing varies by vendor (if available). |
| Scales to hundreds of nodes. | Seldom scale beyond 20 nodes. |
| Built for commodity hardware. | Often built on proprietary hardware (especially when scaling out). |
| Low cost per petabyte. | What's a petabyte? |

# Your Cluster is not your database

- Client-server base RDBMS database management systems have many strengths
  - Very fast response time
  - support for transactions
  - Allows modification of existing records
  - Can serve thousands of simultaneous clients
- Hive does not turn your Hadoop cluster into an RDBMS
- It simply produce MapReduce jobs from HiveQL queries
- Limitations of HDFS and MapReduce still apply
- Hive is best suited for Data Warehousing applications where data is structured, static and formatted.
- Hive queries have higher latency due to start up overhead as most Hive queries turn out into Map Reduce jobs

# Hive Architecture



1 User issues SQL query

2 Hive parses and plans query

3 Query converted to MapReduce and executed on Hadoop

# Hive configuration property

- Hive configuration files are Primarily written in XML
- A number of configuration variables in Hive can be used by the Hive administrator , few of those are mentioned below:
- Hive configuration file name is  **hive-site.xml**.
- In hive-site.xml. This is used for setting values for the entire Hive configuration For example:

```
<property>
  <name>hive.exec.scratchdir</name>
  <value>/tmp/mydir</value>
  <description>Scratch space for Hive jobs</description>
</property>
```

# Basic SELECT Queries

- **Hive supports most familiar SELECT syntax**

```
hive> SELECT * FROM shakespeare LIMIT 10;

hive> SELECT * FROM shakespeare
WHERE freq > 100 ORDER BY freq ASC
LIMIT 10;
```

# Submitting Hive Queries

- Hive CLI
  - Traditional Hive client that connects to a HiveServer instance
  - To launch the Hive shell, start a terminal and run

    *$ hive*

  - Results in the Hive prompt:

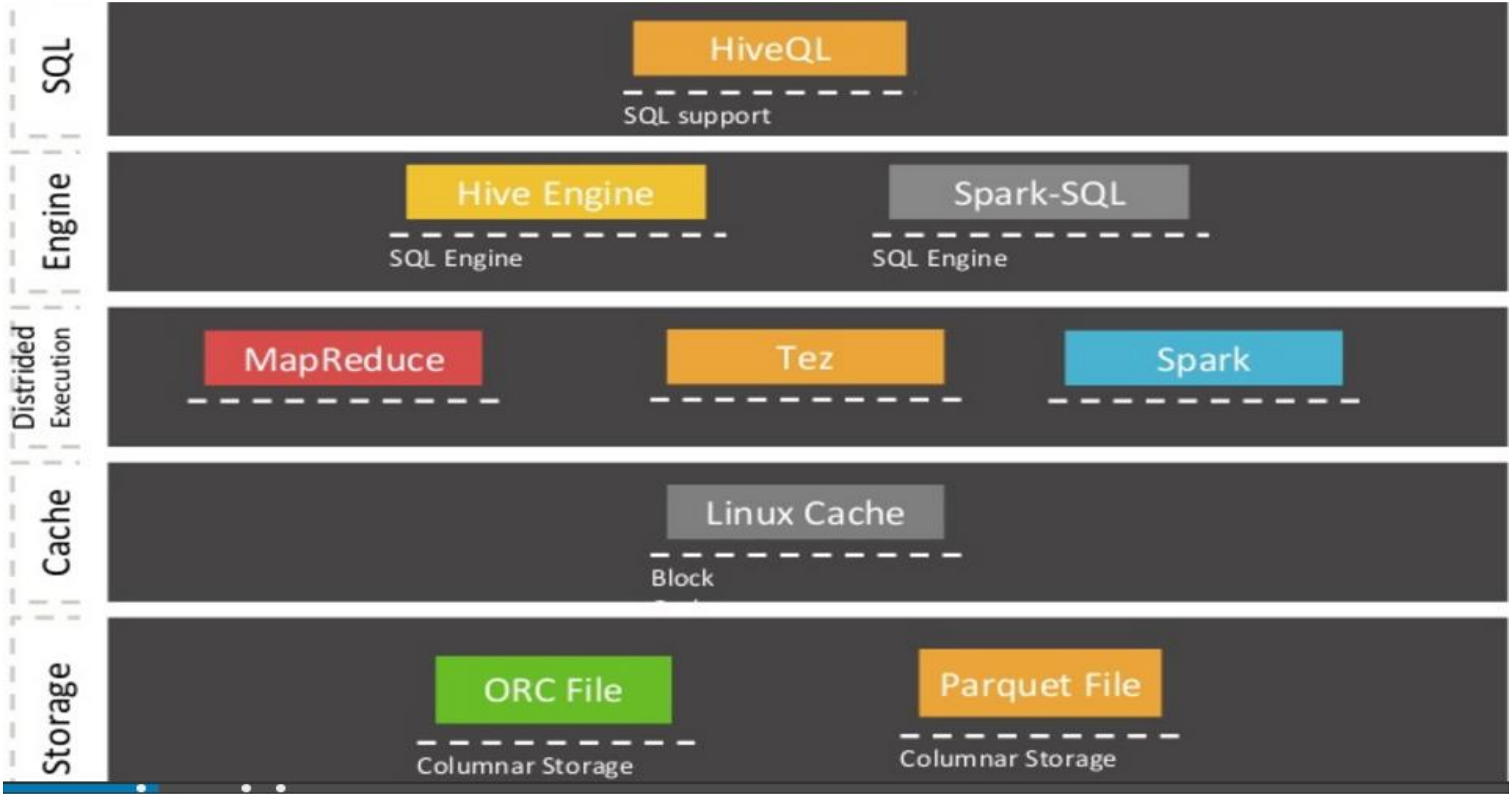
    hive>

- Beeline
  - A new command line client that connects to a HiveServer2 instance

    *$ beeline*

  - Hive version 0.11.0-SNAPSHOT by Apache beeline>

# Query Processing in Hive

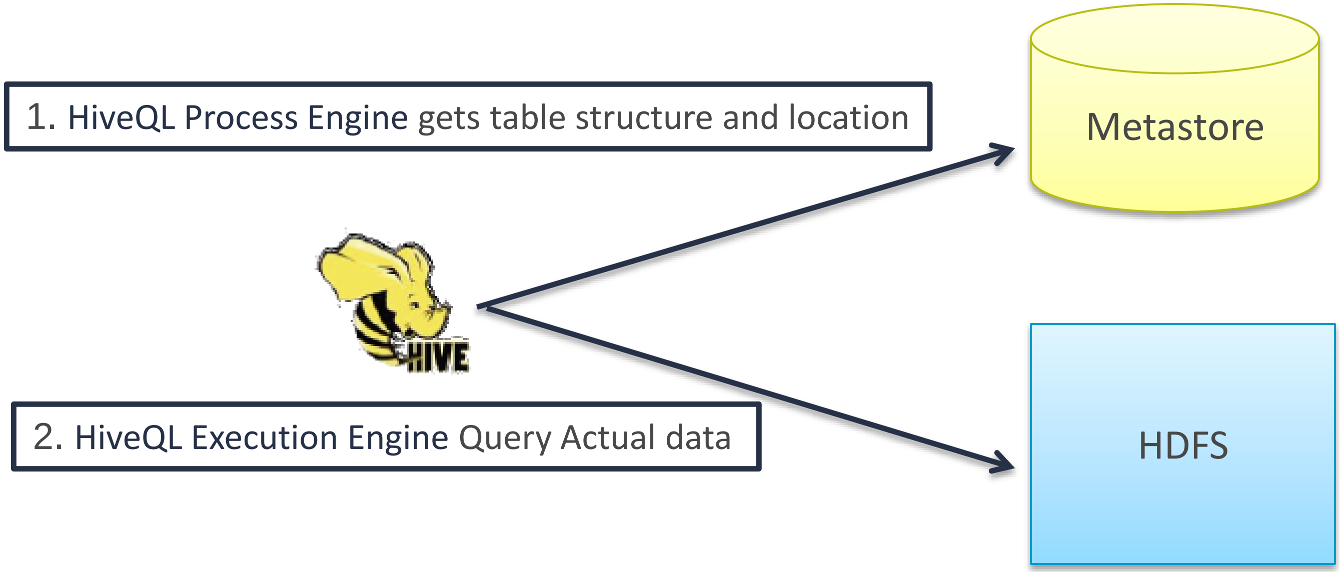


# Module-2 Outline

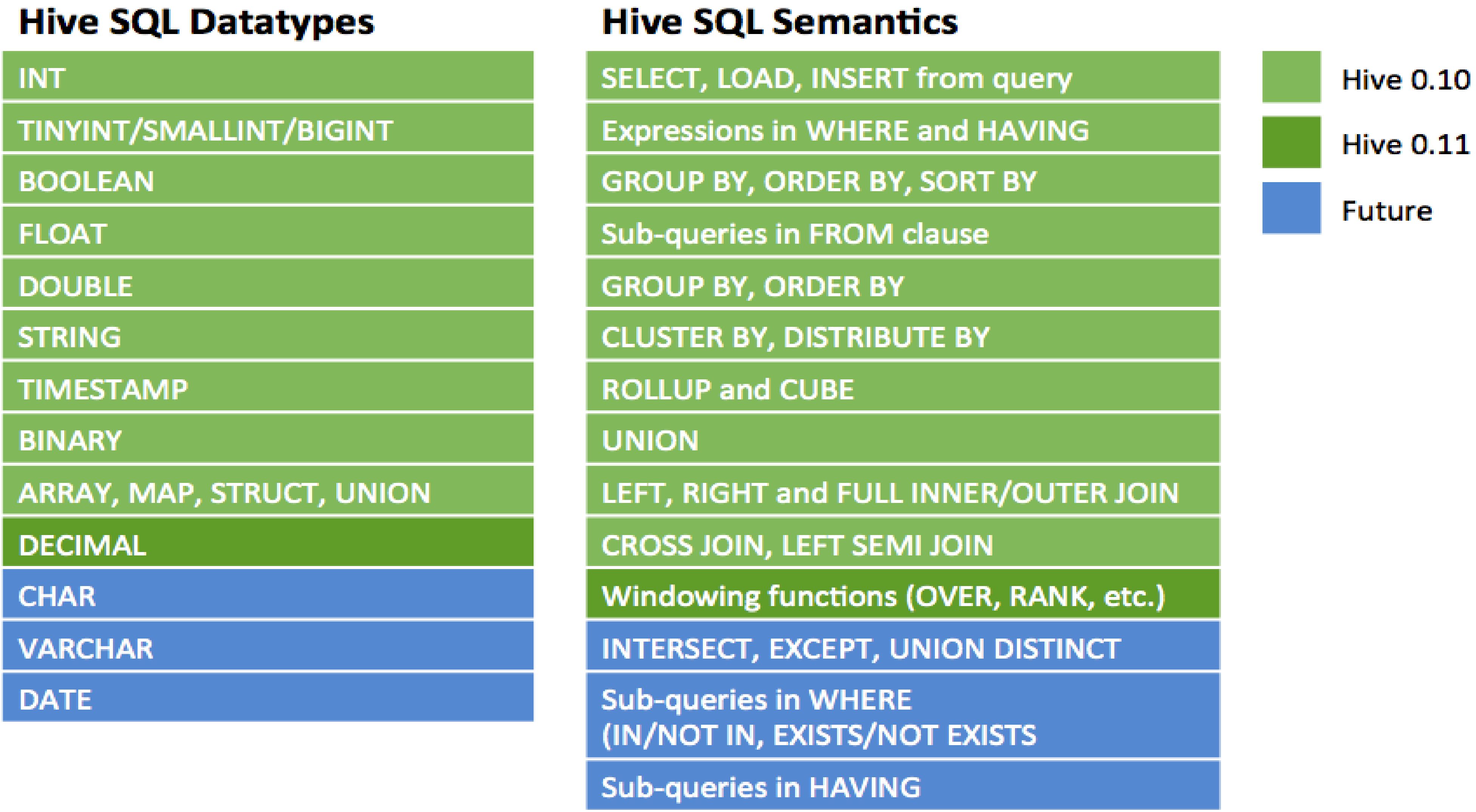
- Hive Data Model
- Hive Data Types
- Hive Metastore
- Types of Tables in Hive - Managed table and External
- Hive Metastore
- Hive Basics: Creating Tables
- Verify Table Types in Hive
- Loading Data into a Hive Table

# Hive Data Model

- How Hive  loads data and store it?
- All the metadata for Hive tables and partitions are accessed through the Hive Metastore
- The HiveQL  itself operates on data stored on a filesystem (typically HDFS )


1. HiveQL Process Engine gets table structure and location
Metastore
HIVE
2. HiveQL Execution Engine Query Actual data
HDFS

# Hive Data Types



### Hive SQL Datatypes

- 🟩 INT
- 🟩 TINYINT/SMALLINT/BIGINT
- 🟩 BOOLEAN
- 🟩 FLOAT
- 🟩 DOUBLE
- 🟩 STRING
- 🟩 TIMESTAMP
- 🟩 BINARY
- 🟩 ARRAY, MAP, STRUCT, UNION
- ▩ DECIMAL
- 🟦 CHAR
- 🟦 VARCHAR
- 🟦 DATE

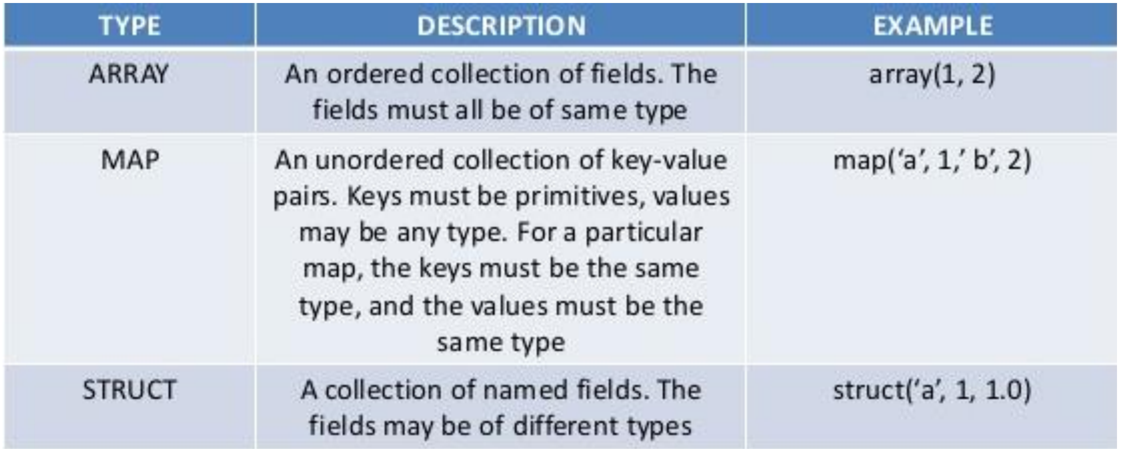
### Hive SQL Semantics

- 🟩 SELECT, LOAD, INSERT from query
- 🟩 Expressions in WHERE and HAVING
- 🟩 GROUP BY, ORDER BY, SORT BY
- 🟩 Sub-queries in FROM clause
- 🟩 GROUP BY, ORDER BY
- 🟩 CLUSTER BY, DISTRIBUTE BY
- 🟩 ROLLUP and CUBE
- 🟩 UNION
- 🟩 LEFT, RIGHT and FULL INNER/OUTER JOIN
- 🟩 CROSS JOIN, LEFT SEMI JOIN
- ▩ Windowing functions (OVER, RANK, etc.)
- 🟦 INTERSECT, EXCEPT, UNION DISTINCT
- 🟦 Sub-queries in WHERE (IN/NOT IN, EXISTS/NOT EXISTS
- 🟦 Sub-queries in HAVING

| | |
|---|---|
| 🟩 | Hive 0.10 |
| ▩ | Hive 0.11 |
| 🟦 | Future |

# Hive Data Model (cont..)

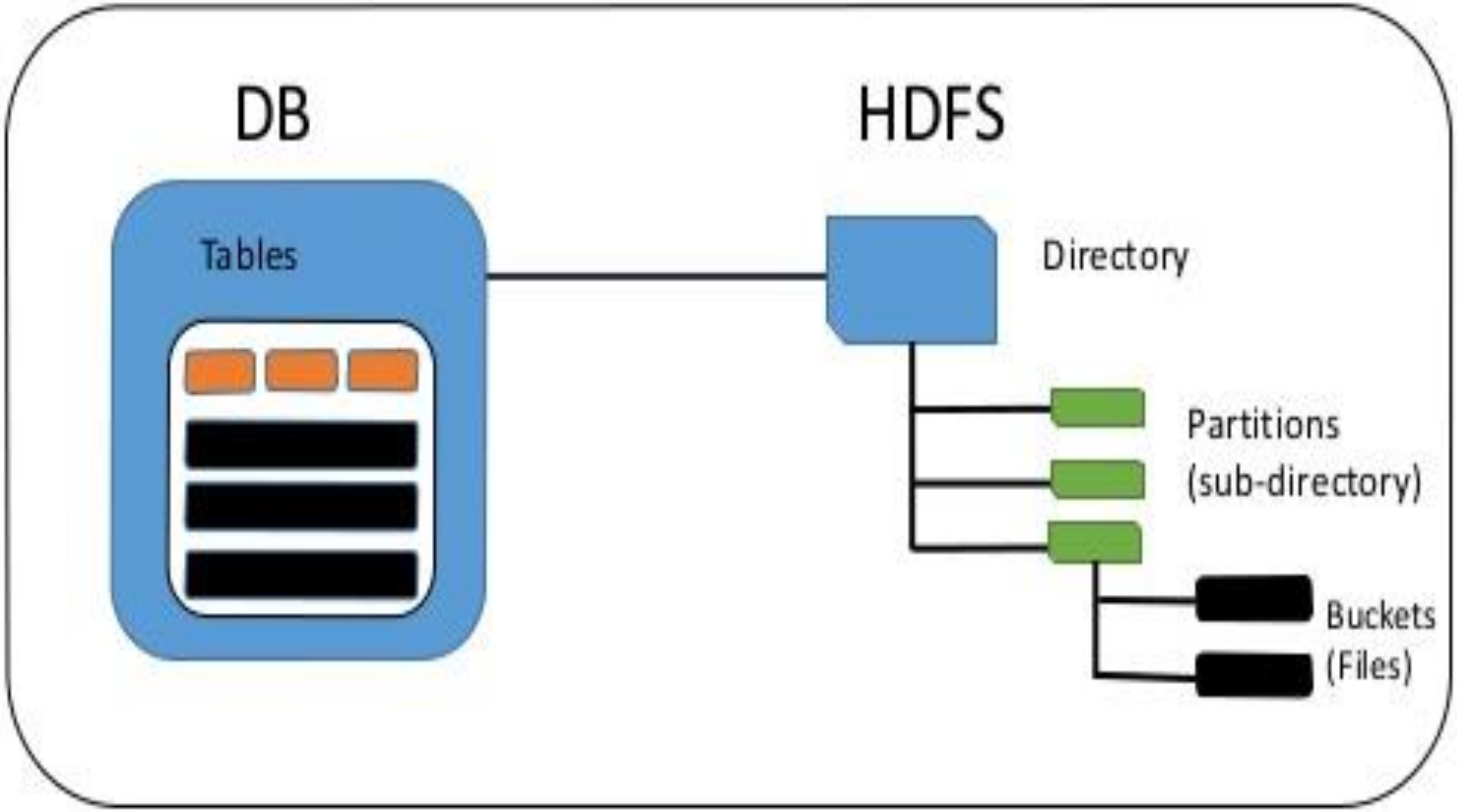
- Hive ‘layers’ table definitions on top of data in HDFS
- Tables
  - Typed columns (int, float, string, boolean and so on)
  - Also Array , Struct , Union,  list: map (for JSON-like data)
- Partitions
  - e.g., to range-partition tables by date
- Buckets
  - Hash partitions within ranges (useful for sampling, join optimization)



| TYPE | DESCRIPTION | EXAMPLE |
|---|---|---|
| ARRAY | An ordered collection of fields. The fields must all be of same type | array(1, 2) |
| MAP | An unordered collection of key-value pairs. Keys must be primitives, values may be any type. For a particular map, the keys must be the same type, and the values must be the same type | map(‘a’, 1,’ b’, 2) |
| STRUCT | A collection of named fields. The fields may be of different types | struct(‘a’, 1, 1.0) |

# Hive Data Model (cont..)

- Hive structure data into a well defined database concept i.e Tables , columns and rows, partitions ,buckets etc .



# Hive Metastore

- Hive’s Metastore is a database containing table definitions and other metadata
  - By default, stored locally on the client machine in a Derby database
  - If multiple people will be using Hive, the system administrator should create a shared Metastore
  - Usually in MySQL or some other relational database server

# Types of Tables in Hive

# Managed table and External table in Hive

- There are two types of tables in Hive ,
  - Managed table
  - External table
- The difference is , when you drop a table, if it is managed table hive deletes both data and meta data, if it is external table Hive only deletes metadata
- External table files are accessible to anyone who has access to HDFS file structure and therefore security needs to be managed at the HDFS file/folder level.

**Table Creation:** by default table is Managed table .If you want to create a external table , you will use external keyword.

**Syntax – Managed Table**

create table managedemp(col1 datatype,col2 datatype, ....) row format delimited fields terminated by 'delimiter character‘ location '/data/employee‘

**Syntax – External Table**

create external table managedemp(col1 datatype,col2 datatype, ....) row format delimited fields terminated by 'delimiter character‘ location '/data/employee'

# Q & A

What is difference between managed table and external Table?

# Hive Basics: Creating Tables

```
hive> SHOW TABLES;

hive> CREATE TABLE shakespeare (freq INT, word STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

hive> DESCRIBE shakespeare;
```

# Verify Table Types in Hive

## How do you check whether existing table is managed or external table?

- How do you check whether existing table is managed or external table?
- To check that we use describe command like below

**Syntax:**

describe formatted tablename;

Example: If it is managed table ,you will see

Table Type:             MANAGED_TABLE

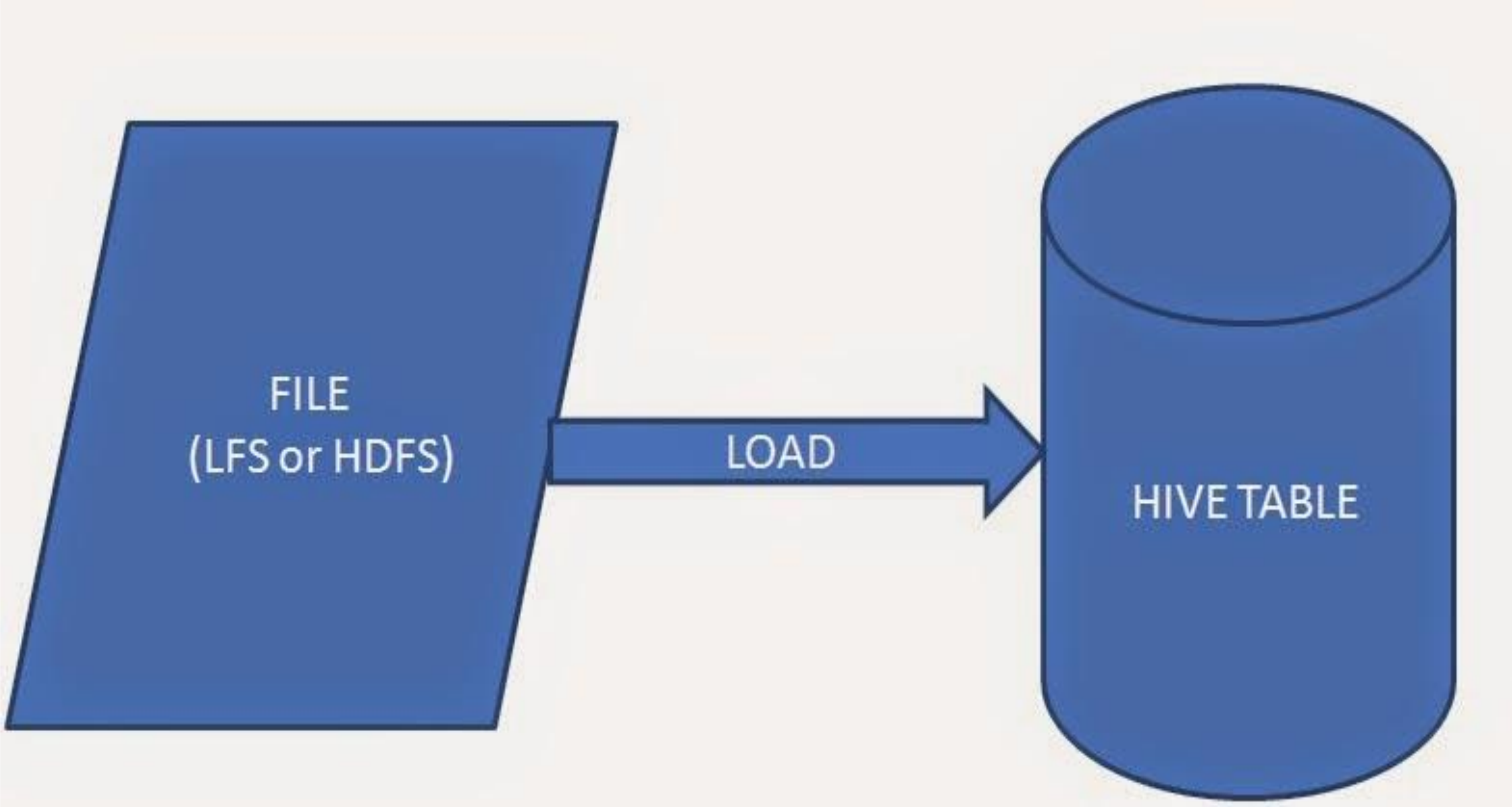
If it is external table ,you will see

Table Type:             EXTERNAL_TABLE

# Loading Data into a Hive Table

- Hive does not do any transformation while loading data into tables.
- Load operations are currently pure copy/move operations that move data files into locations corresponding to Hive tables.
- The load command will look for file path in the local file system
- If the keyword LOCAL is not specified, then the load command will look for file path in the HDFS
- If the OVERWRITE keyword is used then contents of the target table (or partition) will be deleted and replaced by the files referred to by file path

# Loading files into Hive tables



# Storing Output Results

- The SELECT statement on the previous slide would write the data to the console
- To store the results in HDFS, create a new table then write, for example:

```
INSERT OVERWRITE TABLE newTable
SELECT s.word, s.freq, k.freq FROM shakespeare s JOIN kjv k ON
(s.word = k.word) WHERE s.freq >= 5;
```

- Results are stored in the table
- Results are just files within the newTable directory
  - Data can be used in subsequent queries, or in MapReduce jobs

# Hive Data: Physical Layout

- Hive tables are stored in Hive's 'warehouse' directory in HDFS
  - By default, /user/hive/warehouse
- Tables are stored in subdirectories of the warehouse directory
  - Partitions form subdirectories of tables
- Possible to create external tables if the data is already in HDFS and should not be moved from its current location
- Actual data is stored in flat files
  - Control character-delimited text, or SequenceFiles
  - Can be in arbitrary format with the use of a custom Serializer/ Deserializer ('SerDe')

# Computing Table Statistics

- ***ANALYZE TABLE*** *tablename COMPUTE  STATISTICS;*
- ***DESCRIBE FORMATTED*** *tablename*
- *DESCRIBE* ***EXTENDED*** *tablename*
- Hive provides an EXPLAIN command that shows the execution plan for a query. The syntax for this statement is as follows:
- *hive> EXPLAIN EXTENDED select * from demo where customer =‘ICICI';*

# Storage - Hive File Formats

HiveQL handles structured data only. By default Hive has derby database to store the data in it. We can configure Hive with MySQL database. Data is eventually stored in files. There are some specific file formats which Hive can handle such as:

- Text file
- SequenceFile
- RCFile
- ORC File

# Storage - Hive File Formats (cont..)

- **Text file**: comma, tab ,space , JSON (Java Script Object Notation) data or other delimited file types. By default if we use TEXTFILE format then each line is considered as a record.

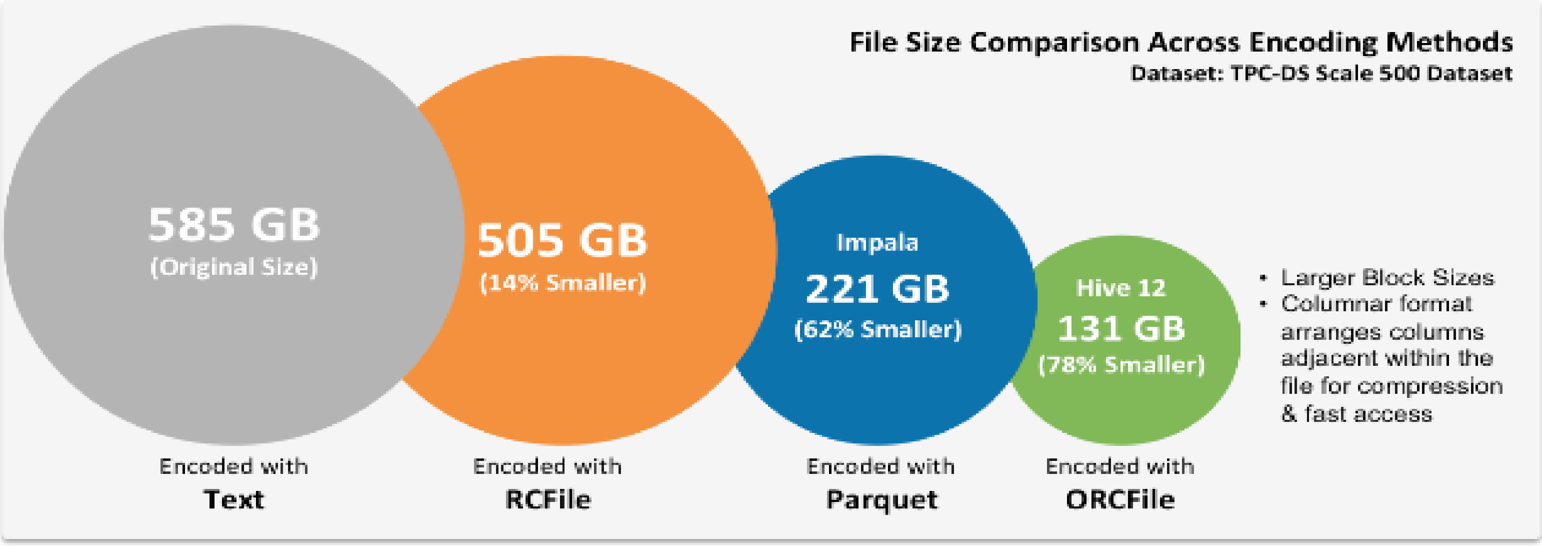
- **SequenceFile**: Sequence files are flat files consisting of binary key-value pairs. Stores serialized key/value pairs that can quickly be deserialized in  Hadoop.
- **RCFILE :** Record Columnar are flat files consisting of binary key/value pairs, which shares much similarity with SEQUENCEFILE. RCFILE stores columns of a table in form of record in a columnar manner in binary file format which offers high compression rate on the top of the rows.

# Compressed Data Storage

- Keeping data compressed in Hive tables
- Give better performance than uncompressed storage
- You can import text files compressed with Gzip or Bzip2 directly into a table stored as TextFile.
- The compression will be detected automatically and the file will be decompressed on-the-fly during query execution

# ORCFile format

- Higher Compression Technique
- ORCFile was introduced in Hive 0.11 and offered excellent compression, delivered through a number of techniques including run-length encoding, dictionary encoding for strings and bitmap encoding.
- ORC reduces the size of the original data up to 75%. As a result the speed of data processing also increases. ORC shows better performance than Text, Sequence and RC file formats.



Source: http://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/

# ORCFile format (cont..)

- The *Optimized Row Columnar* (ORC) file format provides  a highly efficient way to store Hive data.
- `CREATE TABLE tablename (`

```
    ...
    ) STORED AS ORC;
    ALTER TABLE tablename SET FILEFORMAT  ORC;
    SET hive.default.fileformat=Orc
```

# ORCFile format (cont..)

- Using ORCFile or converting existing data to ORCFile is simple. To use it just add STORED AS orc to the end of your create table statements like this:

  ```
  CREATE TABLE mytable (
  ...
  ) STORED AS orc;
  ```

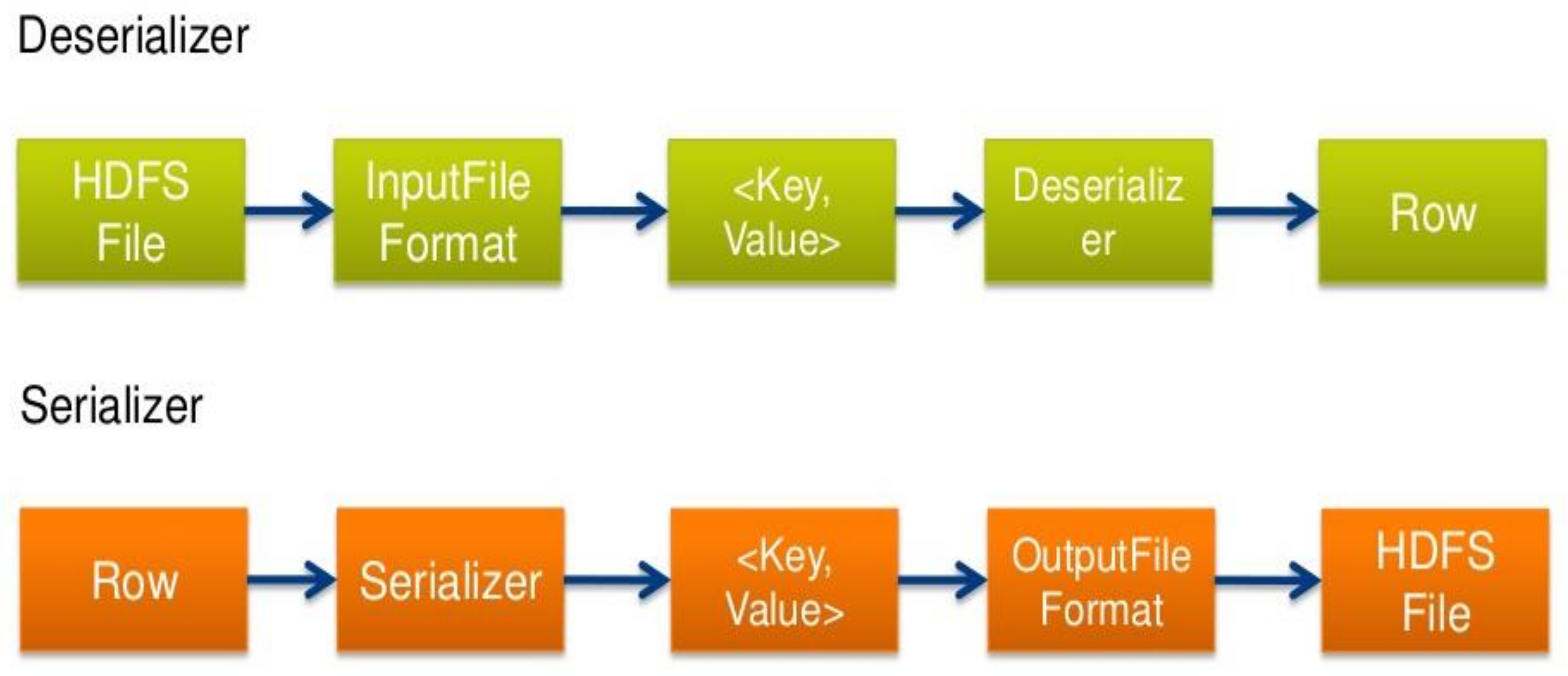
- To convert existing data to ORCFile create a table with the same schema as the source table plus stored as orc, then you can use issue a query like:

  Example:

  ```
  INSERT INTO TABLE orctable SELECT * FROM oldtable;
  ```

# Storage - SerDe file format

- SerDe stands for Serialize and De-serializer. Hive uses the SerDe interface for IO.



Deserializer

HDFS File → InputFile Format → <Key, Value> → Deserializer → Row

Serializer

Row → Serializer → <Key, Value> → OutputFile Format → HDFS File

# Q & A

What is SerD in Hive?

# Update, Delete, & INSERT in HIVE

# Update, Delete, & INSERT in HIVE (cont..)

Apache Hive provides a convenient SQL query engine and table abstraction for data stored in Hadoop. Hadoop is a batch processing system and Hadoop jobs tend to have high latency and incur substantial overheads in job submission and scheduling. Hive is not designed for online transaction processing and does not offer real-time queries and row level updates. Hive uses Hadoop to provide highly scaleable bandwidth to the data.

Up until Hive 0.13, atomicity, consistency, and durability were provided at the partition level and  did not support updates, deletes, or transaction isolation. This has prevented many desirable use cases, such as updating of dimension tables or doing data clean up.

But now Hive0.14 have implemented the standard SQL commands insert, update, and delete allowing users to insert new records as they become available, update changing dimension tables, repair incorrect data, and remove individual records. This also allows very low latency ingestion of streaming data from tools like Storm and Flume.

**Limitation :** BEGIN, COMMIT, and ROLLBACK are not yet supported.  All language operations are auto-commit.

# Update, Delete, & INSERT in HIVE (cont..)

- UPDATE is available starting in Hive 0.14.
- Updates can only be performed on tables that support ACID.
- Partitioning columns cannot be updated.
- Bucketing columns cannot be updated.
- Upon successful completion of this operation the changes will be auto-committed

**Standard Syntax:**

UPDATE tablename SET column = value [, column = value ...] [WHERE expression]

# Update, Delete, & INSERT in HIVE (cont..)

## HIVE DELETE

- DELETE is available starting in Hive 0.14.
- Deletes can only be performed on tables that support ACID. See Hive Transactions for details.
- Upon successful completion of this operation the changes will be auto-committed

**Standard Syntax:**

*DELETE FROM tablename [WHERE expression]*

# Update, Delete, & INSERT in HIVE

## HIVE INSERT

- The INSERT...VALUES statement can be used to insert data into tables directly from SQL.
- INSERT...VALUES is available starting in Hive 0.14.
- Inserting values from SQL statements can only be performed on tables that support ACID

**Syntax:**

*INSERT INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]*

*VALUES values_row [, values_row...]*

# Create Table As Select (CTAS)

- Tables can also be created and populated by the results of a query in one create-table-as-select (CTAS) statement.

- There are two parts in CTAS,
  - the SELECT part can be any SELECT statement supported by HiveQL.
  - The CREATE part of the CTAS takes the resulting schema from the SELECT part and creates the target table with other table properties such as the SerDe and storage format.

- CTAS has these restrictions:

  The target table cannot be a partitioned table.

  The target table cannot be an external table.

  The target table cannot be a list bucketing table.

# CTAS - Syntax

- CREATE TABLE new_key_value_store

  ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"

  STORED AS RCFile

  AS

  SELECT (key % 1024) new_key, concat(key, value) key_value_pair

  FROM key_value_store

  SORT BY new_key, key_value_pair;

# Difference between DELETE, TRUNCATE and DROP

- DELETE : Delete particular row with where condition.

- TRUNCATE: Truncate all the rows, which can not even be restored at all.

- DROP: it drops the table along with the data associated with Hive Meta store.
- If you had created a TABLE with EXTERNAL keyword then you **can NOT** remove all the rows because all data resides outside of Hive Meta store.

  **Syntax:**

  *hive> TRUNCATE TABLE tablename;*

**Syntax – Drop Table**

*hive>DROP TABLE [IF EXISTS] table_name [PURGE];*

# Module-3 Outline

- Hive's Join
- Hive Sub queries
- Using Distribute By & sort by
- The OVER Clause
- Partitioned Tables
- Bucketed Hive table
- Skewed Tables
- Understanding Views
- Defining Views
- Overview of Indexes

# Module Outline (cont…)

- Defining Indexes
- Using Windows
- Hive Analytics Function

# Hive Module - III

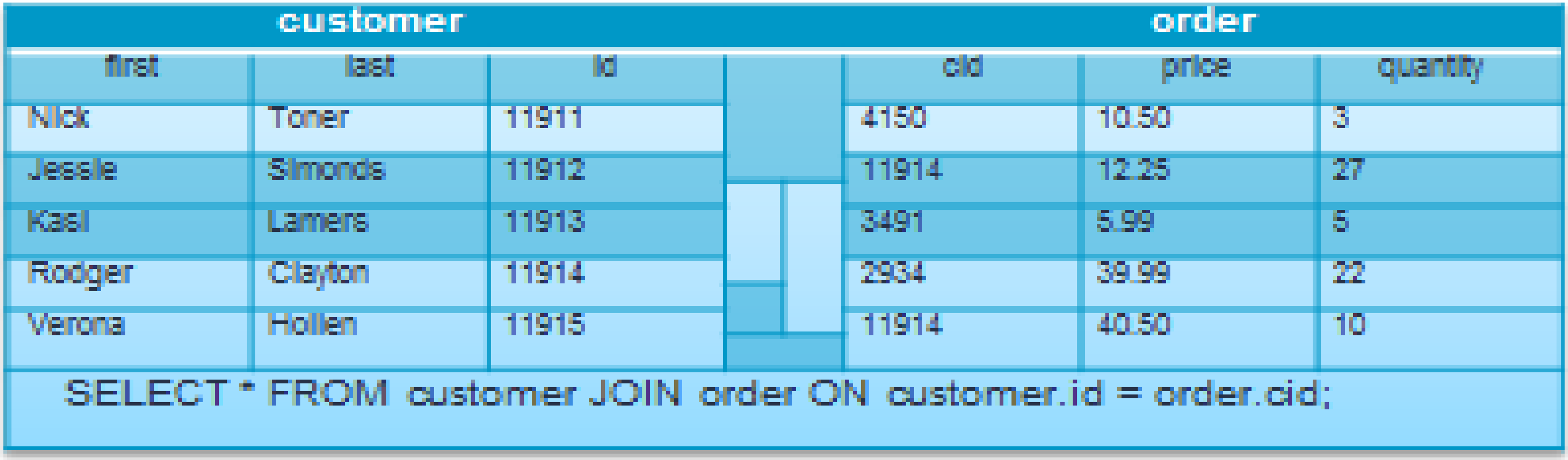
# Hive Join (cont..)

Hive uses three join strategies :

| Type | Approach | Pros | Cons |
|---|---|---|---|
| Shuffle Join | Join keys are shuffled using MapReduce and joins are performed on the reduce side. | Works regardless of data size or layout. | Most resourceG intensive and slowest join type. |
| Map (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 coGlocation of keys to do efficient joins. | Very fast for tables of any size. | Data must be sorted and bucketed ahead of time. |

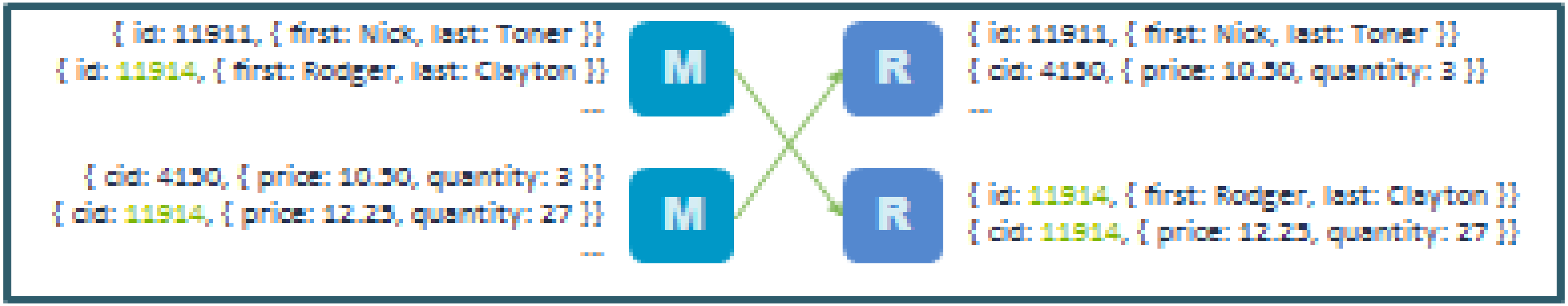
# Hive Join (cont..)

## Shuffle Joins



| 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 | | 3491 | 5.99 | 5 |
| Rodger | Clayton | 11914 | | 2934 | 39.99 | 22 |
| Verona | Hollen | 11915 | | 11914 | 40.50 | 10 |

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



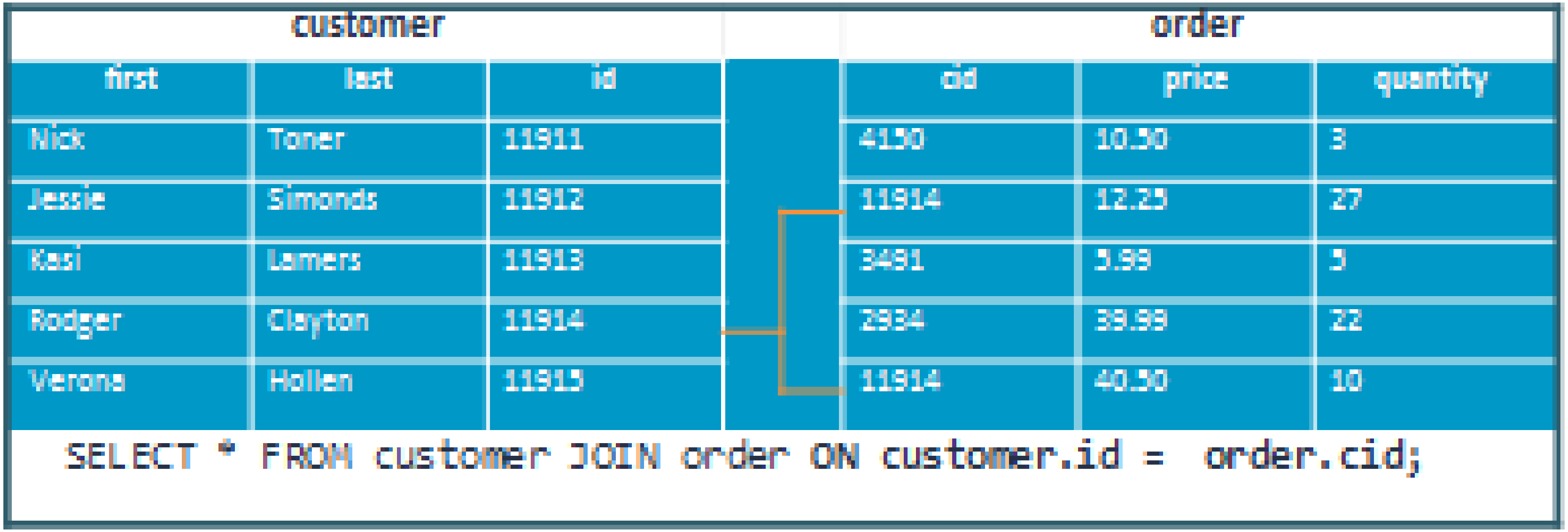
{ id: 11911, { first: Nick, last: Toner }}
{ id: 11914, { first: Rodger, last: Clayton }}
...

{ cid: 4150, { price: 10.50, quantity: 3 }}
{ cid: 11914, { price: 12.25, quantity: 27 }}
...

{ id: 11911, { first: Nick, last: Toner }}
{ cid: 4150, { price: 10.50, quantity: 3 }}
...

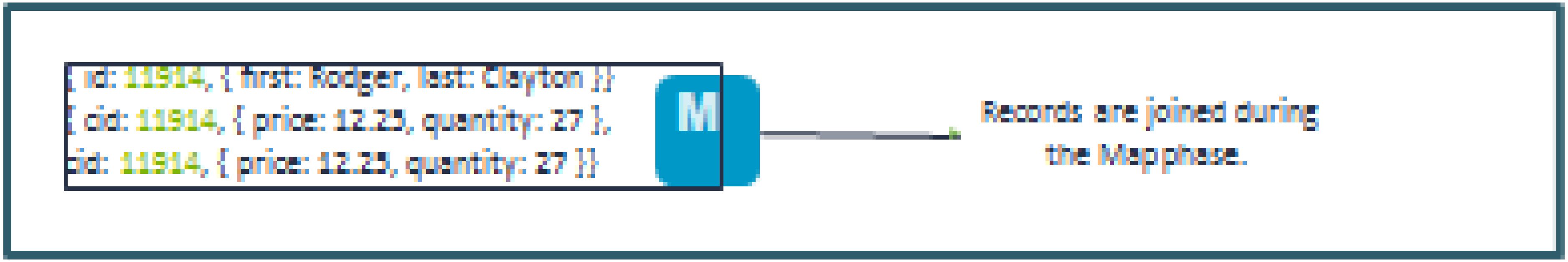
{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 }}

# Hive Join (cont..)



| 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 | 3491 | 5.99 | 5 |
| Rodger | Clayton | 11914 | 2934 | 39.99 | 22 |
| Verona | Hollen | 11915 | 11914 | 40.50 | 10 |

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

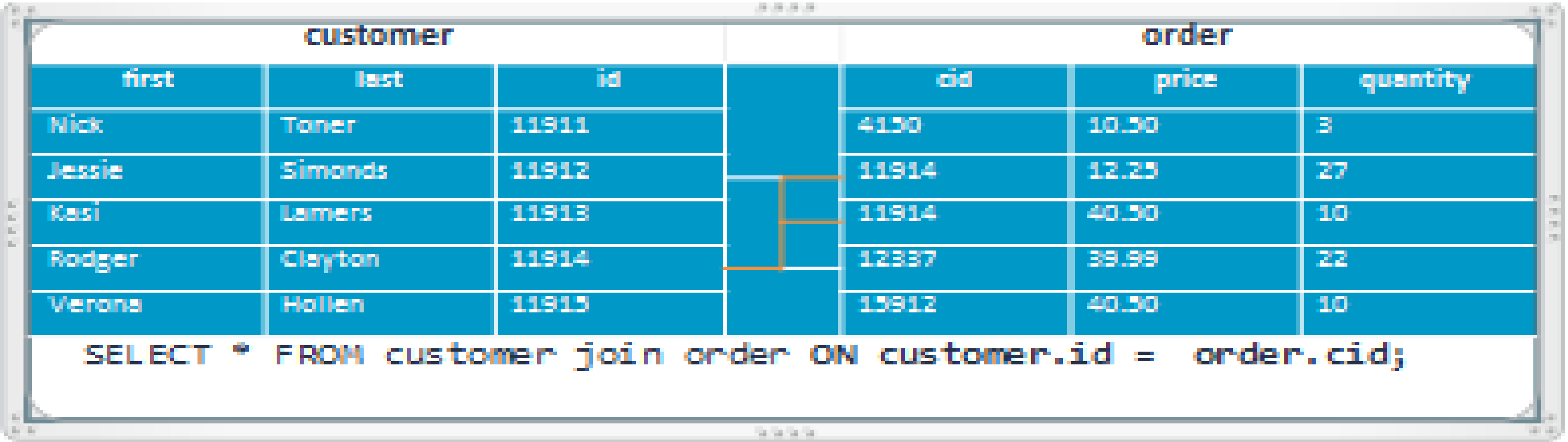


```
{ id: 11914, { first: Rodger, last: Clayton }}
{ cid: 11914, { price: 12.25, quantity: 27 },
cid: 11914, { price: 12.25, quantity: 27 }}
```

Records are joined during the Map phase.

# Hive Join (cont..)

## Sort-Merge-Bucket Joins



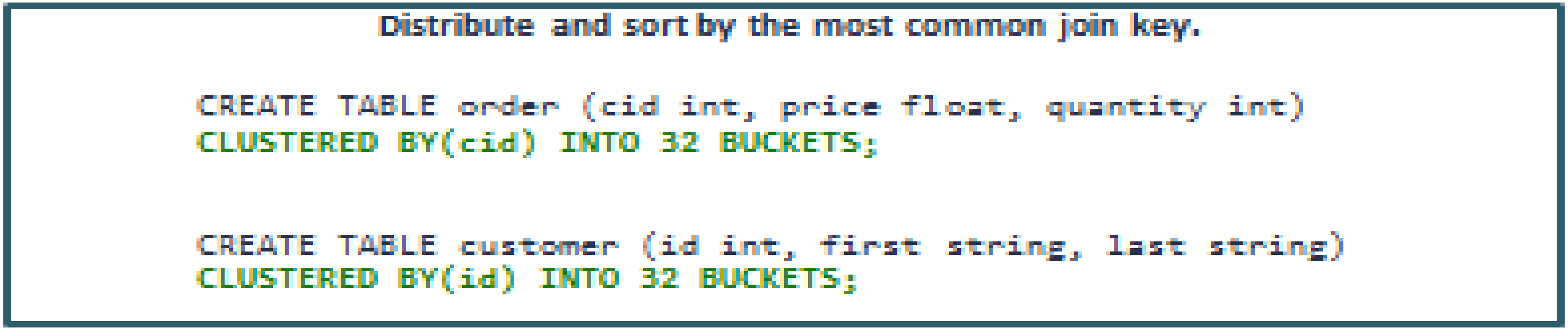
customer

| 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 |
| 11914 | 40.50 | 10 |
| 12337 | 39.99 | 22 |
| 15912 | 40.50 | 10 |

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

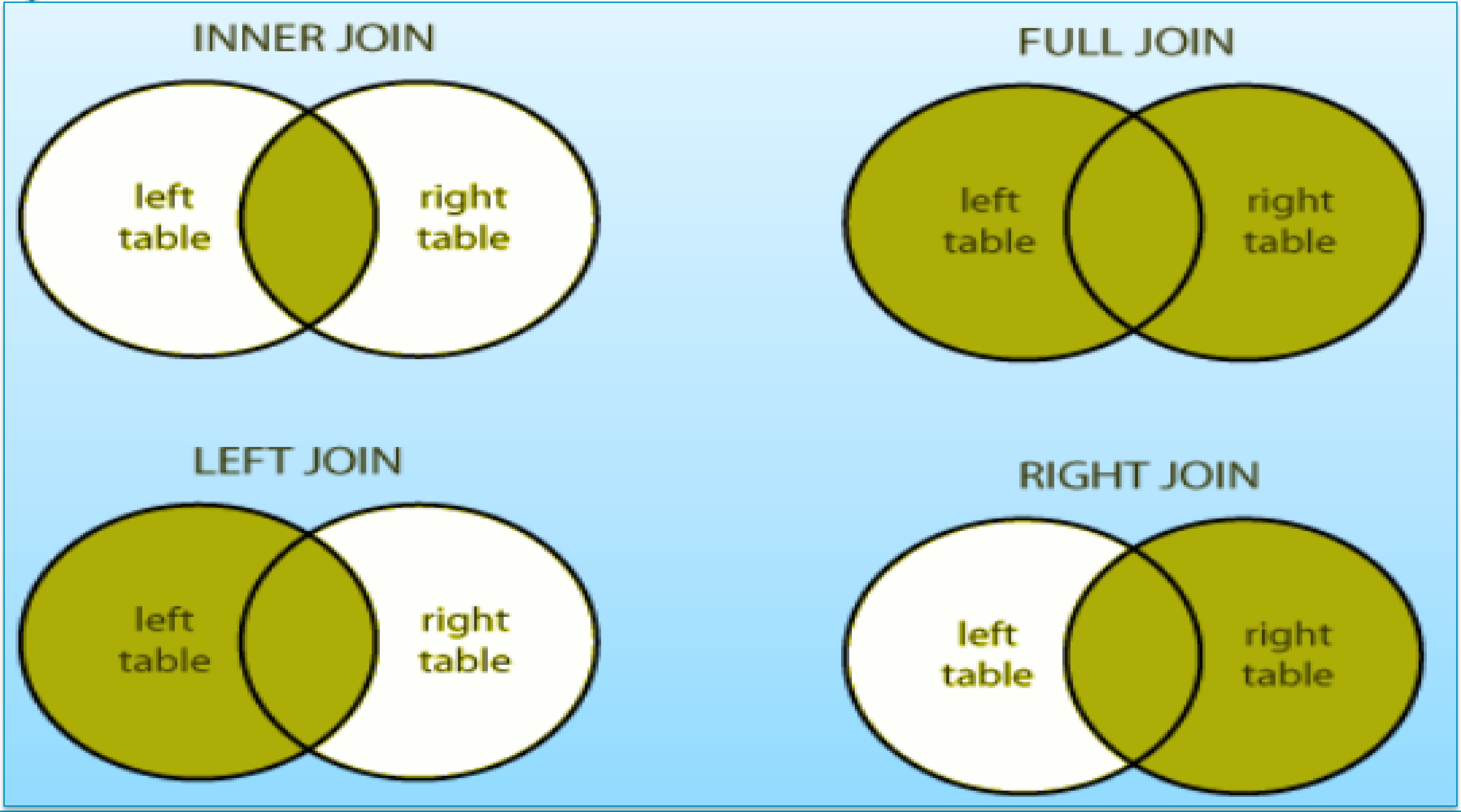


**Distribute and sort by the most common join key.**

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


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

# Hive Join


INNER JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

# Hive Join (cont..)

- JOIN clause is used to combine and retrieve the records from multiple tables.
- JOIN is same as OUTER JOIN in SQL.
- A JOIN condition is to be raised using the primary keys and foreign keys of the tables
- There are following types of Join in Hive:
  - INNER JOIN .
  - LEFT JOIN (LEFT OUTER JOIN)
  - RIGHT JOIN (RIGHT OUTER JOIN)
  - FULL JOIN (FULL OUTER JOIN)

# Hive Join (cont..)

- **INNER JOIN** – Select records that have matching values in both tables.

  *Example:*

  *hive> select c.id, c.name, o.order_date, o.amount from customers c inner join orders o ON (c.id = o.customer_id);*

- **LEFT JOIN** (LEFT OUTER JOIN) – returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching join predicate

  *Example:*

  *hive> select c.id, c.name, o.order_date, o.amount from customers c left outer join*

  *orders o  ON (c.id = o.customer_id);*

# Hive Join (cont..)

- **RIGHT JOIN** (RIGHT OUTER JOIN) A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate

  *Example:*

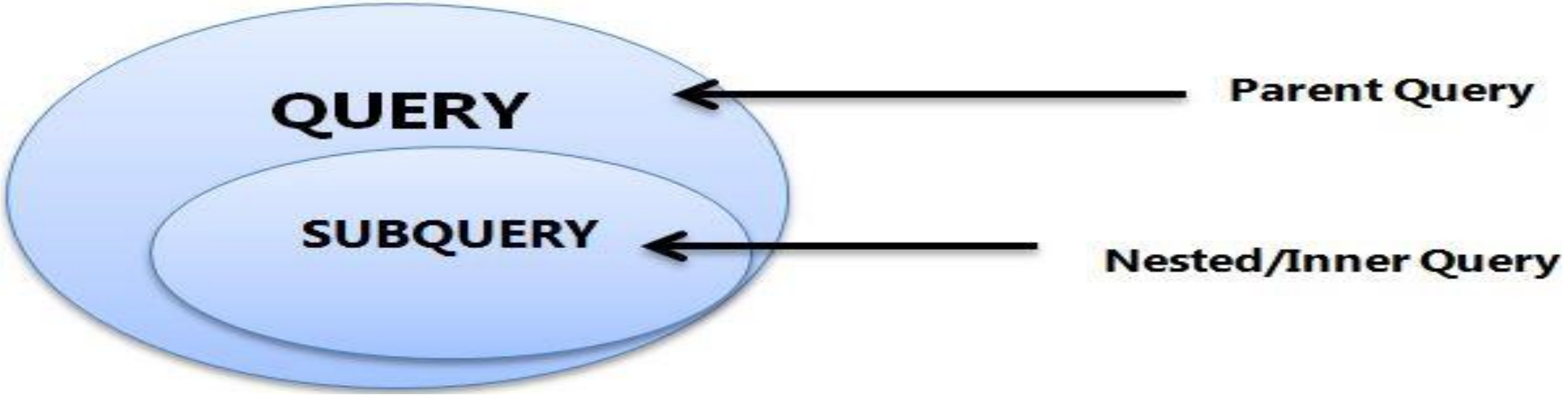
  *hive> select c.id, c.name, o.order_date, o.amount from customers c left outer join orders o ON (c.id = o.customer_id);*

- **FULL JOIN** (FULL OUTER JOIN) – Selects all records that match either left or right table records.

  *Example:*

  *hive> select c.id, c.name, o.order_date, o.amount from customers c full outer join orders o ON (c.id = o.customer_id);*

# Hive - Sub queries



- A Subquery or Inner query or Nested query is a query within another HQL query and embedded within the WHERE clause.

- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
  - Subqueries in the FROM Clause
  - Subqueries in the WHERE Clause

# Hive - Sub queries (cont..)

Subqueries in the WHERE Clause

- Hive supports Subqueries only in the FROM clause through Hive version 0.12

  Example :

  SELECT a.empid,a.ename, b.deptno FROM (SELECT * FROM emp) a JOIN (SELECT * FROM dept) b ON a.deptno = b.deptno;

Subqueries in the WHERE Clause

- As of Hive 0.13 some types of Subqueries are supported in the WHERE clause

  Example:

  SELECT state, net_payments FROM transfer_payments WHERE

  transfer_payments.year IN (SELECT year FROM us_census);

# Sorting Data

- Hive has two sorting clauses:
  - **sort by**: Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer and data output is sorted per reducer
  - **order by**: a complete ordering of the data and It orders data at each of ‘N’ reducers. ORDER BY guarantees total ordering of data, but for that it has to be passed on to a single reducer, which is normally unacceptable and therefore in strict mode, hive makes it compulsory to use LIMIT with ORDER BY so that reducer doesn’t get overburdened.

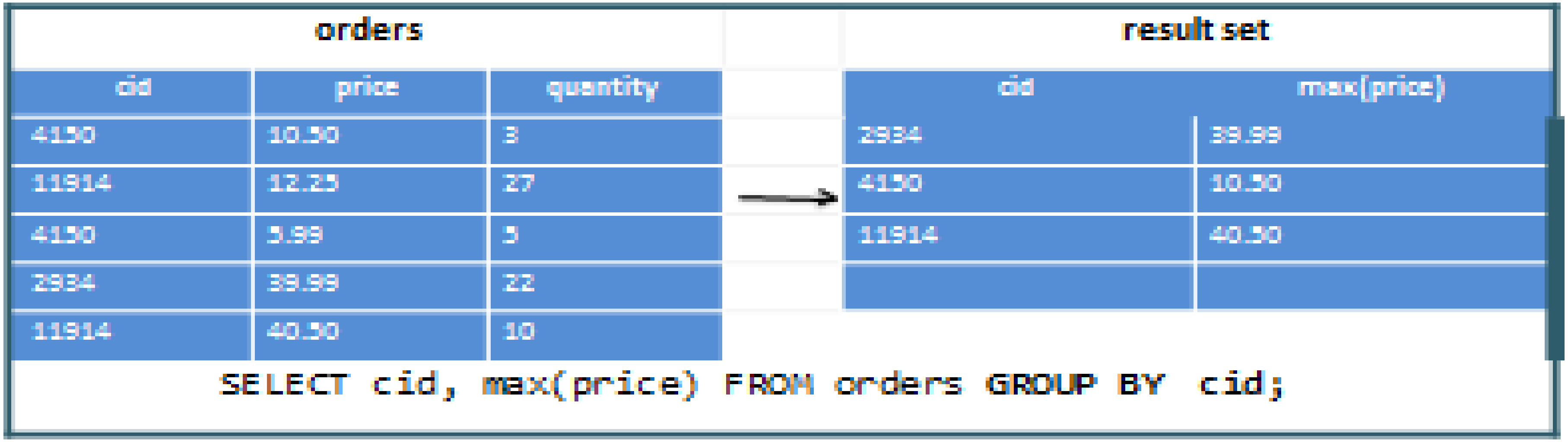
# Using Distribute By & sort by

Hive uses the columns in Distribute By to distribute the rows among reducers. All rows with the same Distribute By columns will go to the same reducer.

Cluster By is a short-cut for both Distribute By and Sort By.
CLUSTER BY x ensures each of N reducers gets non-overlapping ranges, then sorts by those ranges at the reducers

- SELECT * from dataset  select * distribute by age;
- SELECT * from dataset  select * distribute by age sort  by age;
- SELECT * from dataset  select * cluster by age;

# The OVER Clause



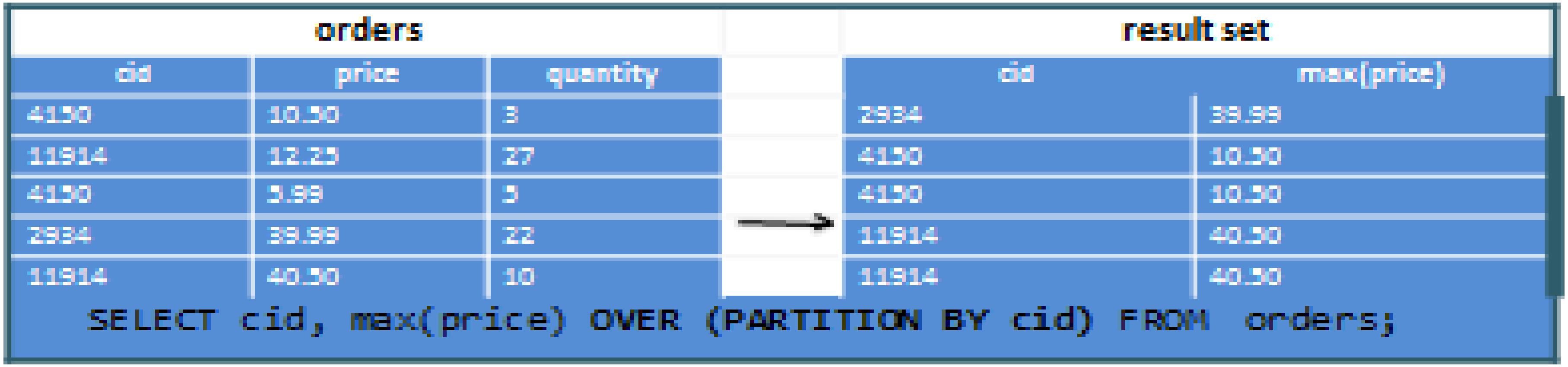
orders

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

result set

| cid | max(price) |
|---|---|
| 2934 | 39.99 |
| 4150 | 10.50 |
| 11914 | 40.50 |
| | |

```
SELECT cid, max(price) FROM orders GROUP BY  cid;
```



orders

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

result set

| cid | max(price) |
|---|---|
| 2934 | 39.99 |
| 4150 | 10.50 |
| 4150 | 10.50 |
| 11914 | 40.50 |
| 11914 | 40.50 |

```
SELECT cid, max(price) OVER (PARTITION BY cid) FROM  orders;
```

# Partitioned Tables

- Partitioned tables can be created using the PARTITIONED BY clause. A table can have one or more partition columns and a separate data directory is created for each distinct

- Partitioning works both managed and external tables.

Usually Partitioning in Hive offers a way of segregating hive table data into multiple files/directories. But partitioning gives effective results when,

- There are limited number of partitions
- Comparatively equal sized partitions

# Bucketed Hive table

- Hive Bucketing is nothing but another technique of decomposing data or decreasing the data into more manageable parts or equal parts

- The Hive Partition can be further subdivided into Clusters or Buckets

- Bucketing can be done along with Partitioning on Hive tables and even without partitioning

- In Hive Partition we used PARTITIONED BY but in Hive Buckets we used CLUSTERED BY

- Records with the same bucketed column will always be stored in the same bucket

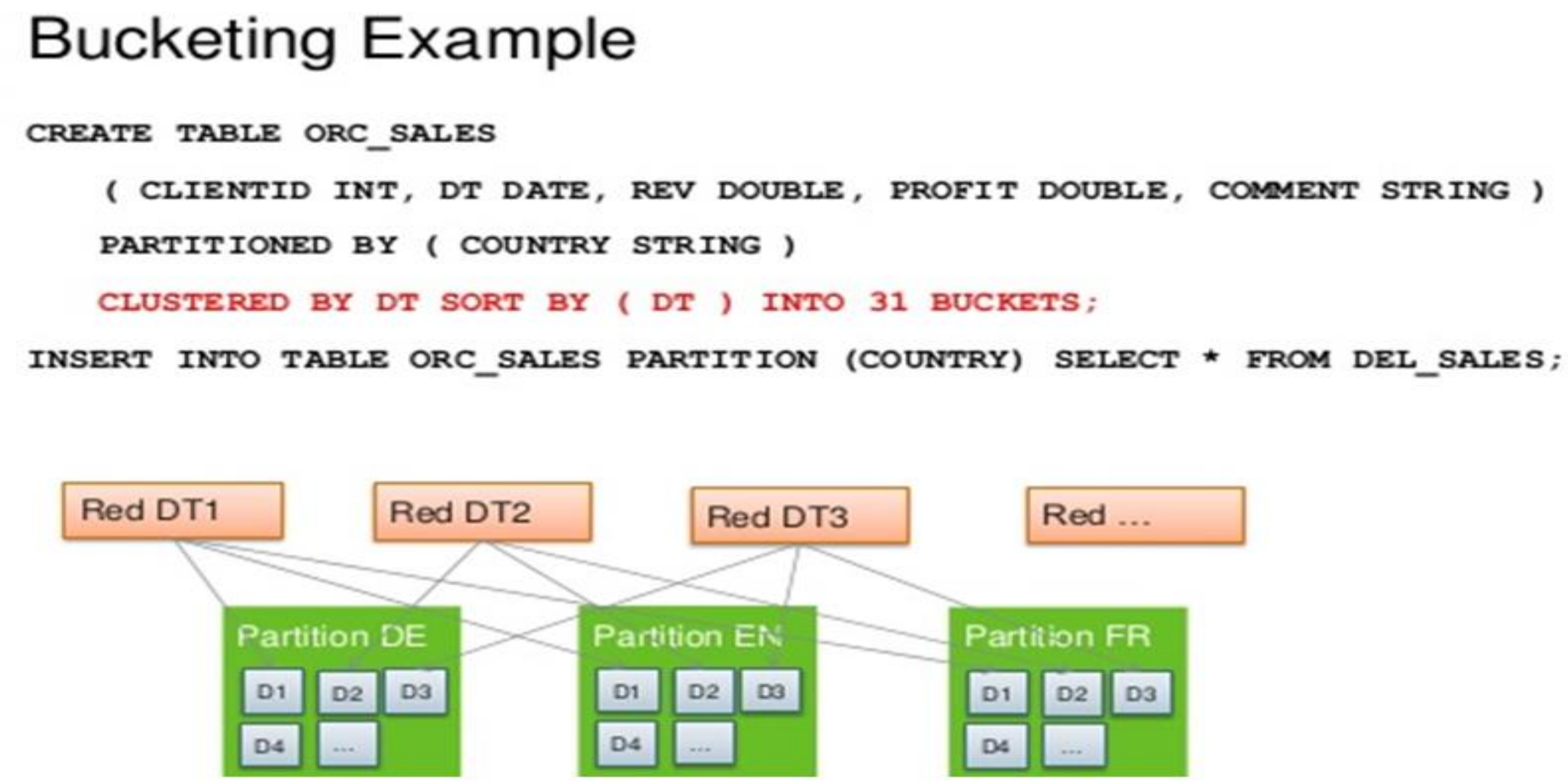
- Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based

  and bucketed tables will create almost equally distributed data file parts

**Advantages**

Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge  As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables

# Bucketed Hive table (cont..)



## Bucketing Example

```
CREATE TABLE ORC_SALES
    ( CLIENTID INT, DT DATE, REV DOUBLE, PROFIT DOUBLE, COMMENT STRING )
    PARTITIONED BY ( COUNTRY STRING )
    CLUSTERED BY DT SORT BY ( DT ) INTO 31 BUCKETS;
INSERT INTO TABLE ORC_SALES PARTITION (COUNTRY) SELECT * FROM DEL_SALES;
```

# Q & A

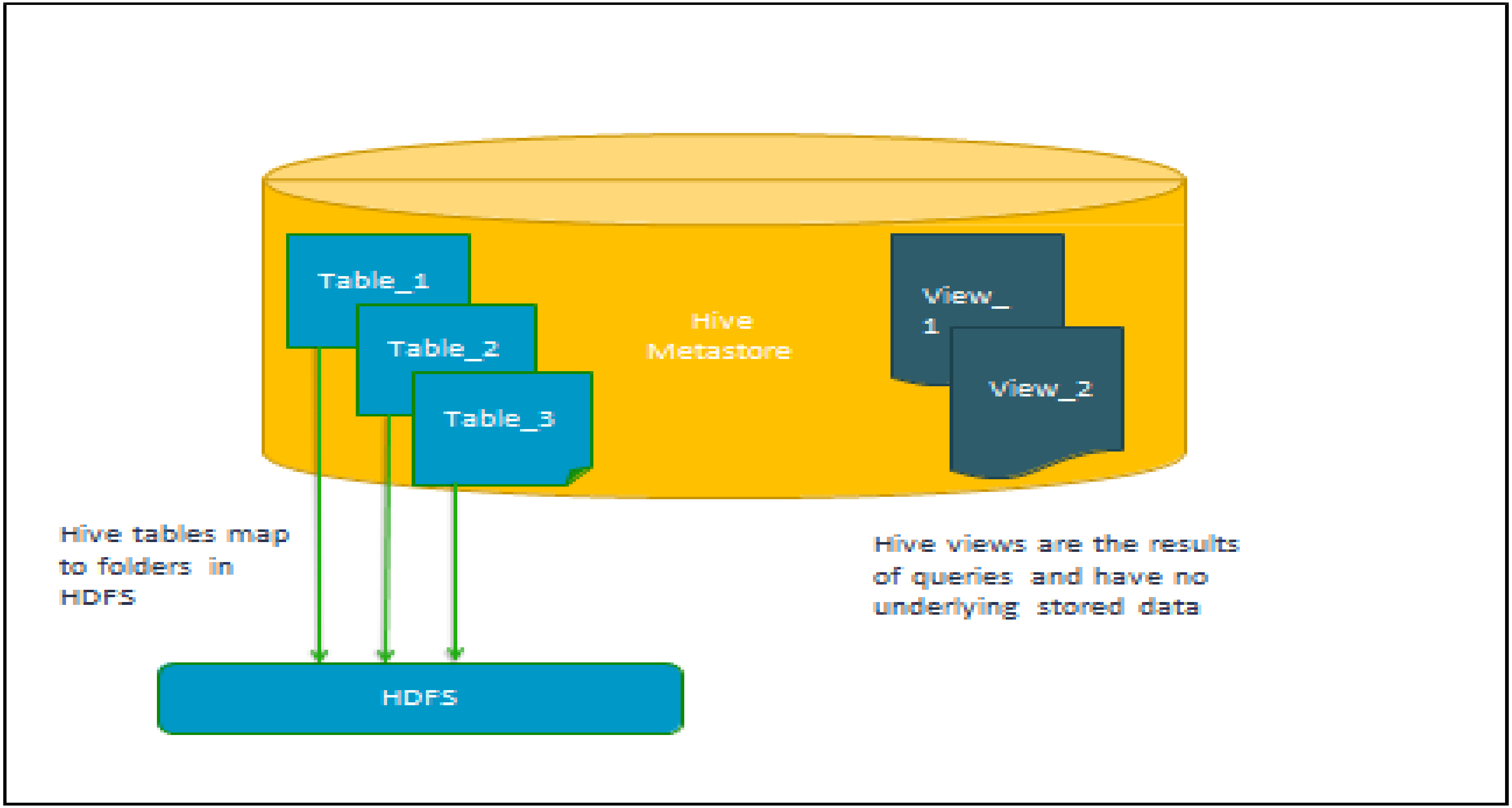
What is difference between bucketing and partitioning?

# Skewed Tables

- In Hive, ***skew*** refers to one or more columns in a table that have values that appear very often

```
CREATE TABLE Customers (  id int,
 username string, zip int
 SKEWED BY (zip) ON (57701, 57702)  STORED as
 DIRECTORIES;
```

# Understanding Views



Hive tables map to folders in HDFS

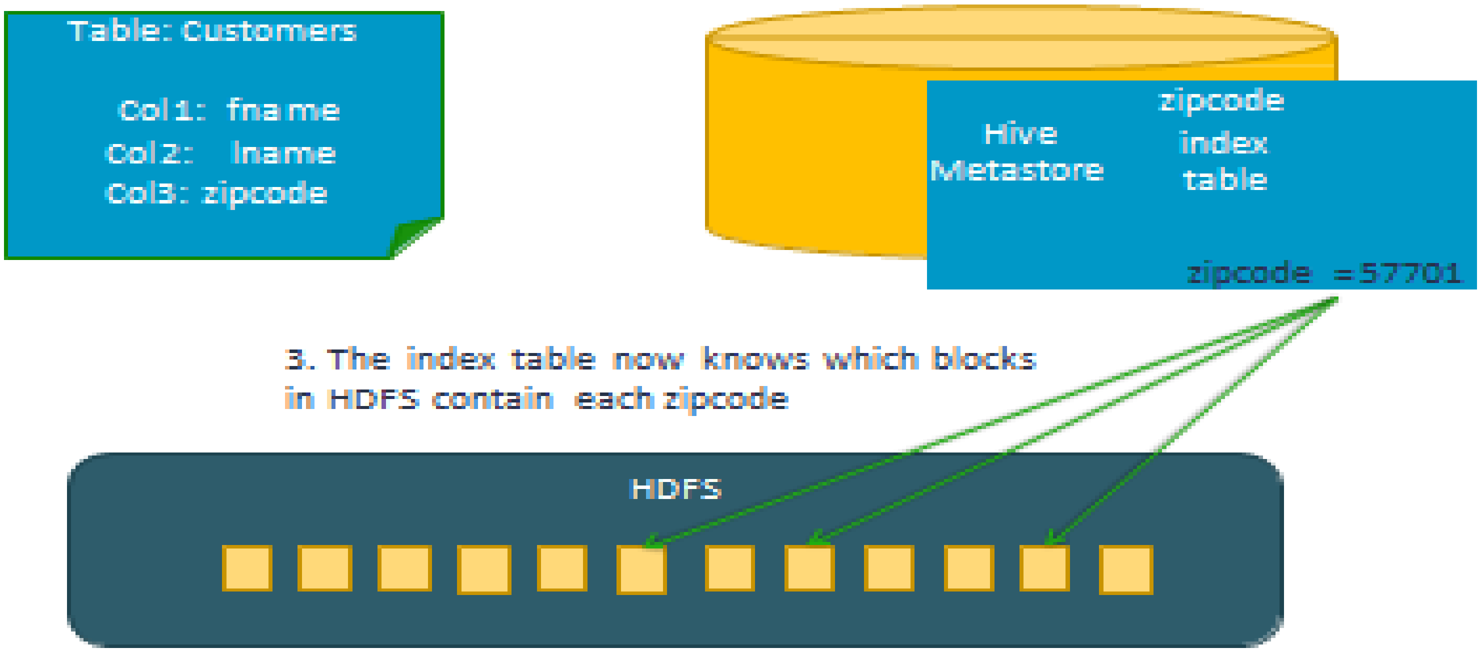
Hive views are the results of queries and have no underlying stored data

# Defining Views

- *CREATE VIEW 2010_visitors AS  SELECT fname, lname, time_of_arrival, info_comment  FROM wh_visits WHERE cast(substring(time_of_arrival,6,4)  AS int) >= 2010 AND cast(substring(time_of_arrival,6,4)  AS int) < 2011;*

# Overview of Indexes

- Indexes are used to speed the search of data within tables
- Hive indexes are limited to managed tables and cannot be used with external tables
- Indexes can be partitioned (matching the partitions of the base table)
- Indexes in Hive are not recommended



1. An index is defined on the zipcode column of a table named Customers.

2. An index table is created in the Hive metastore for the zipcode column

3. The index table now knows which blocks in HDFS contain each zipcode

# Defining Indexes

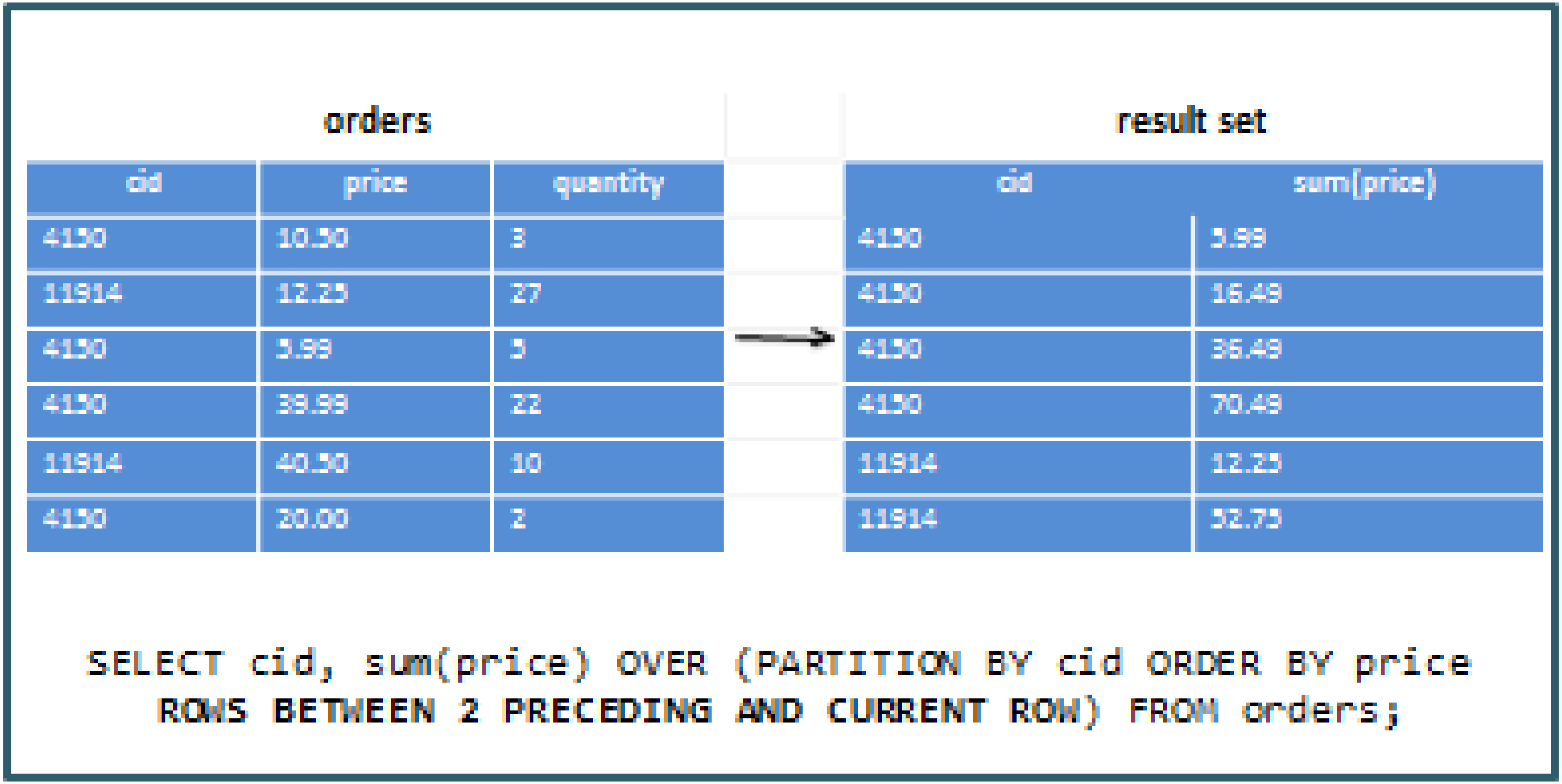
```
CREATE INDEX zipcode_index ON TABLE
Customers (zipcode) AS 'COMPACT' WITH
DEFERRED REBUILD;
```

```
ALTER INDEX zipcode_index ON
Customers REBUILD;
```

```
SHOW INDEX ON Customers;
```

```
DROP INDEX zipcode_index ON
Customers;
```

# Using Windows



**orders**

| cid | price | quantity |
|---|---|---|
| 4150 | 10.50 | 3 |
| 11914 | 12.25 | 27 |
| 4150 | 5.99 | 5 |
| 4150 | 39.99 | 22 |
| 11914 | 40.50 | 10 |
| 4150 | 20.00 | 2 |

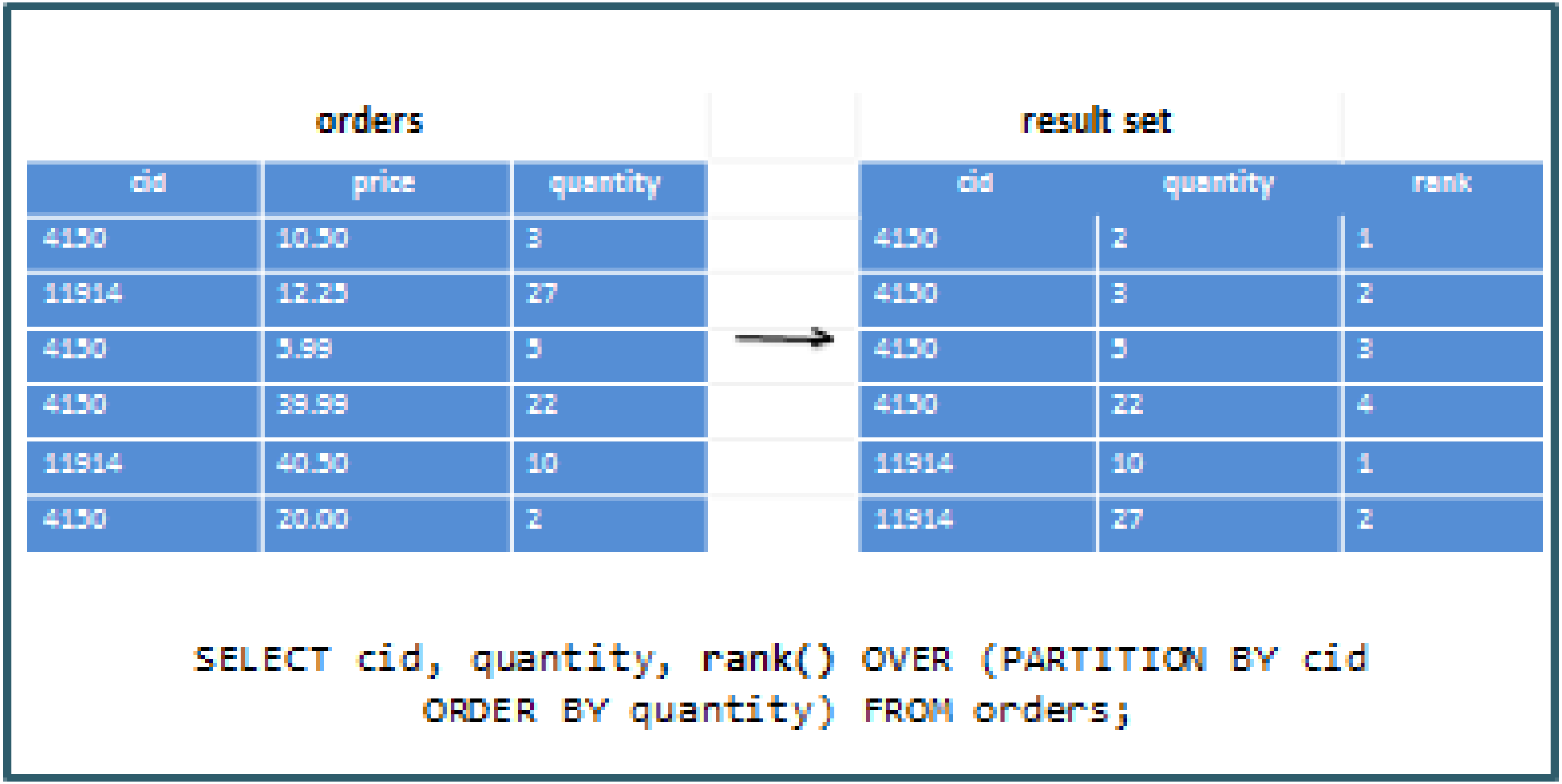
→

**result set**

| cid | sum(price) |
|---|---|
| 4150 | 5.99 |
| 4150 | 16.49 |
| 4150 | 36.49 |
| 4150 | 70.49 |
| 11914 | 12.25 |
| 11914 | 52.75 |

```
SELECT cid, sum(price) OVER (PARTITION BY cid ORDER BY price
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) FROM orders;
```

# Hive Analytics Function



orders

| cid | price | quantity |
|---|---|---|
| 4150 | 10.50 | 3 |
| 11914 | 12.25 | 27 |
| 4150 | 5.99 | 5 |
| 4150 | 39.99 | 22 |
| 11914 | 40.50 | 10 |
| 4150 | 20.00 | 2 |

result set

| cid | quantity | rank |
|---|---|---|
| 4150 | 2 | 1 |
| 4150 | 3 | 2 |
| 4150 | 5 | 3 |
| 4150 | 22 | 4 |
| 11914 | 10 | 1 |
| 11914 | 27 | 2 |

```
SELECT cid, quantity, rank() OVER (PARTITION BY cid
ORDER BY quantity) FROM orders;
```

# Module-4 Outline

- Hive Optimization
- Hive query using Tez Execution Engine
- Run a Hive query using vectorization
- Run a Hive query using CBO
- User-Defined Functions (UDFs)

# Apache Hive - Optimization

Hive on Hadoop makes data processing so straightforward and scalable but

Well designed tables and queries can greatly improve speed and reduce processing cost.

There are broadly three areas in which we can optimize Hive utilization are:

- Data Layout (Partitions and Buckets)
- Data Sampling (Bucket and Block sampling)
- Data Processing (Bucket Map Join and Parallel execution)

# Hive Optimization

Below are some helpful design tips for improving the speed of Hive queries.

- Divide data amongst different files which can be pruned out by using partitions, buckets and skews
- Sort data ahead of time to simplify joins
- Use the ORC file format
- Sort and Bucket on common join keys
- Use map (broadcast) joins whenever possible
- Increase the replication factor for hot data (which reduces latency)
- Use CBO
- Use Vectorization
- Take advantage of Tez

# Hive Optimization (cont..)

- Table Partitions ：Cloudera recommends keeping table partitions below two or three thousand for optimal performance.

  When a hive query has to reference more than a few thousand partitions, performance can suffer. Multiple queries must be run against the Hive Metastore database to retrieve and update these partitions and HDFS must move these files around.

  For the best performance, design your tables to partition on fewer columns or to have less granular time frames, for example by day instead of hourly. Also, hone your queries to use only a subset of a table's partitions

**Note :** Hive has a special file called the **.hiverc** file that gets executed each time you launch a Hive shell. This makes the **.hiverc** file a great place for adding custom configuration settings that you use all the time, or loading JAR files that contain frequently--used UDFs. The file is saved in the Hive conf directory, which is **/etc/hive/conf** for an HDP installation.

# Hive Optimization (cont..)

- mapreduce.input.fileinpu@ormat.split.maxsize
- mapreduce.input.fileinpu@ormat.split.minsize
- mapreduce.tasks.io.sort.mb
- In addition, set the important join and bucket properties to true in hive)site.xml or by using the set command.

# Hive Optimization (cont..)

## Hive Query Tunings

Hive has a lot of parameters that can be set globally in **hive--site.xml** or at the script level using the **set** command. Here are some of the more important parameters to improve the performance of your Hive queries:

- **mapreduce.input.fileinputformat.split.maxsize** and **mapreduce.input.fileinputformat.split.minsize**: if the min is too large, you will have too few mappers. If the max is too small, you will have too many mappers.

- **mapreduce.tasks.io.sort.mb**: increase this value to avoid disk spills. Set

the following properties all the time:

# Hive query using Tez Execution Engine

# Hive query using Tez Execution Engine (cont..)

- Hive can use the Apache Tez execution engine instead of the venerable Map-reduce engine

- it’s not turned on by default in your environment

- To Enable Tez model use :

```
hive > set hive.execution.engine=tez;
```

**Limitation of Tez**

- Tez does not support the following actions:
  - Index creation
  - Skew joins

# Hive on Spark

- Hive traditionally uses MapReduce behind the scenes to parallelize the work, and perform the low-level steps of processing a SQL statement such as sorting and filtering. Hive can also use Spark as the underlying computation and parallelization engine
- Hive on Spark provides better performance than Hive on MapReduce while offering the same features. Running Hive on Spark requires no changes to user queries
- Hive user-defined functions (UDFs) are fully supported, and most performance-related configurations work with the same semantics

To configure Hive to run on Spark do both of the following steps:

- Configure the Hive client to use the Spark execution engine

  ```
  set hive.execution.engine=spark;
  ```

- Identify the Spark service that Hive uses. Cloudera Manager automatically sets this to the configured MapReduce

# Run a Hive query using vectorization

## Hive Vectorization

- Vectorization allows Hive to process a batch of rows together as a unit instead of processing one row at a time. Each batch consists of a column vector which is usually an array of primitive types. Operations are performed on the entire column vector, which improves the instruction pipelines and cache usage.

- It perform in batches of 1024 rows at once instead of single row each time.

- Vectorized query execution improves performance of operations like scans, aggregations, filters and joins.

- Introduced in Hive 0.13, this feature significantly improves query execution time, and is easily enabled with two parameters settings:

```
set hive.vectorized.execution.enabled = true;
set hive.vectorized.execution.reduce.enabled = true;
```

# Run a Hive query using CBO

- Cost-based optimization (CBO), performs optimizations based on query cost.
- To use CBO set the following parameters at the beginning of your query:

```
hive>set hive.cbo.enable=true;
hive>set hive.compute.query.using.stats=true;
hive>set hive.stats.fetch.column.stats=true;
hive>set hive.stats.fetch.partition.stats=true;
```

# Creating User-Defined Functions

- Hive supports manipulation of data via user-created functions
- Example:

```
INSERT OVERWRITE TABLE u_data_new
SELECT TRANSFORM (userid, movieid, rating, unixtime)
USING 'python weekday_mapper.py‘ AS (userid, movieid, rating, weekday)
FROM u_data;
```

# Hive Use Cases

Hive is a petabyte-scale data warehouse system built on the Hadoop platform, it is a good choice for environments experiencing phenomenal growth in data volume. The underlying MapReduce interface with HDFS is hard to program directly, but Hive provides an SQL interface, making it possible to use existing programming skills to perform data preparation.

Hive on MapReduce or Spark is best-suited for batch data preparation or ETL:

- You must run scheduled batch jobs with very large ETL sorts with joins to prepare data for Hadoop. Most data served to BI users in Impala is prepared by ETL developers using Hive
- You run data transfer or conversion jobs that take many hours. With Hive, if a problem occurs partway through such a job, it recovers and continues.
- You receive or provide data in diverse formats, where the Hive SerDes and variety of UDFs make it convenient to ingest and convert the data. Typically, the final stage of the ETL process with Hive might be to a high-performance, widely supported format such as Parquet.

# Q & A

# Thank You

|  |  |
|---|---|
| **Review Date** | **25-May-2016** |
| **Version** | **1.0** |
| **Next Review Due** | **15-July-2016** |