Hive Programming

Topics Covered

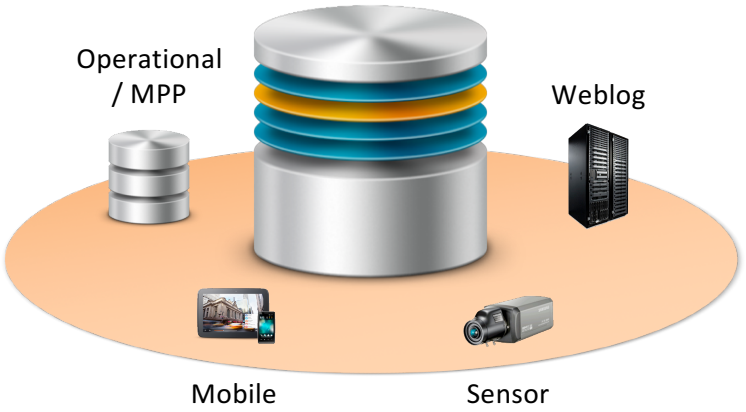
- About Hive
- Comparing Hive to SQL
- Hive Architecture
- Submitting Hive Queries
- Defining Tables
- Loading Data into Hive
- Performing Queries
- Hive Partitions, Buckets, and Skewed
- Sorting Data
- Hive Join Strategies

- About Hive

Objectives

About Hive

Store and Query all Data in Hive



Use Existing SQL Tools and Existing SQL Processes

About Hive – cont.

- It is a data warehouse system for Hadoop
- It maintains metadata information about your big data stored on HDFS
- It treats your big data as tables
- It performs SQL-like operations on the data using a scripting language called **HiveQL**

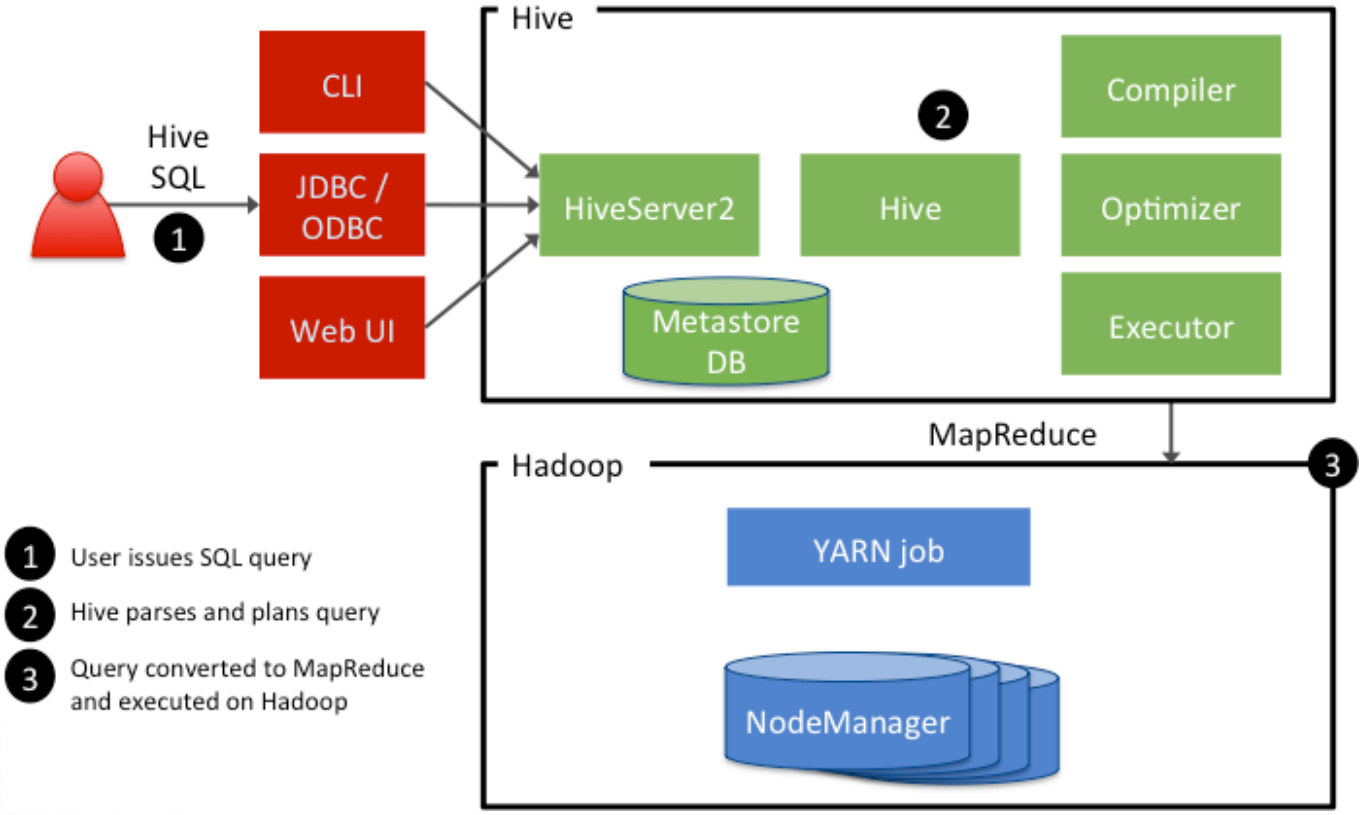
Objectives

- About Hive
- Comparing Hive to SQL

Hive's Alignment with SQL

SQL Datatypes	SQL Semantics
INT	SELECT, LOAD, INSERT from query
TINYINT/SMALLINT/BIGINT	Expressions in WHERE and HAVING
BOOLEAN	GROUP BY, ORDER BY, SORT BY
FLOAT	CLUSTER BY, DISTRIBUTE BY
DOUBLE	Sub-queries in FROM clause
STRING	GROUP BY, ORDER BY
BINARY	ROLLUP and CUBE
TIMESTAMP	UNION
ARRAY, MAP, STRUCT, UNION	LEFT, RIGHT and FULL INNER/OUTER JOIN
DECIMAL	CROSS JOIN, LEFT SEMI JOIN
CHAR	Windowing functions (OVER, RANK, etc.)
VARCHAR	Sub-queries for IN/NOT IN, HAVING
DATE	EXISTS / NOT EXISTS

Hive Architecture



Objectives

- About Hive
- Comparing Hive to SQL
- Hive Queries

Submitting Hive Queries

- Hive CLI
  - Traditional Hive “thick” client
  - ```
    $ hive
    hive>
    ```
- Beeline
  - A new command-line client that connects to a HiveServer2 instance
  - ```
    $ beeline -u url -n username -p password
    beeline>
    ```

Defining a Hive-Managed Table

```
CREATE TABLE customer (
          customerID INT,
          firstName STRING,
          lastName STRING,
          birthday TIMESTAMP
  ) ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ',';
```

Defining an External Table

```
CREATE EXTERNAL TABLE salaries (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ',';
```

Defining a Table LOCATION

```
CREATE EXTERNAL TABLE SALARIES (
   gender string,
   age int,
   salary double,
   zip int
 ) ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LOCATION '/user/train/salaries/';
```

Loading Data into Hive

```
LOAD DATA LOCAL INPATH ’/tmp/customers.csv'
OVERWRITE INTO TABLE customers;
```

```
LOAD DATA INPATH '/user/train/customers.csv'
OVERWRITE INTO TABLE customers;
```

```
INSERT INTO TABLE birthdays
    SELECT firstName, lastName, birthday
    FROM customers
    WHERE birthday IS NOT NULL;
```

Performing Queries

```
SELECT * FROM customers;
```

```
FROM customers
    SELECT firstName, lastName, address, zip
    WHERE orderID > 0
    ORDER BY zip;
```

```
SELECT customers.*, orders.*
    FROM customers
    JOIN orders ON
    (customers.customerID = orders.customerID);
```

Objectives

- About Hive
- Comparing Hive to SQL
- Hive Queries
- Hive Partitions, Buckets, and Skewed

Hive Partitions

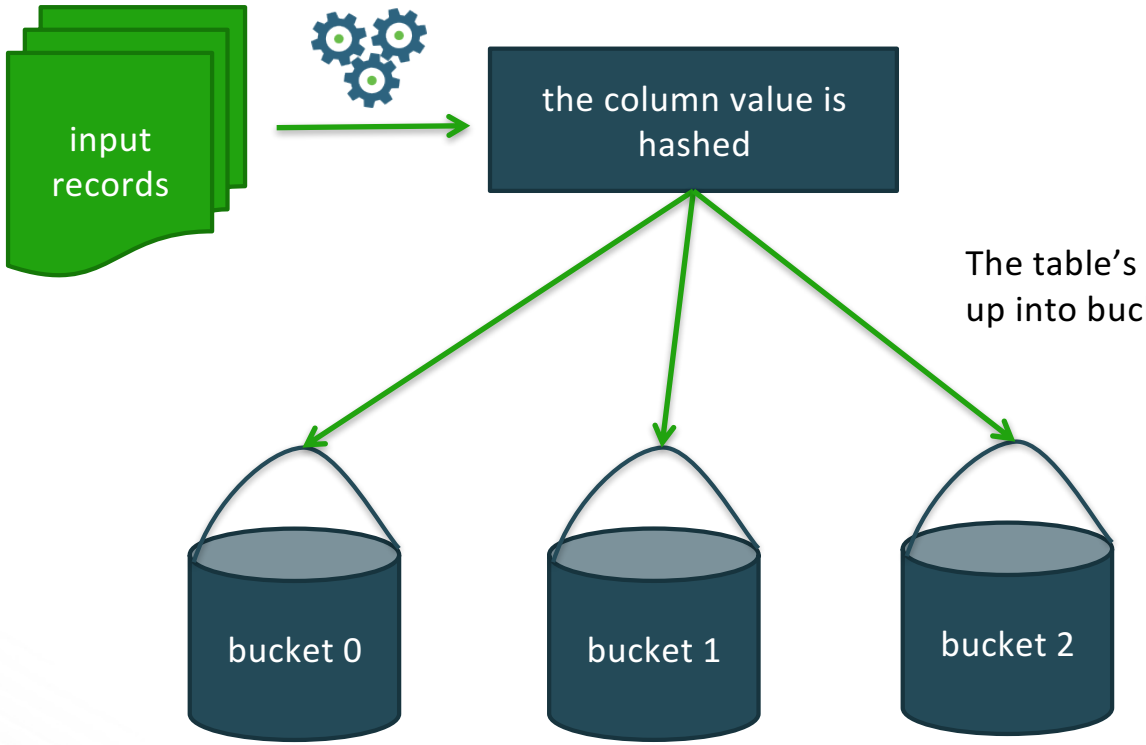
- Use the **partitioned by** clause to define a partition when creating a table:

```
create table employees (id int, name string, salary double)
partitioned by (dept string);
```

- Subfolders are created based on the partition values:

```
/apps/hive/warehouse/employees
    /dept=hr/
    /dept=support/
    /dept=engineering/
    /dept=training/
```

Hive Buckets



The table's data is divided up into buckets

Skewed Tables

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

Objectives

- About Hive
- Comparing Hive to SQL
- Hive Queries
- Hive Partitions, Buckets, and Skewed
- Sorting Data

Sorting Data

Hive has two sorting clauses:

- **order by**: a complete ordering of the data
- **sort by**: data output is sorted per reducer

Using Distribute By

```
insert overwrite table mytable
    select gender,age,salary
    from salaries
    distribute by age;
```

```
insert overwrite table mytable
    select gender,age,salary
    from salaries
    distribute by age
    sort by age;
```

Storing Results to a File

```
INSERT OVERWRITE DIRECTORY
  '/user/train/ca_or_sd/'
  from names
    select name, state
    where state = 'CA'
    or state = 'SD';
```

```
INSERT OVERWRITE LOCAL DIRECTORY
  '/tmp/myresults/'
  SELECT * FROM bucketnames
  ORDER BY age;
```

Specifying MapReduce Properties

```
SET mapreduce.job.reduces = 12
```

```
hive -f myscript.hive
 -hiveconf mapreduce.job.reduces=12
```

```
SELECT * FROM names
      WHERE age = ${age}
hive -f myscript.hive -hivevar age=33
```

Objectives

- About Hive
- Comparing Hive to SQL
- Hive Queries
- Hive Partitions, Buckets, and Skewed
- Sorting Data
- Hive Join Strategies

Hive 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 resource-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 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

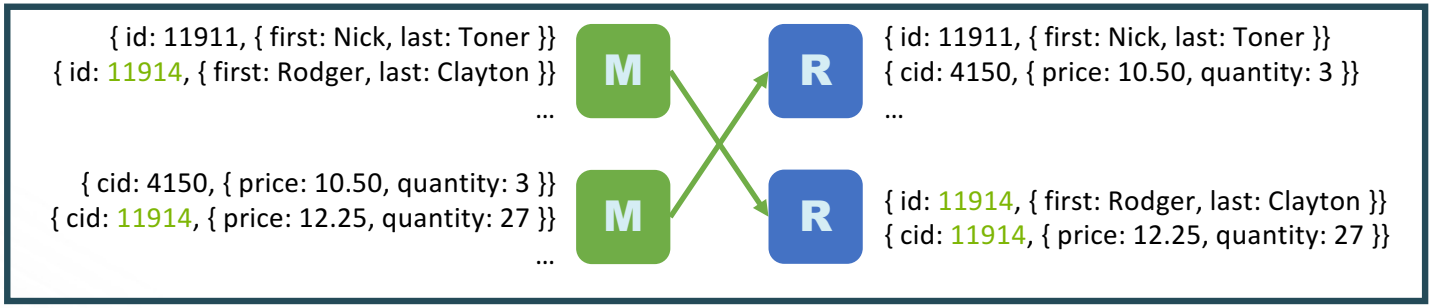
customer

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

orders

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 orders ON customer.id = orders.cid;
```



Map (Broadcast) Joins

customer

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

orders

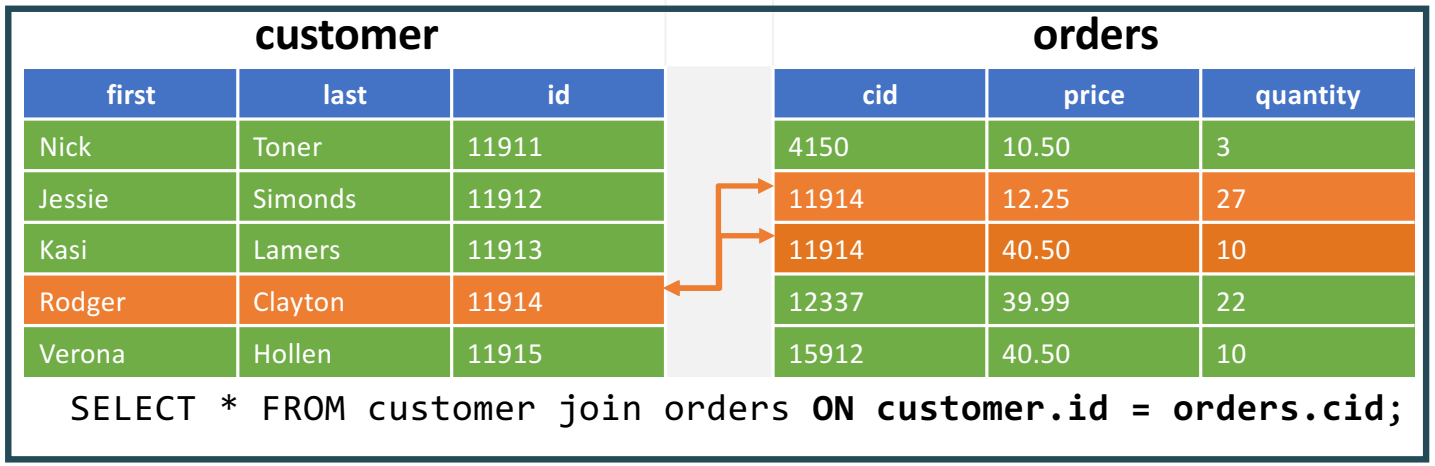
cid	price	quantity
4150	10.50	3
11914	12.25	27
3491	5.99	5
2934	39.99	22
11914	40.50	10

```
SELECT * FROM customer JOIN orders ON customer.id = orders.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.

Sort-Merge-Bucket Joins



customer

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

orders

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 orders ON customer.id = orders.cid;
```

Distribute and sort by the most common join key.

```
CREATE TABLE orders (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(cid) INTO 32 BUCKETS;
```

Invoking a Hive UDF

```
ADD JAR /myapp/lib/myhiveudfs.jar;
CREATE TEMPORARY FUNCTION
ComputeShipping
   AS 'hiveudfs.ComputeShipping';

FROM orders SELECT
   address,
   description,
   ComputeShipping(zip, weight);
```


Advanced Hive
Programming
HORTONWORKS
UNIVERSITY

Topics Covered

- Performing a Multi-Table/File Insert
- Understanding Views
- Defining Views
- Using Views
- The OVER Clause
- Using Windows
- Hive Analytics Functions
- *Lab: Advanced Hive Programming*
- Hive File Formats
- Hive SerDe

- Performing a Multi-table/File Insert

Objectives

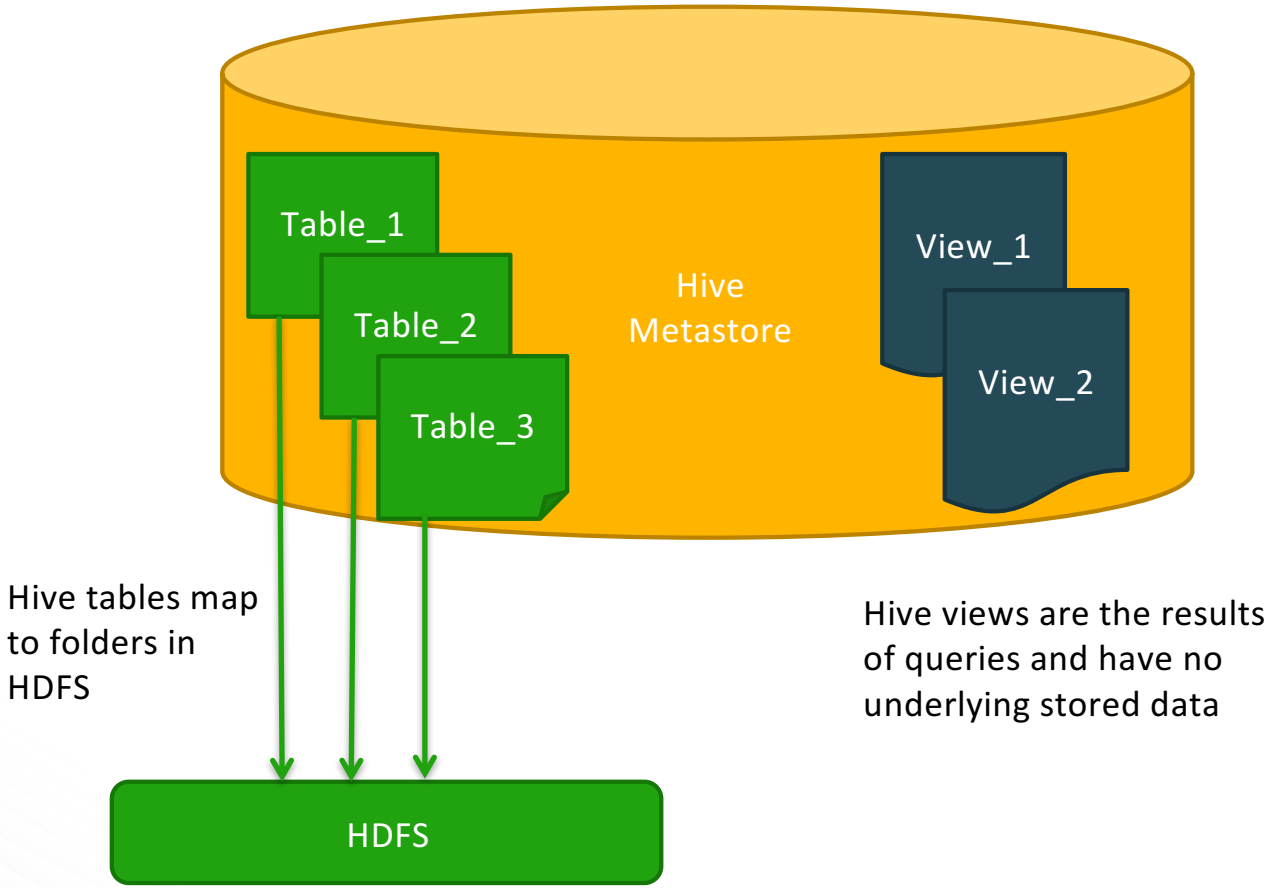
Performing a Multi-Table/File Insert

```
insert overwrite directory '2014_visitors' select * from wh_visits
where visit_year='2014'
insert overwrite directory 'ca_congress' select * from congress
where state='CA' ;
```

No semicolon

```
from visitors
INSERT OVERWRITE TABLE gender_sum
   SELECT visitors.gender, count_distinct(visitors.userid)
   GROUP BY visitors.gender
 INSERT OVERWRITE DIRECTORY '/user/tmp/age_sum'
   SELECT visitors.age, count_distinct(visitors.userid)
   GROUP BY visitors.age;
```

Understanding Views



Objectives

- Performing a Multi-Table/File Insert
- Defining and Using Views

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;
```

Using Views

You use a view just like a table:

```
from 2010_visitors
  select *
  where info_comment like "%CONGRESS%"
  order by lname;
```

The TRANSFORM Clause

- You can write your own custom mappers and/or reducers in Hive
- Use the **TRANSFORM** clause to specify your custom script:

```
add file splitwords.py;
add file countwords.py;

FROM (
   FROM mytable
   SELECT TRANSFORM(keywords) USING 'python splitwords.py'
   AS word, count
   CLUSTER BY word
) wc
INSERT OVERWRITE TABLE word_count
  SELECT TRANSFORM(wc.word, wc.count) USING 'python countwords.py'
  AS word, count;
```

Objectives

- Performing a Multi-Table/File Insert
- Defining and Using Views
- Using Clauses and Windows

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;
```

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;
```

Using Windows – cont.

```
SELECT cid, sum(price) OVER
(PARTITION BY cid ORDER BY price ROWS
BETWEEN 2 PRECEDING AND 3 FOLLOWING)
FROM orders;
```

```
SELECT cid, sum(price) OVER
(PARTITION BY cid ORDER BY price ROWS
BETWEEN UNBOUNDED 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;
```

Objectives

- Performing a Multi-Table/File Insert
- Defining and Using Views
- Using Clauses and Windows
- Hive Programming

Hive File Formats

- Text file
- SequenceFile
- RCFile
- ORC File

```
CREATE TABLE names
  (fname string, lname string)
STORED AS RCFile;
```

Hive SerDe

- SerDe = serializer/deserializer
- Determines how records are read from a table and written to HDFS

```
CREATE TABLE emails (
  from_field string,
  sender string,
  email_body string)
  ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
  OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
  TBLPROPERTIES (
'avro.schema.url'='hdfs//nn:8020/emailschema.avsc');
```

Hive ORC Files

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
```

Computing Table and Column Statistics

```
ANALYZE TABLE tablename COMPUTE
STATISTICS;
```

```
ANALYZE TABLE tablename COMPUTE
STATISTICS FOR COLUMNS column_name_1,
column_name_2, ...
```

```
DESCRIBE FORMATTED tablename
```

```
DESCRIBE EXTENDED tablename
```

Objectives

- Performing a Multi-Table/File Insert
- Defining and Using Views
- Using Clauses and Windows
- Hive Programming
- Hive Optimization and Queries

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***
- To use CBO, you need to:
  - Analyze the table and relevant columns
  - Set the appropriate properties

Optimizing Queries with Statistics

```
analyze table tweets compute statistics;

analyze table tweets compute statistics for
columns sender, topic;

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

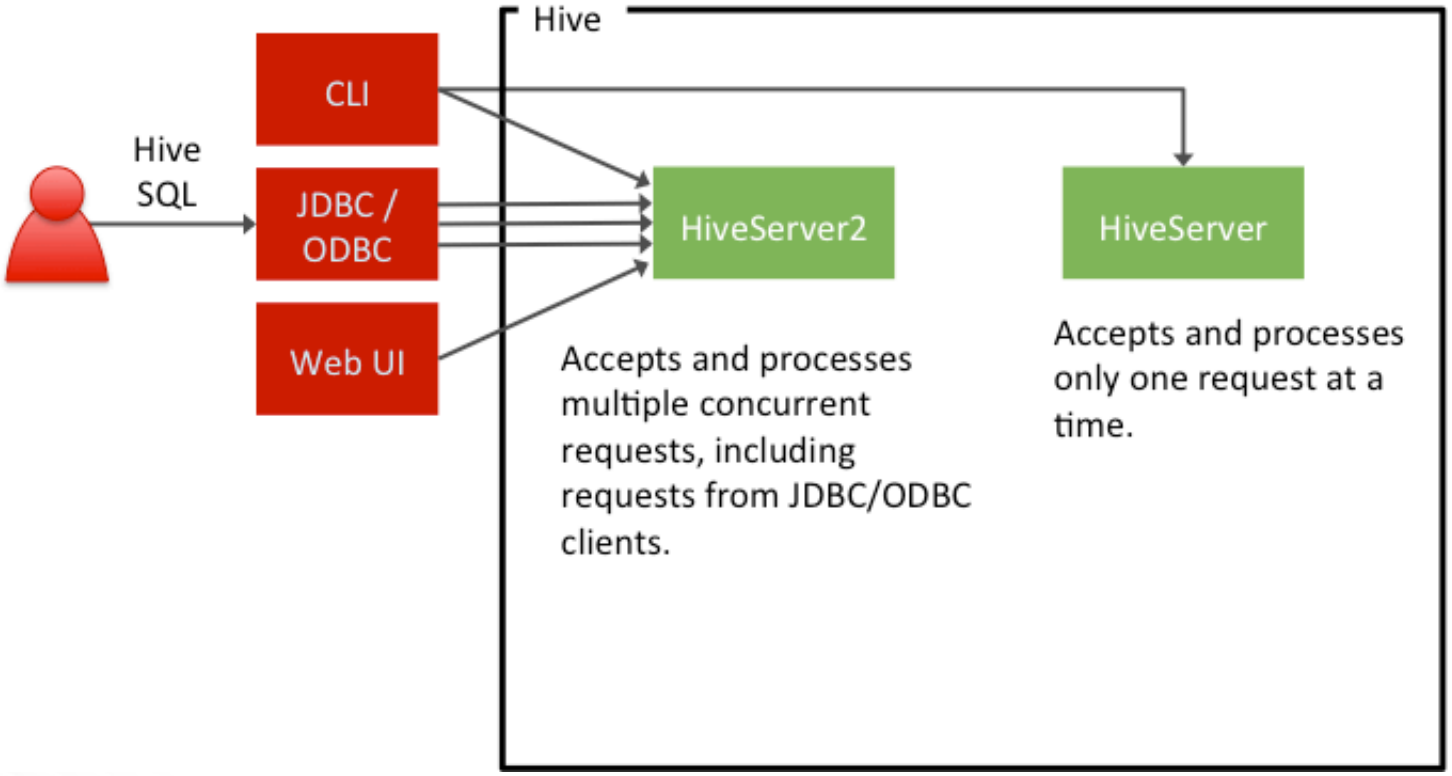
Vectorization

Before

After

Vectorization + ORC files = a huge breakthrough in Hive query performance

Using HiveServer2

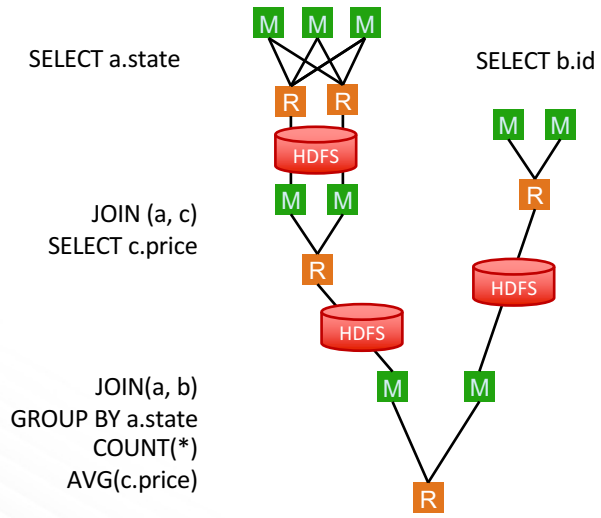


Understanding Hive on Tez

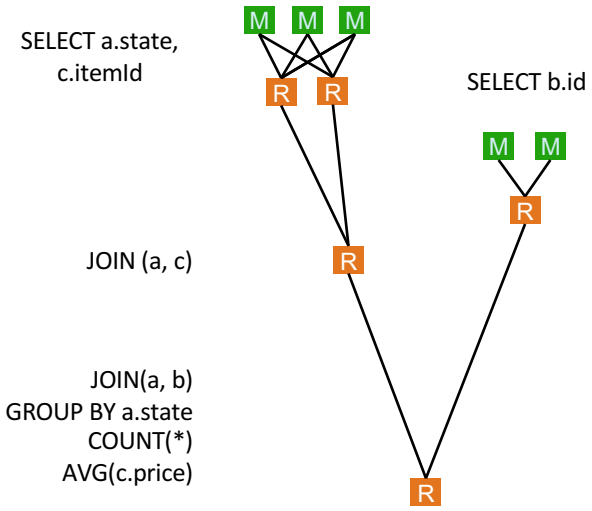
```
SELECT a.state, COUNT(*), AVG(c.price)
FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```

Tez avoids unneeded writes to HDFS

Hive – MapReduce



Hive – Tez



Using Tez for Hive Queries

Set the following property in either **hive-site.xml** or in your script:

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

Hive Optimization Tips

- Divide data amongst different files that can be pruned out by using partitions, buckets, and skews
- 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)
- Take advantage of Tez

Hive Query Tunings

- mapreduce.input.fileinputformat.split.maxsize
- mapreduce.input.fileinputformat.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.

Lesson Review

1. What is the benefit of performing two **insert** queries in the same Hive command?
2. **True or False**: Hive views are materialized when they are defined.
3. Suppose an **employees** table has 200 rows and its **department** column has 15 distinct values. How many rows would be in the result set of the following query?

```
from employees
select fname,lname,MAX(salary)
over (partition by department);
```

4. Explain what the following query is computing:

```
from employees
   select fname,lname,AVG(salary)
   over (partition by department order by salary
       rows between 5 preceding and current row);
```

5. Which Hive file format provides the best performance?
6. What does DAG stand for?