# Advanced Hive Programming

# 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

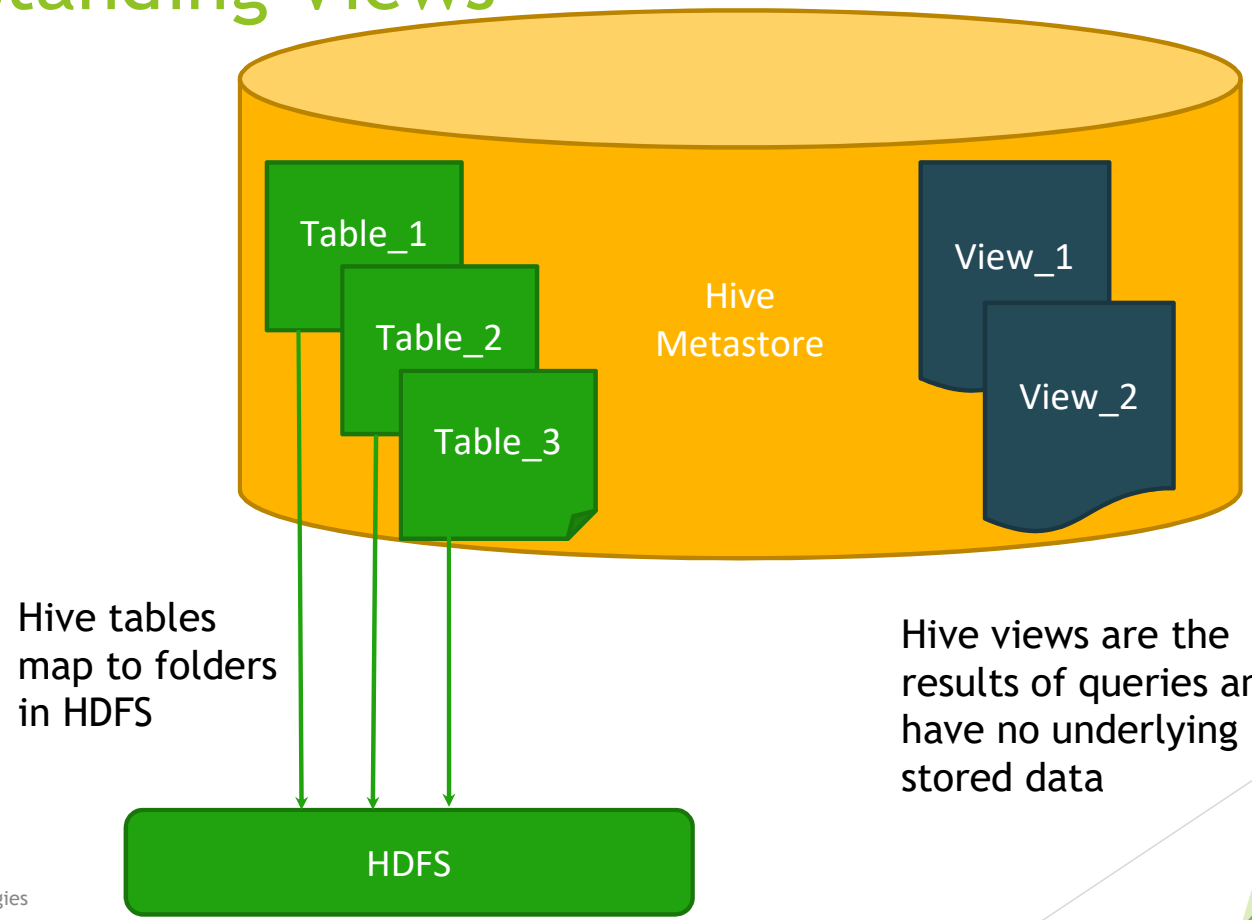
# Performing a Multi-Table/File Insert

```
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




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;
```

# Using Views

You use a view just like a table:

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

# 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;
```

# **Lab:** Advanced 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.AvroContainerInputForma
t'
  OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputForm
at'
  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
```

# 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

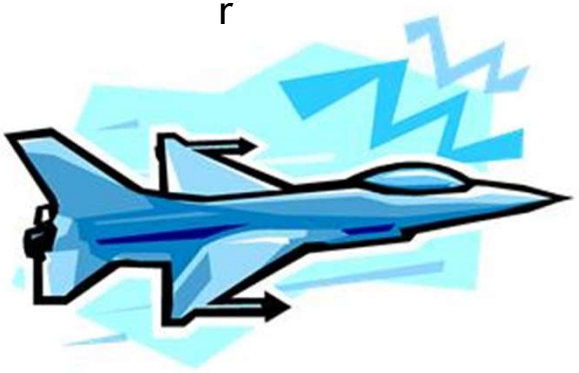
# Optimizing Queries with Statistics

```
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

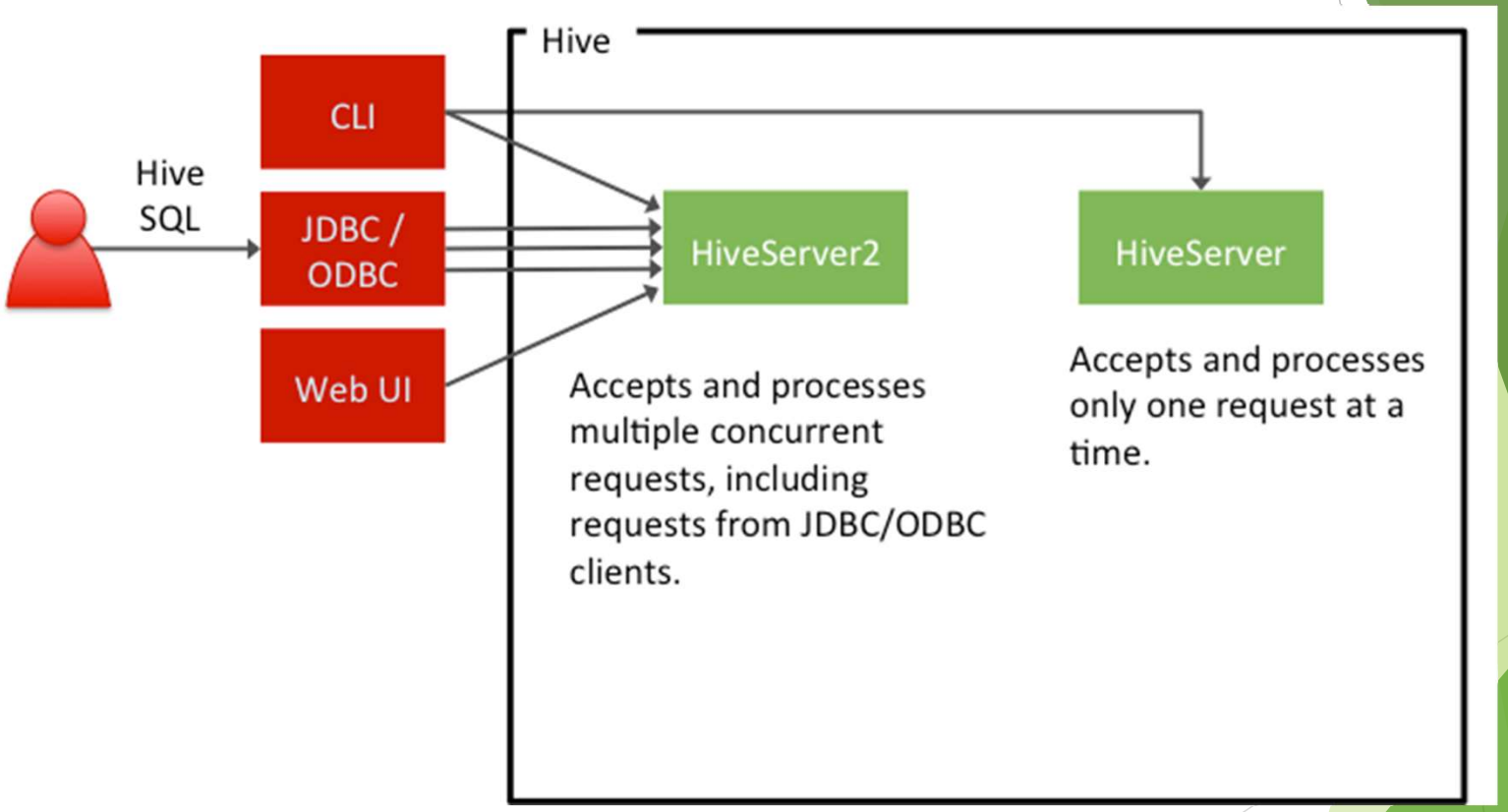
Before

After



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

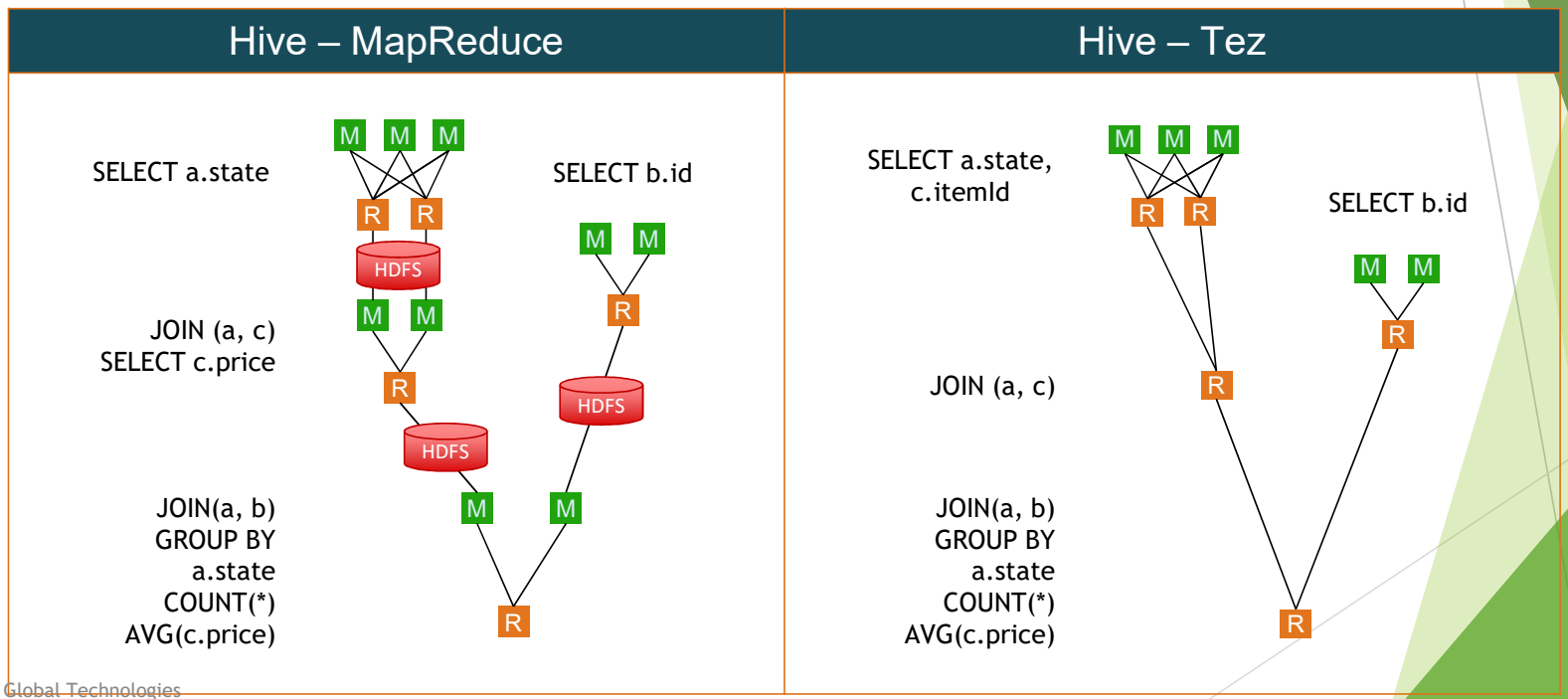
# Using HiveServer2



# 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



# 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