CSP 554 BIG DATA TECHNOLOGIES

Module 04

Hive

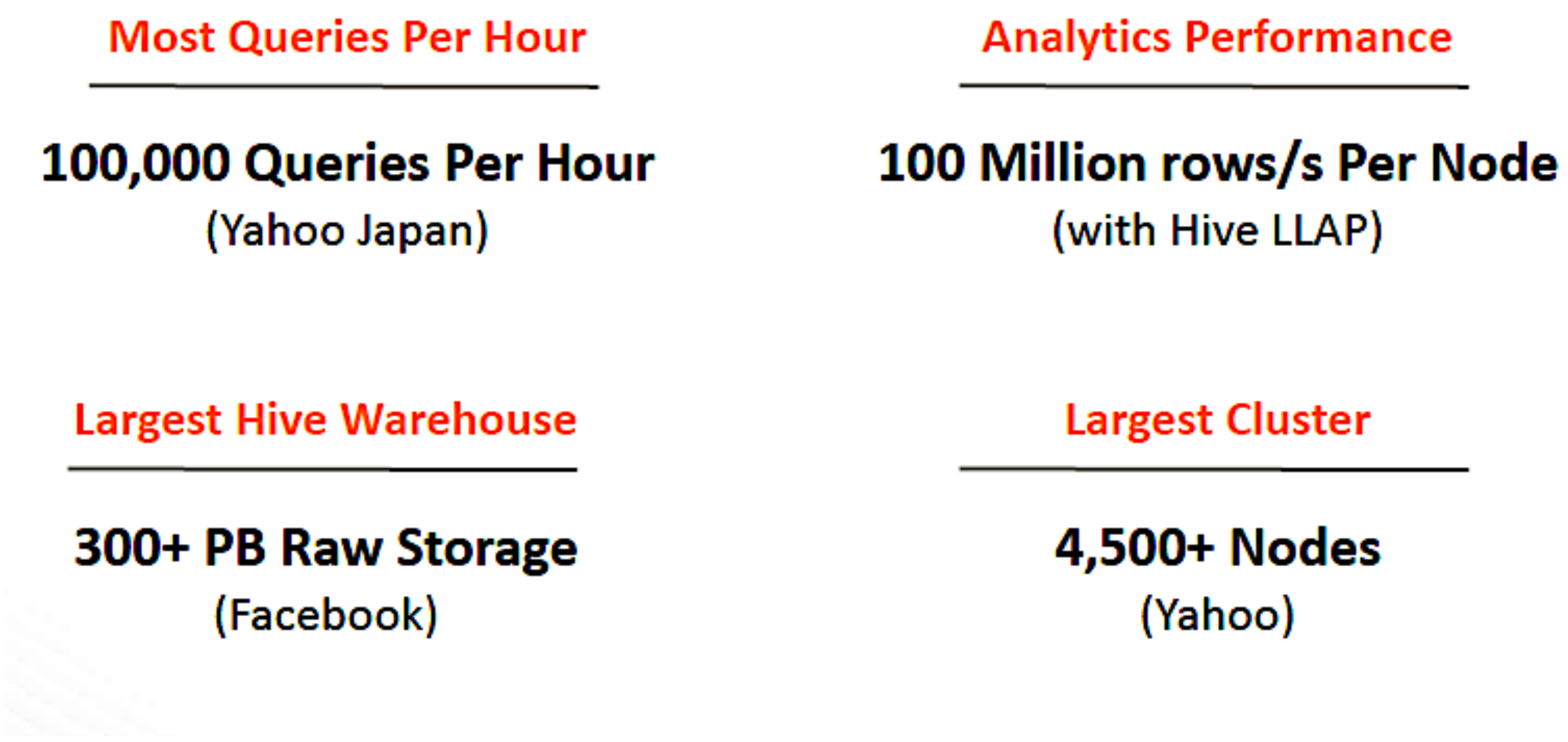
Hadoop For Big Data Management

- The *Hadoop* ecosystem has emerged as a cost-effective way of working with large data sets
- It imposes a particular programming model, called *MapReduce*
- Underneath this computation model is the *Hadoop Distributed Filesystem* (HDFS)
- However, a challenge remains…
  - How do you move an existing data infrastructure to Hadoop
  - When that infrastructure is based on traditional relational databases and the *Structured Query Language* (SQL)?
- What about the large base of SQL users, both expert database designers and administrators
  - As well as casual users who use SQL to extract information from their data warehouses?
- This is where *Hive* comes in

Apache Hive

- An open source SQL-like system built on top of Hadoop for querying and analyzing large datasets
- As an alternative to writing complex MapReduce jobs, with Hive, you just need to submit SQL-like queries
- Hive is targeted towards users who are comfortable with SQL and abstracts the complexity of Hadoop
- Hive use a language called HiveQL (HQL), which is very similar (although not completely identical) to SQL
- The Hive system operates by automatically translating SQL-like queries into (optimized) MapReduce jobs
    - Or jobs for one of two more specialized parallel execution engines: Tez or LLAP

Hive Usage



Most Queries Per Hour

100,000 Queries Per Hour
(Yahoo Japan)

Analytics Performance

100 Million rows/s Per Node
(with Hive LLAP)

Largest Hive Warehouse

300+ PB Raw Storage
(Facebook)

Largest Cluster

4,500+ Nodes
(Yahoo)

Some Hive Alternatives

- Apache Impala
  - Developed by Cloudera
- Apache Drill
- Presto
  - Facebook
- SparkSQL
  - We will explore this as well
- AWS Athena
  - Performs SQL queries over the contents of S3 buckets

Who Developed Hive and Why

- Facebook confronted significant challenges working with big data before their implementation of Apache Hive
- The volume of the data Facebook generated exploded, making it very difficult to handle
- And their traditional relational database systems could not handle the load
- As a result, Facebook began looking or better options
- Facebook initially tried using MapReduce but they found it difficult to program
- On the other hand their development organization had a deep knowledge of SQL
- Hive allowed them to overcome the challenges they were facing: (1) big data and (2) limited MapReduce knowledge

Who Developed Hive and Why

- SQL knowledge is widespread for a reason
- It’s an effective, reasonably intuitive model tor organizing and using data
- Mapping these familiar data operations to the low-level MapReduce Java API can be daunting
  - Even for experienced Java developers
- Hive does this dirty work for you, so you can focus on the query itself
- Hive translates most queries to MapReduce jobs exploiting the scalability of Hadoop
  - While presenting a familiar SQL abstraction.

Who Developed Hive and Why

- Most data warehouse applications are implemented using relational databases that use SQL as the query language
- Hive lowers the barrier for moving these applications to Hadoop
- People who know SQL can learn Hive easily
- Without Hive, these users must learn new languages and tools to become productive again
- Hive makes it easier for developers to port SQL-based applications to Hadoop, compared to other tool options
- Without Hive, developers would face a challenge when porting their SQL applications to Hadoop

But Hive is Not…

- Not a relational database, but a parallel processing layer over HDFS and S3
  - Hive complies HQL requests into a job executed by one of the MapReduce, Tez (specific to Hive) or Spark execution engines
- Not designed for On Line Transaction Processing (OLTP), but is used for data exploration, analysis, and reporting
- Not a system for supporting real-time queries
  - Because Hadoop is a batch-oriented system, Hive queries have higher latency due to the start-up overhead for MapReduce jobs
  - Queries that would finish in seconds for a relational database take longer for Hive, even for relatively small data sets
  - Because the data files or S3 objects it processes are not specially formatted for efficient SQL queries
- Not a system supporting updates to records/rows or ACID semantics in general *but only under special conditions*

Hive For Big Data Management

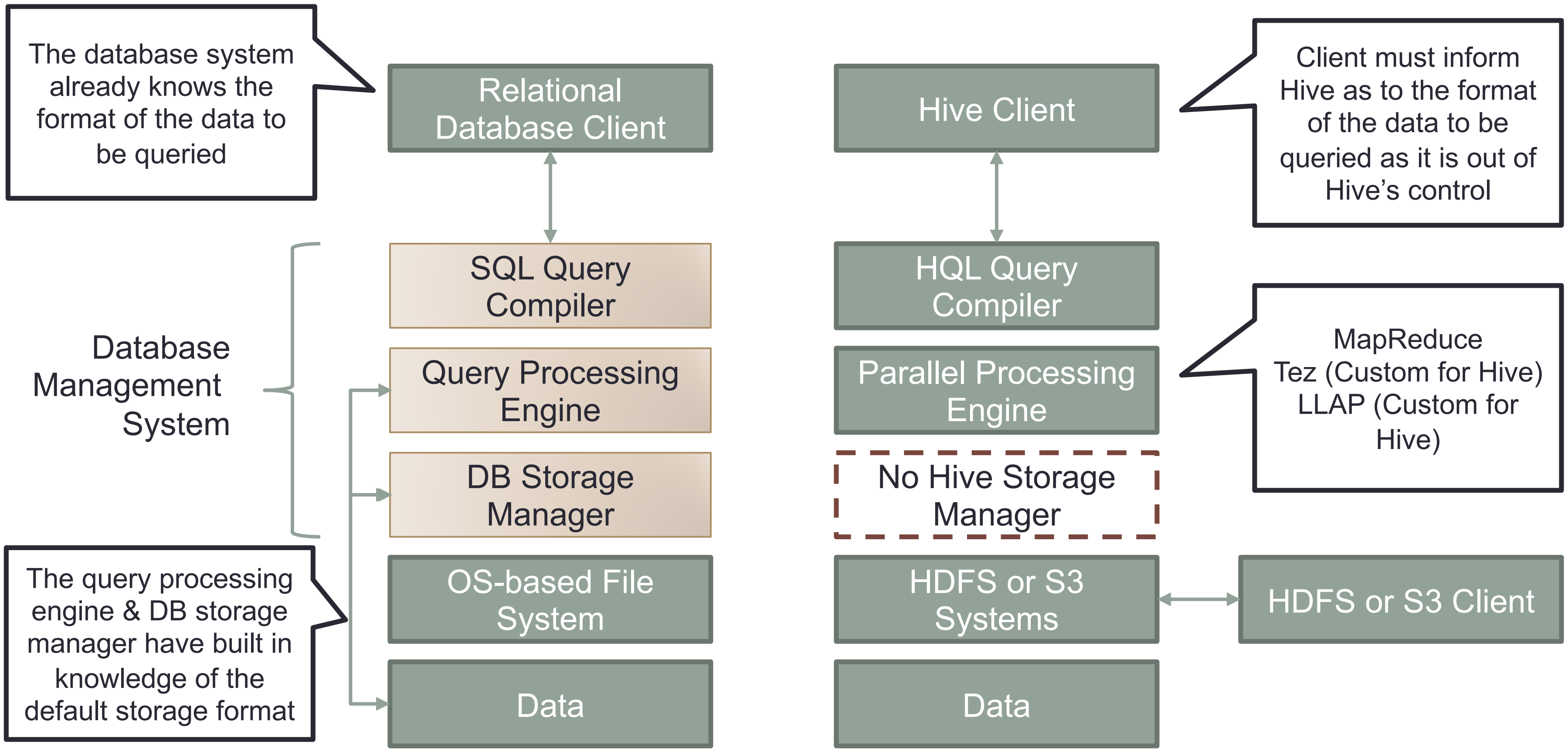
- So, Hive doesn't provide crucial features required for OLTP, *Online Transaction Processing*
- It's closer to being an OLAP tool, *Online Analytic Processing*,
- But Hive isn't ideal for satisfying the "online" part of OLAP, at least today
  - Since there can be significant latency between issuing a query and receiving a reply,
  - Due to the overhead of Hadoop and due to the size of the data sets Hadoop was designed to serve.
- If you need OLTP features for large-scale data, you should consider using a *NoSQL* database:
  - *Hbase*—a *NoSQL* database integrated with Hadoop,
  - *Commercial (also Cloud) NoSQL Databases: Cassandra, MongoDB*
  - *Cloud NoSQL Databases: AWS DynamoDB*, Azure CosmoDB

How Hive Works

- Hive has a compiler taking one language—HQL—and converting it in to code for a parallel processing engine
- And that code executes against data stored in HDFS or S3
- But the question arises: how did the data get into HDFS files or S3 objects in the first place?
- And the answer is: by any of a number of means… Via Sqoop, MapReduce, Spark, etc.
- But that means any process can create and organize data in HDFS and S3 outside of the awareness of Hive
- And because of this Hive has no control over what format the data in HDFS or S3 might be

How Hive Works

- Now this is very different from relational databases
- A relational database controls the format of the data it manages on behalf of its clients
- Hive must adapt its processing to make use of the format of data controlled by other clients


The database system already knows the format of the data to be queried
Relational Database Client
SQL Query Compiler
Query Processing Engine
DB Storage Manager
OS-based File System
Data
Database Management System
The query processing engine & DB storage manager have built in knowledge of the default storage format
Hive Client
Client must inform Hive as to the format of the data to be queried as it is out of Hive's control
HQL Query Compiler
Parallel Processing Engine
MapReduce
Tez (Custom for Hive)
LLAP (Custom for Hive)
No Hive Storage Manager
HDFS or S3 Systems
HDFS or S3 Client
Data

How Hive Works

Relational Database

The table definition assumes the database storage engine has control of the format of stored data and the query processing engine also knows that format…

So, there s no need to describe the table data format as part of the schema definition

```
CREATE TABLE employees (
name       VARCHAR,
state      VARCHAR);
```

A table schema can be superimposed over an existing HDFS file or S3 object

Hive

The Hive query processing engine does not control or have knowledge of the format of data in HDFS or S3

So, the table definition includes a detailed spec of the format of of the stored data…

```
CREATE TABLE employees (
name STRING,
state STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION 'hdfs://user/Hadoop/mydata.txt';
```

Hive Example

- Create a Hive database. This does not allocate any storage only create a name space

```
CREATE DATABASE IF NOT EXISTS DemoDB;
```

Hive Example

- Define the schema of a table in the DemoDB database
- Indicate each record (row) in the database has tab separated fields and each record (row) is terminated with a newline (‘\n’)

```
CREATE TABLE IF NOT EXISTS DemoDB.salaries (
name STRING,
jobTitle STRING,
agencyID STRING,
agency STRING,
hireDate STRING,
annualSalary DOUBLE,
grossPay DOUBLE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t'
LINES TERMINATED BY ‘\n'
STORED AS TEXTFILE;
```

The default is CTRL-A (/001). Here fields are separated by tabs

This is the default, no need to mention in practice

This is the default, no need to mention in practice

Hive Example

- Load data from a Linux file holding records with tab separated fields
- This data is held in an HDFS file managed by Hive

```
LOAD DATA LOCAL INPATH './Salaries.tsv’
OVERWRITE INTO TABLE DemoDB.salaries;
```

- Now you can query information from this table (actually an HDFS file)

```
SELECT name FROM DemoDB.salaries
WHERE agency = ‘Finance’;
```

Hive

- An essential tool in the *Hadoop* ecosystem…
- That provides an *SQL* (Structured Query Language) dialect…
- Hive provides an *SQL* dialect called *Hive Query Language* (*HiveQL, HQL*)…
- For querying data stored in the *Hadoop Distributed Filesystem* (HDFS)…
- Or other filesystems that integrate with Hadoop
  - Such as Amazon's *S3…*
- *And* databases like *HBase* (the Hadoop database)

Hive Query Language (HIVEQL, HQL) Capabilities

- Hive's SQL provides the basic SQL operations
- These operations work on tables or partitions
- Operations are:
  - Ability to filter rows from a table using a WHERE clause
  - Ability to select certain columns from the table using a SELECT clause
  - Ability to do equijoins between two tables
  - Ability to evaluate aggregations on multiple "group by" columns for the data stored in a table
  - Ability to store the results of a query into another table
  - Ability to download the contents of a table to a local directory
  - Ability to manage tables and partitions (create, drop and alter)

Hive New Features

Transactional Read and Write

- Originally Hive supported write only by adding partitions or loading new files into existing partitions
- Support record level INSERT, UPDATE, DELETE
- Example transactional table definition

  ```
  CREATE TABLE T(a int, b int) STORED AS ORC TBLPROPERTIES
  ('transactional'='true');
  ```
- Restrictions
  - Managed Table
  - Table cannot be sorted
  - Currently requires ORC file format
  - … and others

Hive New Features

Other

- HPLSQL
  - Procedural SQL (stored procedures), similar to Oracle's PL/SQL and Teradata's stored procedures
  - Adds cursors, loops (FOR, WHILE, LOOP), branches (IF), HPLSQL procedures, exceptions (SIGNAL)
  - Aims to be compatible with major dialects of procedural SQL to maximize re-use of existing scripts
- LLAP
  - Persistent daemons
  - Saves time on process start up (eliminates container allocation and JVM start up time)
  - All code JITed within a query or two
  - Hot data cached in memory (columnar aware, so only hot columns cached)
  - When possible work scheduled on node with data cached, if not work will be run in other node
- HBase Metastore
  - Add option to use HBase to store Hive's metadata
- Hive-On-Spark Improvements
- Cost Based Optimizer Improvements

WordCount
MapReduce Java API (Partial)

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
                String line = value.toString();
                StringTokenizer tokenizer = new StringTokenizer(line);
                while (tokenizer.hasMoreTokens()) {
                        word.set(tokenizer.nextToken());
                        context.write(word, one);
                }
        }
}
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
                int sum = 0;
                for (IntWritable val : values) {
                        sum += val.get();
                }
                context.write(key, new IntWritable(sum));
        }
}
```

WordCount
HiveQL (Full)

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

WordCount

- In both examples, files were tokenized into words using the simplest possible approach; splitting on whitespace boundaries
- This approach doesn’t properly handle punctuation
- It doesn’t recognize that singular and plural forms of words are the same word, etc. However, it’s good enough for our purposes here
- The virtue of the Java API is ability to customize each detail of algorithm implementation
- However, most of the time, you just don’t need that level of control and it slows you down considerably when you have to manage all those details.
- If you’re not a programmer, then writing Java MapReduce code is largely out of reach
- However, if you already know SQL, learning Hive is relatively straightforward and many applications are quick and easy to implement

Hive For Big Data Management

- So, Hive is best suited for data warehouse applications, where a large data set is maintained and mined for insights and to produce reports
- Because data warehouse applications are implemented using SQL-based relational databases…
  - Hive lowers the barrier for moving these applications to Hadoop.
- But, unlike most SQL dialects…
  - HiveQL does not conform to the ANSI SQL standard
  - And it differs in various ways from the familiar SQL dialects provided by Oracle, MySQL, and SQL Server
  - However, it is closest to MySQL's dialect of SQL

Hive For Big Data Management

SQL	HQL
Works on RDBMS	Works on HDFS
Interactive; queries provide results in near real time	Batch; queries have overhead due to MapReduce
Works best on small or medium datasets (megabytes or gigabytes)	Works best on large datasets (terabytes or petabytes)

Relational Databases Versus Hive

- Let's say I have a file of records and I want to write them into a relational database to query them using SQL
- Imagine the file is in HDFS as /user/Hadoop/mydata.txt
- Can I do the following
  - Create a database called 'Demo' in my RDBMS (relational database management system)
  - Insert each record from mydata.txt into database 'Demo'
  - Define a table/schema for the data I just wrote to the database
  - Query the data
- No!!!
  - You can only write data into a database after you define a table schema
- This is referred to as "Schema on Write"
  - You must have a table schema defined before you write any data to a database

Relational Databases Versus Hive

- Now say I have a file of records and I want to write them into a Hive database to query them using HQL
- Imagine the file is in HDFS as /user/Hadoop/mydata2.txt
- Can I do the following
    - Create a database called ‘Demo’ in Hive
    - Use the file mydata2.txt exactly as it is without having to somehow specially write it into Hive, because Hive just uses HDFS files directly
    - So any file written to HDFS can become a Hive database table
    - When I want to query the data, only then define a table/schema to superimpose over the existing file mydata2.txt
    - Query the data
- Yes!!
    - You can use any HDFS file as a table in a Hive database
- This is referred to as “Schema on Read”
    - Your schema can be defined after the data file which it describes already exists
    - You need a table schema, not when you write the data (data is just an HDFS file which you can have written previously)…
    - But when you want to query/read the data

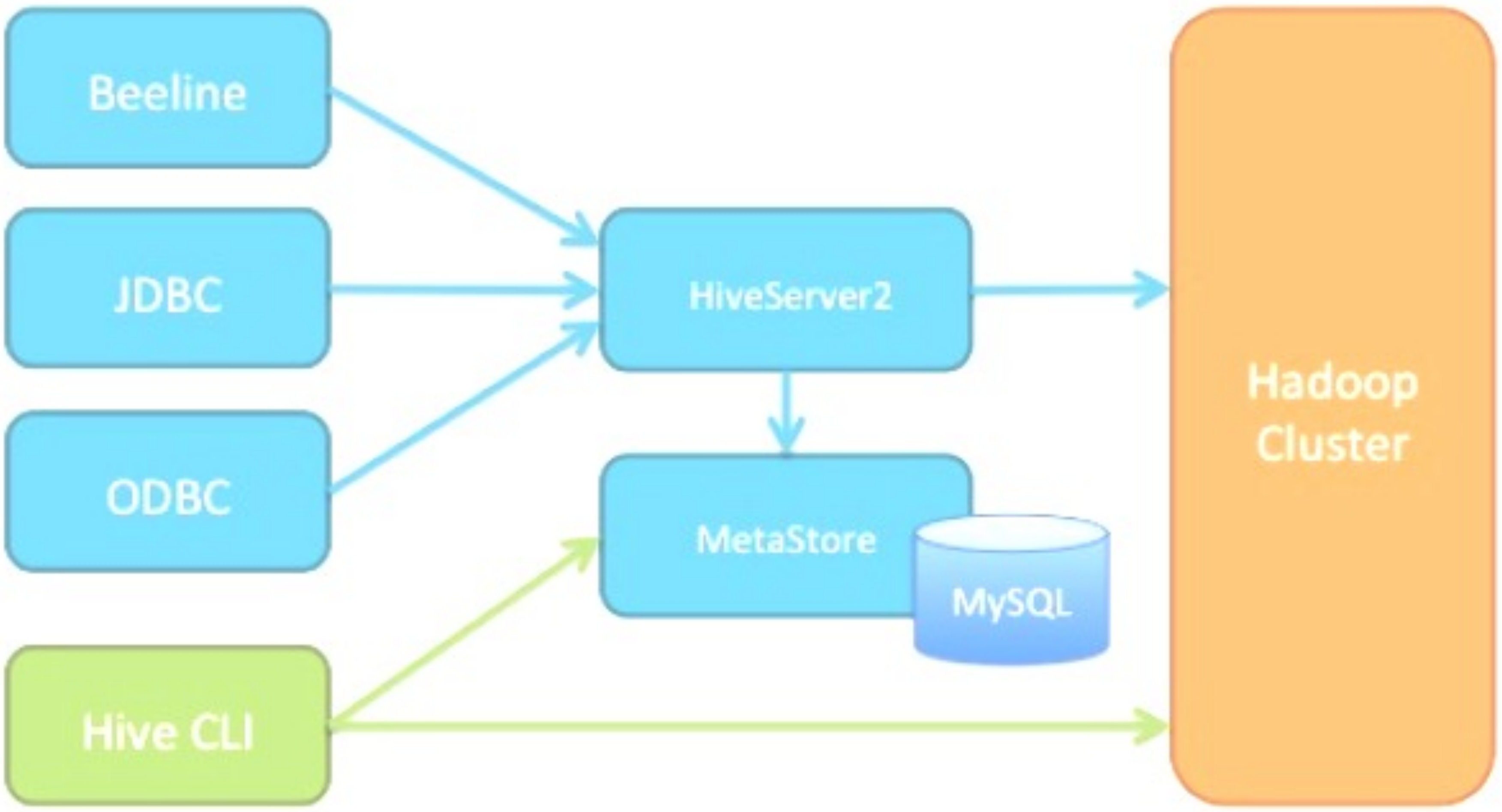
Relational Databases Versus Hive

- Hive enforces schema on read whereas RDBMS enforces schema on write
  - In RDBMS, a table's schema is enforced at data load time
  - If the data being loaded doesn't conform to the schema, then it is rejected
  - But Hive doesn't verify the data when it is loaded, but rather when a it is queried
- Schema on read makes for a very fast initial load, since the data does not have to be read, parsed, and serialized to disk in the database's internal format.
- The load operation is just a copy or move to HDFS directory
- Schema on write makes query time performance faster, since database can index columns and perform compression on the data
  - But it takes longer to load data into the database.
- Hive is based on the notion of Write once, Read many times but RDBMS is designed for Read and Write many times
- In RDBMS, record level updates, insertions and deletes, transactions are possible
- These are not allowed in Hive because Hive was built to operate over HDFS data using MapReduce, where full-table scans are the norm
  - And a table update is achieved by transforming the data into a new table.

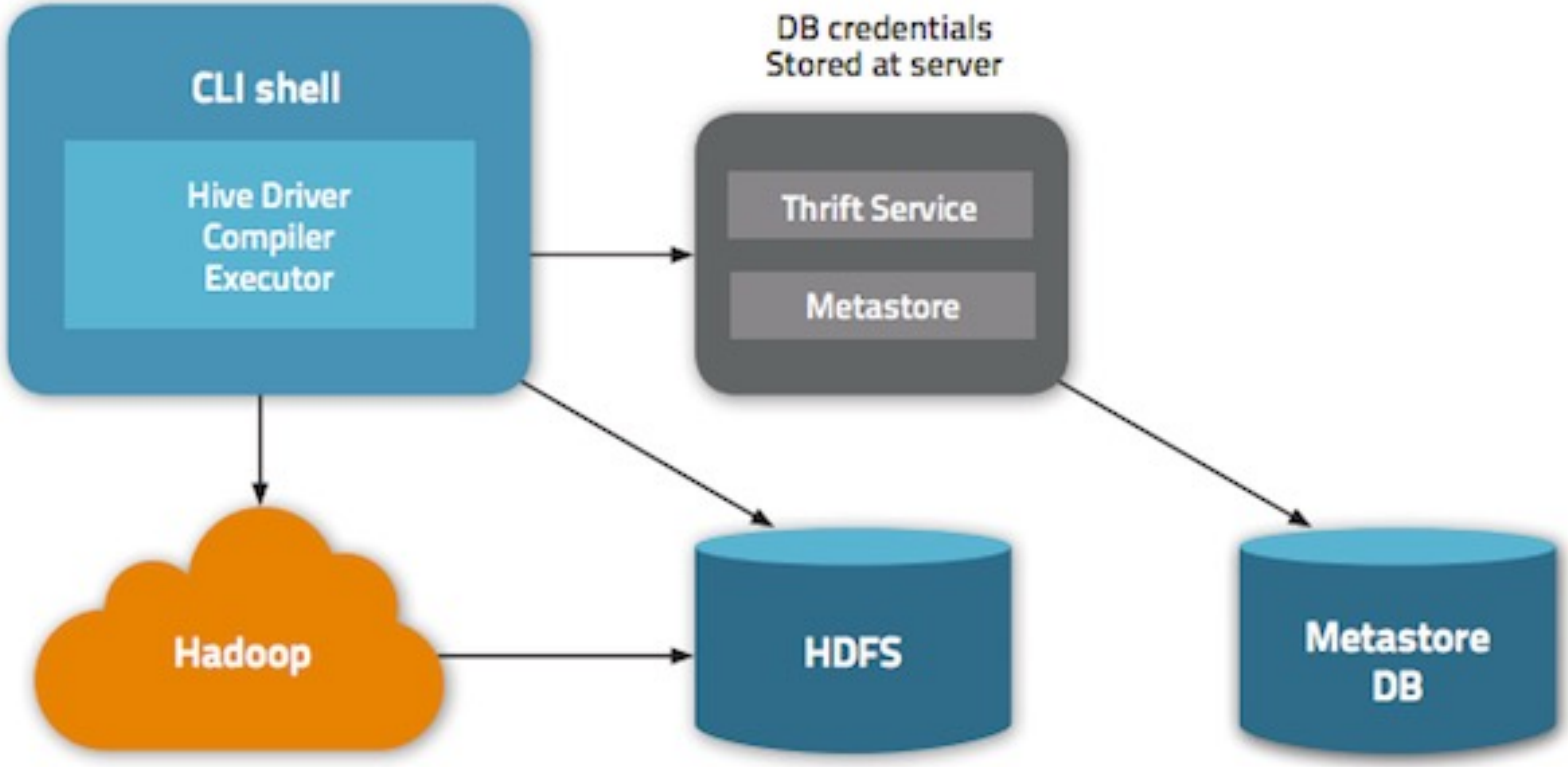
Relational Databases Versus Hive

- In RDBMS, the maximum data size allowed will be in 10’s of Terabytes where Hive can handle 100’s Petabytes easily
- As Hadoop is a batch-oriented system, Hive doesn’t support OLTP (Online Transaction Processing) but is closer to OLAP (Online Analytical Processing)
- But there is significant latency between issuing a query and receiving a reply
  - Due to the overhead of MapReduce jobs
  - Due to the size of the Hadoop data sets
- RDBMS is best suited for dynamic data analysis where fast responses are expected
- But Hive is suited to data warehouse applications, where relatively static data is analyzed
  - Fast response times are not required
  - The data is not changing that rapidly
- To overcome the limitations of Hive, HBase is being integrated with Hive to support record level operations and OLAP

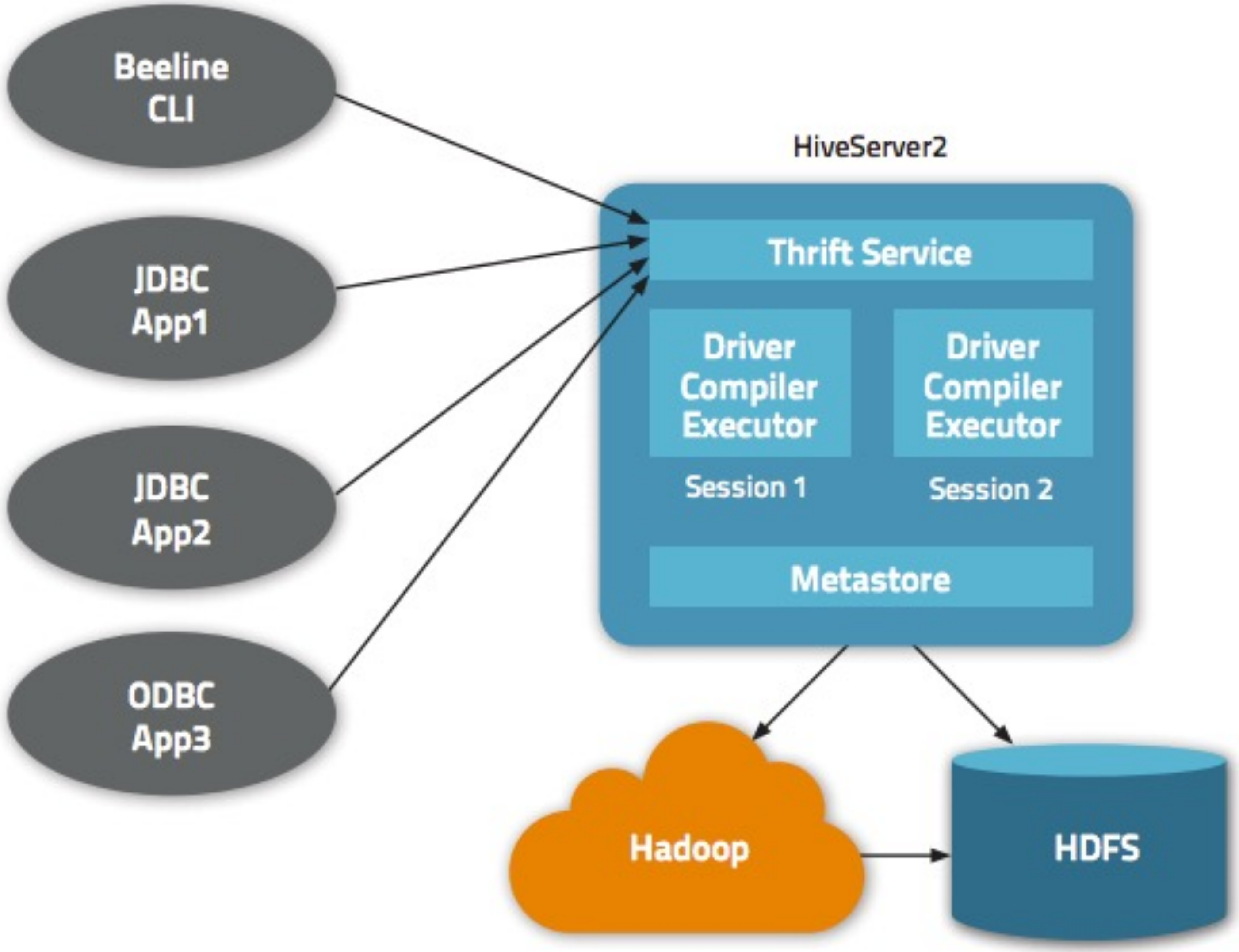
Hive Architecture



Hive Architecture



Hive Architecture



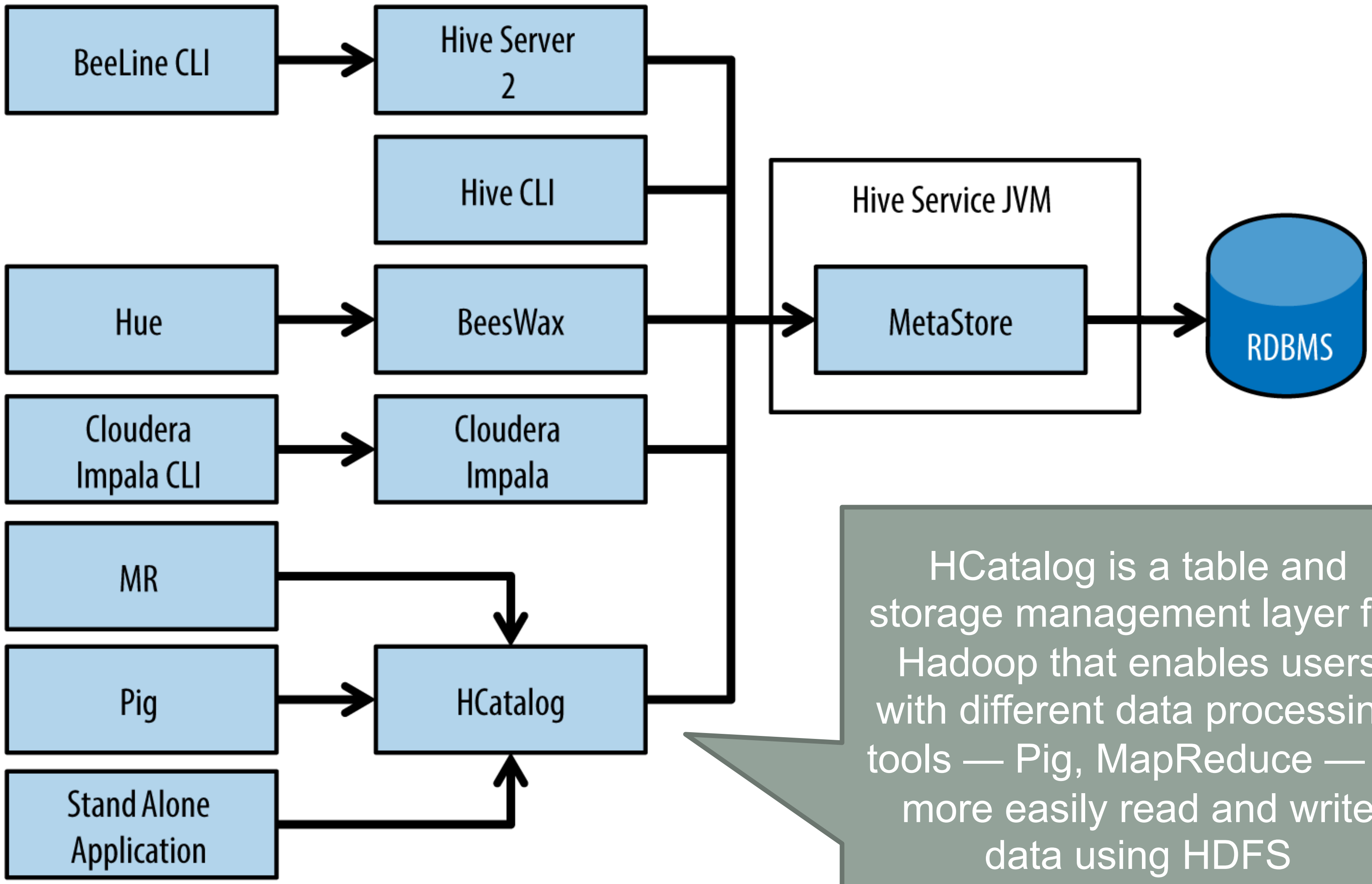
Hive Architecture

- Hive requires only one extra component that Hadoop does not already have; the *metastore* component
- The metastore stores metadata such as table schema and partition information
- Any JDBC-compliant database can be used for the metastore
- In practice, most installations of Hive use MySQL or similar open source database
- Hive metastore HA requires a database that is highly available…
- Such as MySQL with replication in active-active mode

Hive Architecture

- The information required for table schema, partition information, etc., is small
  - Typically much smaller than the large quantity of data stored in Hive
- As a result, you typically don't need a powerful dedicated database server for the metastore
- However because it represents a Single Point of Failure (SPOF), it is strongly recommended that…
- You replicate and back up this database using standard techniques you would normally use with other relational database instances

Hive Architecture

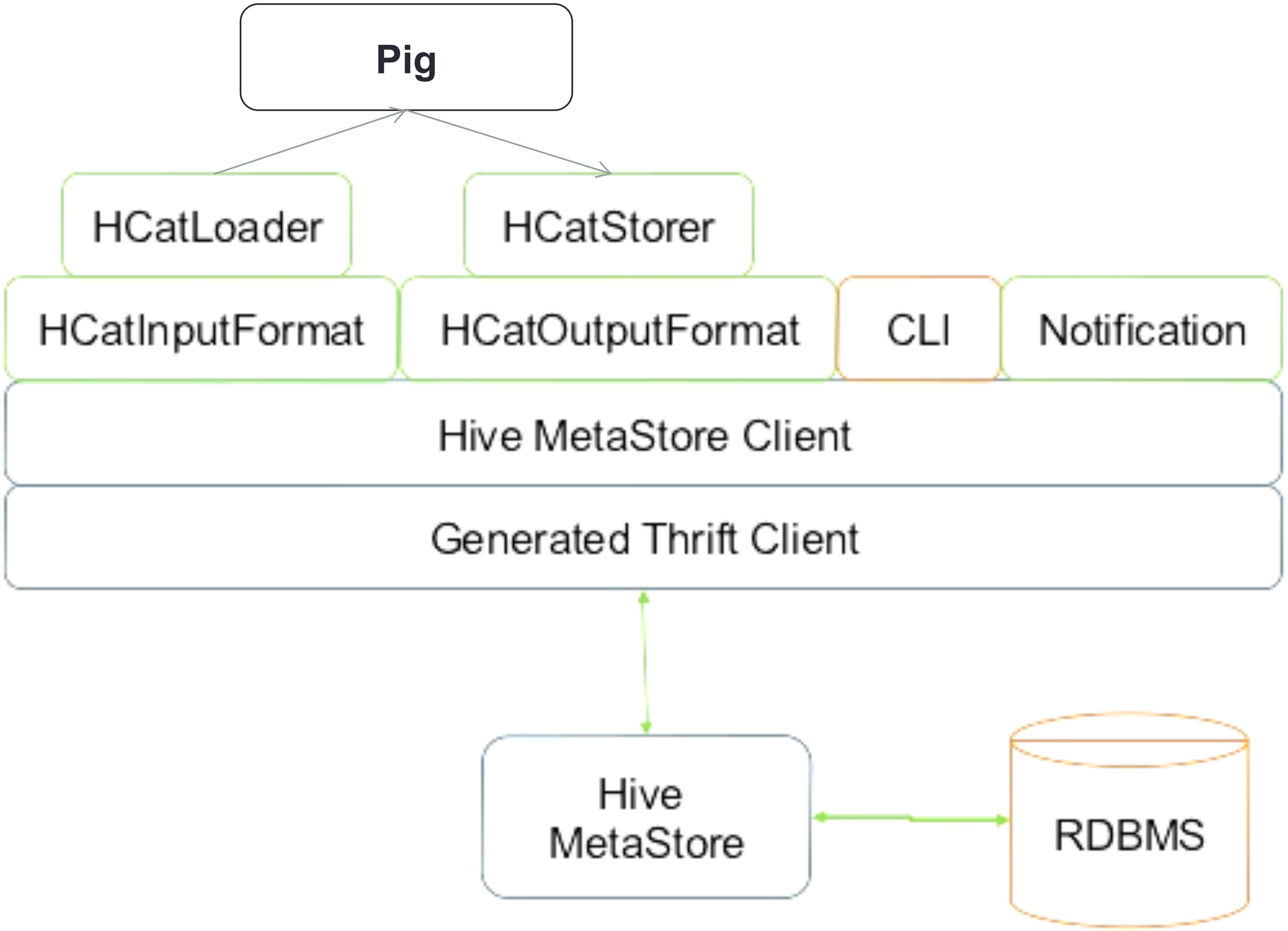


HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools — Pig, MapReduce — to more easily read and write data using HDFS

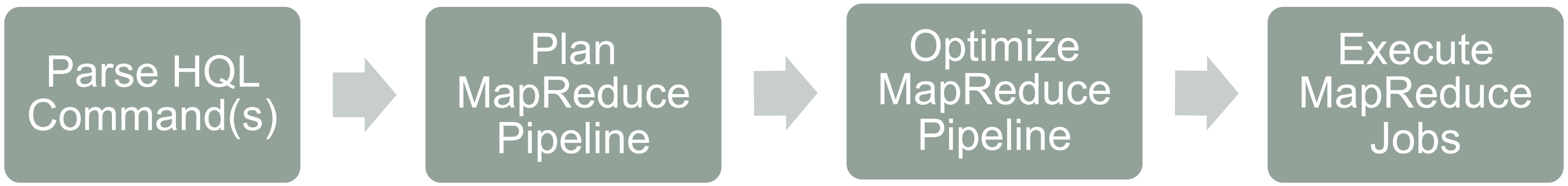
Hive Architecture

- HCatalog is built on top of the Hive metastore and incorporates Hive's DDL
- HCatalog provides read and write interfaces for Pig and MapReduce
- Uses Hive's command line interface for issuing data definition and metadata exploration commands
- HCatalog’s table abstraction presents users with a relational view of data in HDFS
- Ensures users need not worry about where or in what format their data is stored
    - RCFile format, Text file format, SequenceFiles, or ORC files

Hive Architecture



Hive Architecture



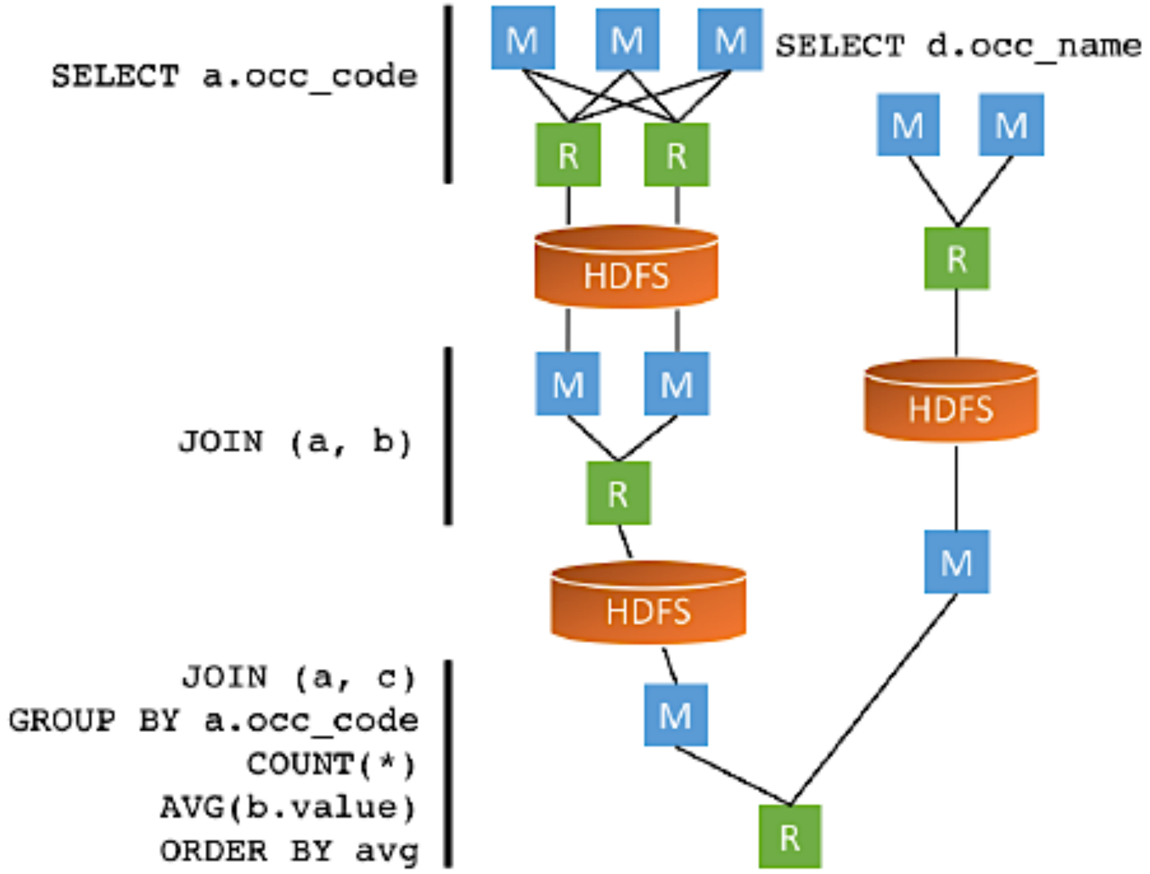
HiveQL Code to Retrieve Data from 3 Tables

```
SELECT a.occ_code, c.occ_name, COUNT(*) AS cnt, AVG(b.value) AS avg
FROM occup a
JOIN occupdata b ON (a.sid = b.sid)
JOIN jobs c ON (a.occ_code = c.occ_code)
GROUP BY a.occ_code, c.occ_name
ORDER BY avg DESC
```

Computation Model of MapReduce.

The following increases the cost and time of the MapReduce execution:

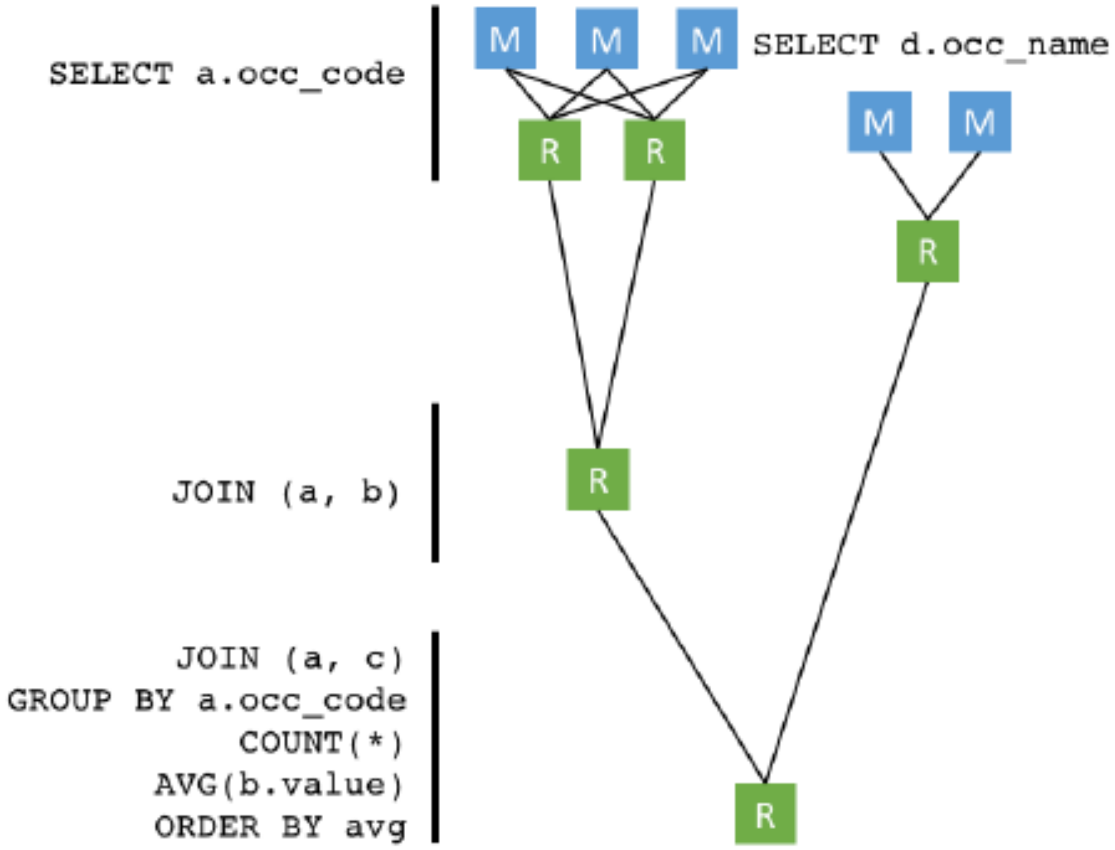
1. To execute this query using the MapReduce execution engine, Hive must launch 4 MR jobs
2. Each MR job has its own start-up time and after processing writes result to HDFS providing data to a subsequent job for read.



Computation Model of Tez

In contrary to MapReduce, Tez performs complex queries as a single execution graph

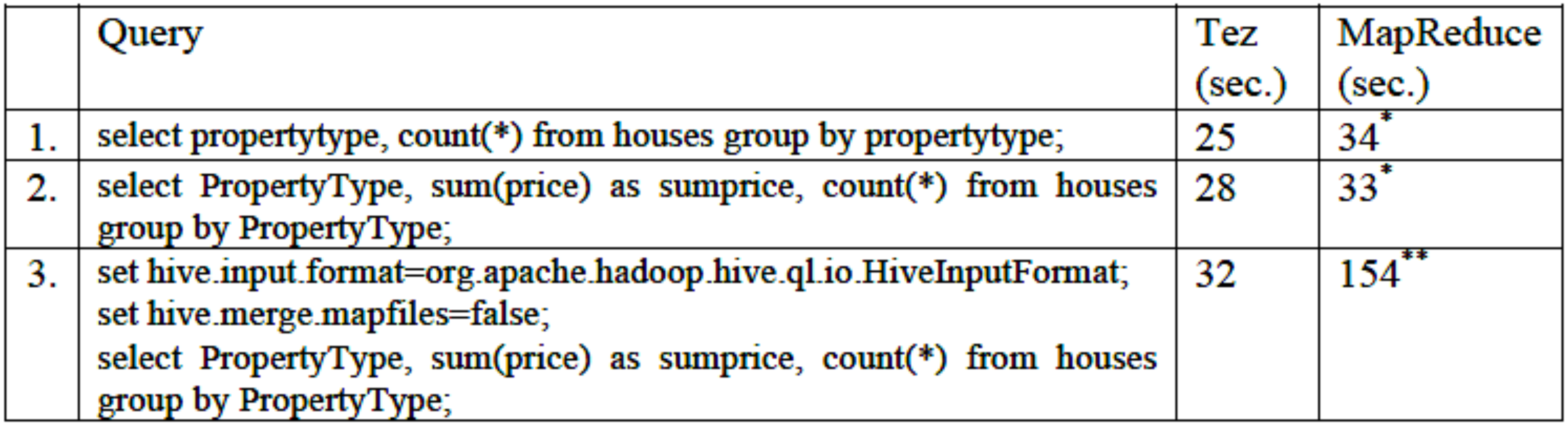
- Tez doesn't implement wasteful intermediate IO operations with HDFS
- Vertexes in the execution graph are processing jobs and edges are data streams
- Tez supports "hot containers" to start jobs immediately without wasting time for
- start-up



Live Long and Process (LLAP)

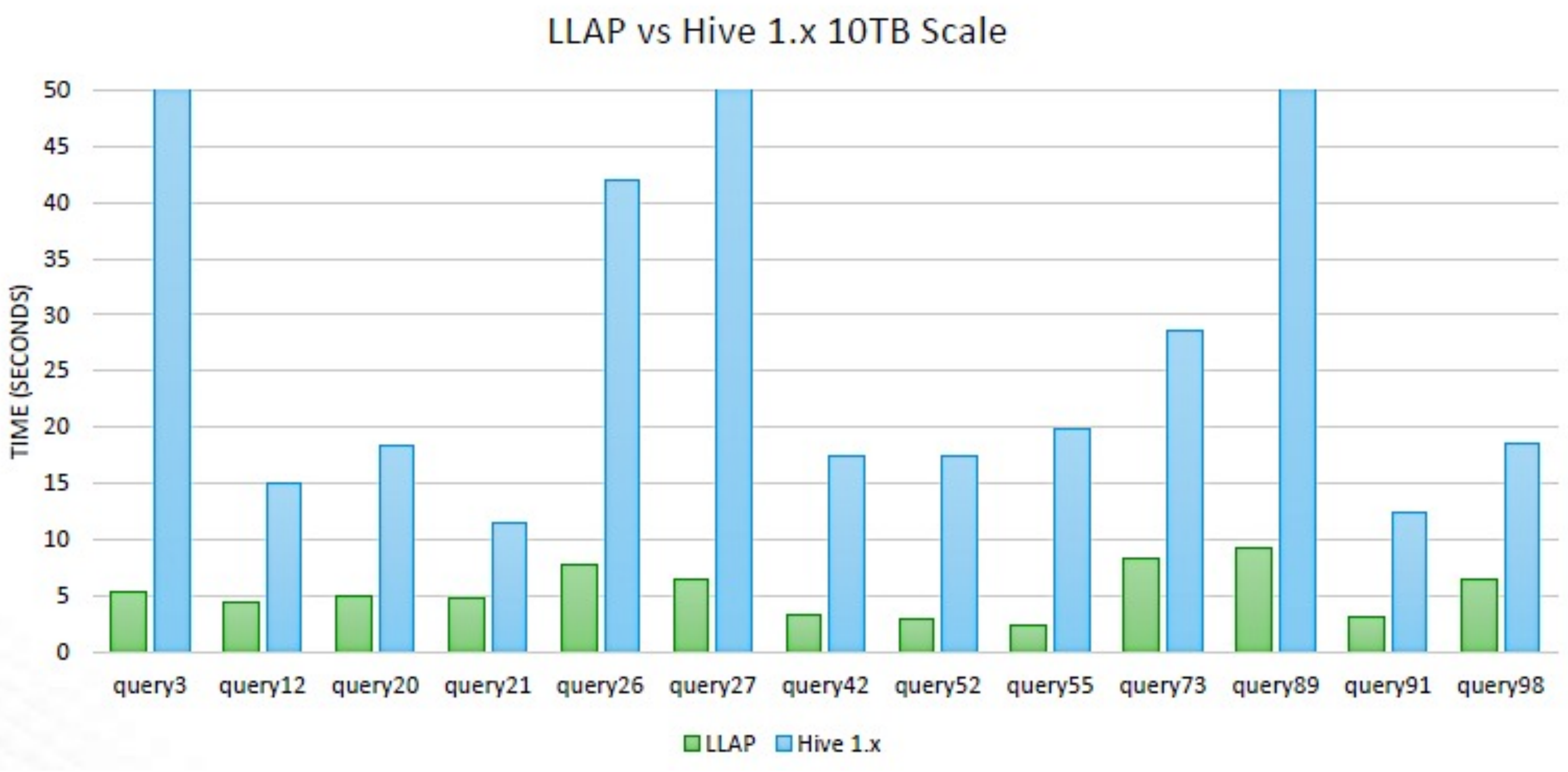
- LLAP is the newest computation paradigm implemented for Hive
- It consists of the set of persistent daemons (processes) that execute fragments of Hive queries
- This persistency allows to start jobs much faster, since containers do not need warm-up.

Example Query Performance by Execution Engine



	Query	Tez (sec.)	MapReduce (sec.)
1.	select propertytype, count(*) from houses group by propertytype;	25	34*
2.	select PropertyType, sum(price) as sumprice, count(*) from houses group by PropertyType;	28	33*
3.	set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat; set hive.merge.mapfiles=false; select PropertyType, sum(price) as sumprice, count(*) from houses group by PropertyType;	32	154**

Example Query Performance by Execution Engine



How Do I Execute Hive Commands Interactively?

- The command line interface or CLI is one of the most common ways to interact with Hive
- Using the CLI you can create databases, create tables, query tables and so on

How Do I Execute Hive Commands Interactively?
Hive CLI (old)

- Hive CLI is a thick client holding logic needed to compile hive commands into MapReduce jobs and execute them on a cluster
- Hive CLI connects directly to HDFS and Hive Metastore
- Can only be used on a client server with direct access to Hadoop cluster services
- Hive CLI depends on HDFS storage access permission for security

How Do I Execute Hive Command Interactively?
Beeline (new)

- Due to security limitations and need for all hive software to be available on the client server the Hive CLI is being deprecated in favor of the Beeline CLI
- The Beeline CLI connects to the Hadoop HiveServer2 service via standard JDBC connections and does not require installation of most Hive libraries on the client machine
  - Only one .jar file: hive-jdbc-<version>-standalone.jar.
- We can run Beeline with limited access to Hadoop cluster and use SQL/JDBC standard-based authorization

Starting Beehive

Enter the following all on one line:

```
beeline -u jdbc:hive2://localhost:10000/ -n hadoop -d
org.apache.hive.jdbc.HiveDriver --showDbInPrompt=true
```

- **-u** *<database URL>*
  - The database URL to connect to
- **-n** *<username>*
  - The user name to connect as
- **-d** *<driver class>*
  - The driver class to use
  - Our choice makes Beeline behave more like the Hive CLI
- **--**showDbInPrompt=[true/false]
  - Display the current database name in prompt. Default is false

How Do I Execute Hive Commands From a File?

- If you are already in Beeline you can use the “source” command to execute a script file (

```
0: jdbc:hive2://localhost:10000/ (default)> source /path/to/file/mycmds.hql
```

- Or equivalently you can use the “!run” command to execute a script file (standard Beeline command)

```
0: jdbc:hive2://localhost:10000/ (default)> !run /path/to/file/mycmds.hql
```

How Do I Add Comments to My Hive Command Files?

- You can embed lines of comments that start with “--” (two hyphens)

file: mycmds.hql

```
-- data and time
-- Intent of the script
SELECT * FROM someTable;
```

- Beeline does not parse these comment lines, if you paste then into the CLI you will get errors

How Can I Execute a Single Hive Command From the Shell Command Line?

- One can use this feature to find a Hive property name easily

  beeline … –e “set;” | grep warehouse

  hive.metastore.**warehouse**.dir=/user/hive/**warehouse**

  hive.**warehouse**.subdir.inherit.perms=true

How Can I Exit Beeline Interactive Mode?

- Simply type “!quit;” followed by the command and terminate the line with a “;”
  - 0: jdbc:hive2://localhost:10000/ (default)> !quit

How Can I Execute Hadoop File System Commands From Hive CLI Interactive Mode?

- You can run the “hadoop fs” commands from within the beeline CLI
- Just enter “dfs” followed by the command then an “;” at the end

```
0: jdbc:hive2://localhost:10000/ (default)> dfs –ls / ;
```

How Can We View and Modify Hive Configuration Parameters?

- The Beeline maintains its configuration parameters in a set of variables internally stored as Java Strings
- You can the values of preexisting variables or add user defined ones
- You can reference variables in queries where Hive replaces the reference with the variable's value
- Inside the Beeline variables are displayed and changed using the SET command
- To output the names and values of all variables

```
0: jdbc:hive2://localhost:10000/ (default)> set;
```

How Can We View and Modify Hive Configuration Parameters?

- To output the name and values a single variable

```
0: jdbc:hive2://localhost:10000/ (default)> set hive.cli.print.current.db
hive.cli.print.current.db =true
```

- To change the value of a variable

```
0: jdbc:hive2://localhost:10000/ (default)> set hive.cli.print.current.db = false;
```

- We will only focus on a few useful or needed variables in our course

What is a Hive “Database”?

- Hive is a technology that can define databases and tables to analyze structured or semi-structured data
- The theme for such data analysis is to store the data in a tabular manner, and issue queries to analyze it
- Although the words are the same, the terms database and table do not mean quite the same thing for Hive as for any relational database management system (RDBMS)
- An RDBMS database is a construct holding the content of tables along with metadata describing access permissions and storage characteristics and other properties

Data Units

Database Namespaces function to avoid naming conflicts for tables, partitions, columns, and so on. Databases can also be used to enforce security for a user or group of users.

Table Records of data described by the same schema

Partition Each Table can have organized into one or more partitions based on the values of a column which determines how the data is grouped

Bucket Data in each partition may in turn be divided into buckets based on some column of the Table

DML Commands

- CREATE
  - CREATE DATABASE …
  - CREATE TABLE …
- DROP
  - DROP DATABASE …
  - DROP TABLE …
- SHOW
  - SHOW DATABASES
  - SHOW TABLES
- DESCRIBE
  - DESCRIBE DATABASE …
  - DESCRIBE TABLE …
- USE
  - USE …

What is a Hive Database?

- Hive uses MapReduce (or Tez) for processing…
- HDFS (or S3) for storage…
- And a small RDBMS and service process for metadata
- For Hive a database is not a storage construct but has two purposes
    - A namespace to disambiguate (avoid naming conflicts for) tables of the same name that may be associated with different projects
    - Db1.Customers and Db2.Customers are two separate tables under Hive even though each has the same name
    - So it is common to use databases to organize production tables into logical grouping
    - Databases can also be used to enforce security (access) policies for users and groups
- In Hive if a table is not explicitly associated with some named database it is associated with "default"

DML Commands

CREATE DATABASE …

Description

Used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables

Syntax

```
CREATE DATABASE [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION hdfs_s3_path]
[MANAGEDLOCATION hdfs_s3_path]
[WITH DBPROPERTIES (property_name=property_value, ...)];
```

DML Commands

CREATE DATABASE …

- The simplest syntax for creating a database is as follows

```
CREATE DATABASE someDatabase;
```

- Hive will indicate an error if “someDatabase” already exists
- You can suppress this error indication with the following variation

```
CREATE DATABASE IF NOT EXISTS someDatabase;
```

- The above is useful for scripts that create a database only if it does not exist and otherwise use the existing one

DML Commands

CREATE DATABASE …

Example

Create a databases called demo (if it does not exist) as a subdirectory of the HDFS ‘user/hive/warehouse’ directory path:

```
CREATE DATABASE IF NOT EXISTS demo
COMMENT “This is a demo database”;
```

Create a databases called demo2 (if it does not exist) in the already existing S3 bucket ‘mydbs’:

```
CREATE DATABASE IF NOT EXISTS demo2
COMMENT “This is another database”
LOCATION ‘s3://mydbs/demo2.db’;
```

DML Commands

CREATE DATABASE …

- Hive will create an HDFS directory or S3 for each database
- Tables in that database will be stored in subdirectories of the database directory
- The exception is tables in the default database which does not have its own directory
- The database directory is created under a top level directory specified by the following Hive property

  hive.metastore.warehouse.dir

  The default value for the property is usually /user/hive/warehouse or sometimes /app/hive/warehouse

DML Commands

CREATE DATABASE …

- You can override this default location for the new databse directory as shown

```
CREATE DATABASE someDatabase LOCATION '/path/subpath'
```

- You can add a descriptive comment to the database as follows (that will be show by the DESCRIBE command)

```
CREATE DATABASE someDatabase COMMENT 'some note'
```

DML Commands

DROP DATABASE …

Description

Remove the database. The default behavior is RESTRICT which means that the database is dropped only if it is empty. To drop the database with tables, use CASCADE.

Syntax

```
DROP DATABASE [IF EXISTS] database_name [RESTRICT|CASCADE];
```

Example

If database has no tables (RESTRICT is the default)…

```
DROP DATABASE IF EXISTS demo2;
```

If database has tables…

```
DROP DATABASE IF EXISTS demo2 CASCADE;
```

DML Commands

SHOW DATABASES

Description

Lists all of the already created databases defined in the metastore

Syntax

```
SHOW DATABASES;
```

If you have many databases defined you can restrict the listing using a regular expression

```
SHOW DATABASES LIKE ‘h.*’;
```

DML Commands

DESCRIBE DATABASES

Description

Shows the name of the database, its comment (if one has been set), and its root location on the filesystem or S3. EXTENDED also shows the database properties.

Syntax

```
DESCRIBE DATABASE [EXTENDED] db_name;
```

Example

```
DESCRIBE DATABASE cs595;
```

```
+----------+----------+--------------------------------------------------+-------------+-------------+-------------+
| db_name  | comment  |                    location                      | owner_name  | owner_type  | parameters  |
+----------+----------+--------------------------------------------------+-------------+-------------+-------------+
| cs595    |          | hdfs://ip-172-31-52-146.ec2.internal:8020/user/hive/warehouse/cs595.db | anonymous   | USER        |             |
+----------+----------+--------------------------------------------------+-------------+-------------+-------------+
```

DML Commands

USE …

Description

To make the default database something other than “default” do this. Sets the current database for all subsequent HiveQL statements.

To revert to the default database, use the keyword "default" instead of a database name.

To check which database is currently being used:
SELECT current_database();

Syntax

USE database_name;

USE DEFAULT;

What is a Hive Database Table?

- A Hive database table is just HDFS file or S3 object which is described to follow a given schema
- Here a schema is a specification of the names, types and groupings to which each record in the table must conform
- The schema and other table properties are maintained in the Hive metadata repository
- Unlike an RDBMS table an HDFS file (or S3 object) full of data could exist prior to its being indicated as a Hive table
  - We will talk more about this when discussing external tables
- If the schema for a table differs from the actual format of the data in a file Hive will not warn you on table creation
- It is only when you try to query the table (via a SELECT statement) that Hive will handle any divergences

How Do I Create a Hive Table?

- The Hive command to create tables follows some SQL conventions with extensions to support a wide range of options as to table format, where tables are stored etc.
- While creating a table, you can specify these aspects and more
  - The database with which the table is associated
  - Whether the table is internal or external
  - The name of the table being created (or included)
  - The table schema (columns and associated data types)
  - The columns used for partitioning the data into multiple files
  - The file format for data files
  - The HDFS directory where the data files are located

DML Commands

CREATE TABLE …

Description

Creates a table with the given name. An error is thrown if a table or view with the same name already exists. You can use IF NOT EXISTS to skip the error.

Syntax

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [column_constraint] [COMMENT col_comment], ...)]
[PARTITIONED BY (col_name data_type [COMMENT 'col_comment'], ...)]
[CLUSTERED BY (col_name, col_name,.......]
[COMMENT table_comment]
[ROW FORMAT row_format]
[FIELDS TERMINATED BY char]
[LINES TERMINATED BY char]
[LOCATION 'hdfs_path']
[STORED AS file_format]
```

DML Commands

CREATE TABLE …

Details

- EXTERNAL – Used to create external table
- ROW FORMAT – Specifies the format of the row.
- FIELDS TERMINATED BY – By default Hive use ^A (\001) field separator, To load a file that has a custom field separator like comma, pipe, tab use this option.
- PARTITION BY – Used to create partition data. Using this improves performance.
- CLUSTERED BY – Dividing the data into a specific number for buckets.
- LOCATION – You can specify the custom location where to store the data on HDFS or S3.
- STORED AS – to tell Hive what type of file to expect. By default this is 'TEXTFILE' but is could be 'ORC', 'PARQUET' or others

- Besides these, Hive also supports many optional clauses.

DML Commands

CREATE TABLE …

Here is a simple example that creates an **employee** table in the **emp** database with **id**, **name**, **age** and **email** columns

```
CREATE TABLE IF NOT EXISTS emp.employee (
 id int,
 name string,
 age int,
 email string )
 COMMENT 'Employee Table'
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ‘,’
 STORED AS TEXTFILE;
```

The table will be loaded from a comma-separated file so we use the ‘ROW FORMAT DELIMITED’ and ‘FIELDS TERMINATED BY’ optional clause to specify the custom (,) delimiter. The table will be stored as a sequence of characters (text).

•

DML Commands

CREATE TABLE …

- Here is a simple example of creating a table:

```
CREATE TABLE mydb.demo (foo INT, bar STRING);
```

- Creates a table called demo in the mydb database with two columns, the first being an integer and the other a string

DML Commands

CREATE TABLE …

- Unspecified defaults are assumed about this table as if the table was specified as follows…

```
CREATE INTERNAL TABLE mydb.demo (foo INT, bar STRING)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001'
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
 STORED AS TEXTFILE
 LOCATION ‘/user/hive/warehouse/mydb/demo’
```

- The table is assumed to be under the control of Hive (INTERNAL)
- The table is assumed to be just a plain old text file (TEXTFILE)
- The row format is records of delimited fields, where…
- Each field in the file is assumed terminated by a Ctrl-A
- Each array or struct member is assumed terminated by a CTRL-B
- Each map key is assumed separated from its value by a CTRL-C
- The table location is in the usual place for INTERNAL tables

DML Commands

CREATE TABLE …

- Here is a more elaborate example of creating a table:

```
CREATE TABLE IF NOT EXISTS mydb.employees (
  name            STRING COMMENT 'Employee name',
  salary          FLOAT  COMMENT 'Employee salary',
  subordinates    ARRAY<STRING> COMMENT 'Names of subordinates',
  deductions      MAP<STRING, FLOAT>
          COMMENT 'Keys are deductions names, values are percentages',
  address         STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
          COMMENT 'Home address‘
)
COMMENT 'Description of the table’;
```

- Notice that unlike a relational database tables, the fields are not only atomic types
- This violates the normal form criteria of no repeating groups and also no complex types

What Are Internal (Managed) Hive Tables?

- Hive controls the lifecycle of internal tables
- Hive stores the data for these tables in a subdirectory under the directory defined by
  - hive.metastore.warehouse.dir (e.g., */user/hive/warehouse*)
- When we drop an internal table Hive deletes the data in the table

What Are External Hive Tables?

- However, internal tables are less convenient for sharing with other tools
- For example, suppose we have data that is created and used primarily by *Pig* or other tools…
- But we want to run some queries against it, but not give Hive *ownership* of the data
- We can define an *external* table that points to that data, but doesn't take ownership of it

What Are External Hive Tables?

Example

- Suppose we are analyzing data from the stock markets
- Periodically, we ingest the data for NASDAQ and the NYSE and we want to study this data with many tools
- Let’s assume the data files are in the distributed filesystem directory */data/stocks*.

What Are External Hive Tables?
Example

- The following table declaration creates an *external* table that can read all the data files for this comma-delimited data in */data/stocks*

```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
  exchange        STRING,
  symbol          STRING,
  ymd             STRING,
  price_open      FLOAT,
  price_high      FLOAT,
  price_low       FLOAT,
  price_close     FLOAT,
  volume          INT,
  price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/data/stocks';
```

Note this is specified as having CSV row format

What Are External Hive Tables? Example

- The EXTERNAL keyword tells Hive this table is external
- And the LOCATION … clause is required to also tell Hive where it's located
- Because it's external, Hive does not assume it *owns* the data
- Therefore, dropping the table *does not* delete the data, although the *metadata* for the table will be deleted

What Are External Hive Tables? Example

- However, it’s important to note that the differences between managed and external tables are smaller than they appear at first
- Even for internal tables, you *know* where they are located, so you can use other tools, hadoop fs commands, etc., to modify and even delete the files in the directories for managed tables
- Hive may technically own these directories and files, but it doesn’t have full control over them
- Still, a general principle of good design to express intent
- If the data is shared between tools, then do creating an external table makes this explicit

Hive and S3

- Ensure the S3 bucket that you want to use with Hive only includes homogeneously-formatted files
  - Don't include a CSV file, Apache log, and tab-delimited file in the same bucket
- We need to tell Hive…
  - The format of the data so that when it reads our data it knows what to expect
  - Where to find our data
- Let's create a Hive table definition that references the data in S3:

```
CREATE EXTERNAL TABLE mydata (key STRING, value INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘,‘
LOCATION 's3://mys3bucket/path/subpath/';
```

Where is Your Hive Database?

- A Hive database is just an S3 bucket or an HDFS subdirectory
- Tables in the database are then just
  - S3 objects in a bucket
  - Or files within lower level subdirectories of an HDFS subdirectory
- The default HDFS directory under which Hive databases subdirectories are held is (for EMR) is:
  - /user/hive/warehouse
- Now if we say:
  - "create database cs595;"
- Then a new subdirectory will be created as follows:
  - /user/hive/warehouse/cs595.db

Where is Your Hive Database?

- And if we create a table for that database
  - “create table cs595.salaries;”
- Then a new subdirectory will be created:
  - /user/hive/warehouse/cs595.db/salaries
- And if you load data into the cs595.salaries table using
  - “LOAD DATA LOCAL INPATH '/home/hadoop/hql/Salaries.tsv' OVERWRITE INTO TABLE cs595.salaries;”
- Then a file would be created in the …salaries subdirectory:
  - /user/hive/warehouse/cs595.db/salaries/Salaries.tsv

Where is Your Hive Database?

- There is no default bucket name for Hive databases held in S3
- So we first need to create some bucket, say
  - s3://csp554-dbs/
- Now if we say:
  - “create database mydb location location 's3://csp554-dbs/mydb.db’
- Then (non-partitioned) tables for this database would have the format
  - s3://csp554-dbs/mydb.db/<someTableName>
  - Where this was the full name of an S3 object

What Data Types Can I Use to Specify Table Schemas?

- Hive supports many of the *primitive* data types you find in relational databases…
- As well as three *collection* data types that are rarely found in relational databases

Hive Primitive Data Types

- Integers
  - TINYINT—1 byte integer
  - SMALLINT—2 byte integer
  - INT—4 byte integer
  - BIGINT—8 byte integer
- Boolean type
  - BOOLEAN—TRUE/FALSE
- Floating point numbers
  - FLOAT—single precision
  - DOUBLE—Double precision
- Fixed point numbers
  - DECIMAL—a fixed point value of user defined scale and precision
- String types
  - STRING—sequence of characters in a specified character set
  - VARCHAR—sequence of characters in a specified character set with a maximum length
- Date and time types
  - TIMESTAMP — A date and time without a timezone
  - TIMESTAMP WITH LOCAL TIME ZONE — A point in time measured down to nanoseconds ("Instant" semantics)
  - DATE—a date
- Binary types
  - BINARY—a sequence of bytes

What Data Types Can I Use to Specify Table Schemas? Complex Types

- Hive supports several collection types
  - ARRAY
  - MAP
  - STRUCT

What Data Types Can I Use to Specify Table Schemas? Complex Types

- Most relational databases don’t support such collection types, because using them tends to break *normal form*.
- For example, in traditional data models, structs might be captured in separate tables, with foreign key relations between the tables
- A practical problem with breaking normal form is the greater risk of data duplication, leading to unnecessary disk space consumption and potential data inconsistencies
  - As duplicate copies can grow out of sync as changes are made
- However, in *Big Data* systems, a benefit of sacrificing normal form is higher processing throughput
- Scanning data off disks with minimal “head seeks” is essential when processing terabytes to petabytes of data
- Embedding collections in records makes retrieval faster with minimal seeks
- Navigating each foreign key relationship requires seeking across the disk, with significant performance overhead.

What Data Types Can I Use to Specify Table Schemas? Complex Types

- Here is a table declaration that demonstrates how to use complex types, an *employees* table in a Human Resources application

```
CREATE TABLE employees (
  name           STRING,
  salary         FLOAT,
  subordinates   ARRAY<STRING>,
  deductions     MAP<STRING, FLOAT>,
  address        STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>);
```

- Note that Java syntax conventions for *generics* are followed for the collection types
- For example, MAP<STRING, FLOAT> means that every key in the map will be of type STRING and every value will be of type FLOAT
- For an ARRAY<STRING>, every item in the array will be a STRING
- STRUCTs can mix different types, but the locations are fixed to the declared position in the STRUCT
  - street is always the first column in the struct and zip is always the last

What Data Types Can I Use to Specify Table Schemas? Complex Types

- Structs (named grouping of types)
  - Declare as…

    STRUCT<col_name : data_type, {col_name : data_type}*>
  - For example…

    STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>);
  - The elements within the array type can be accessed using the DOT (.) notation.
  - For example, for a column c of type STRUCT {a INT; b INT}, the a field is accessed by the expression c.a
- Maps (collection of key value pairs)
  - Declare as MAP <*key-type, value-type*>
  - For example MAP<STRING, FLOAT>
  - The elements are accessed using ['element name'] notation
  - For example in a map M comprising of a mapping from 'group' to id, the id value can be accessed using M['group']

What Data Types Can I Use to Specify Table Schemas? Complex Types

- Arrays (indexable lists)
  - Declare as ARRAY <*element-type*>
  - For example, ARRAY<STRING>
  - The elements in the array have to be in the same type
  - Elements can be accessed using the [n] notation where n is an index (zero based) into the array
  - For example, for an array A having the elements ['a', 'b', 'c'], A[1] returns 'b'.

What Are Hive Storage Formats?

- Hive can read and write HDFS files in a range of formats
- A file format is the way in which information is stored and encoded in a file
- Each of the ways a file is formatted or encoded can have its own advantages and disadvantages
- Some formats are more nearly standard and assume basic text encoding and delimiter separated columns
- Other formats are unique to Hadoop and have been developed to optimize query performance or storage

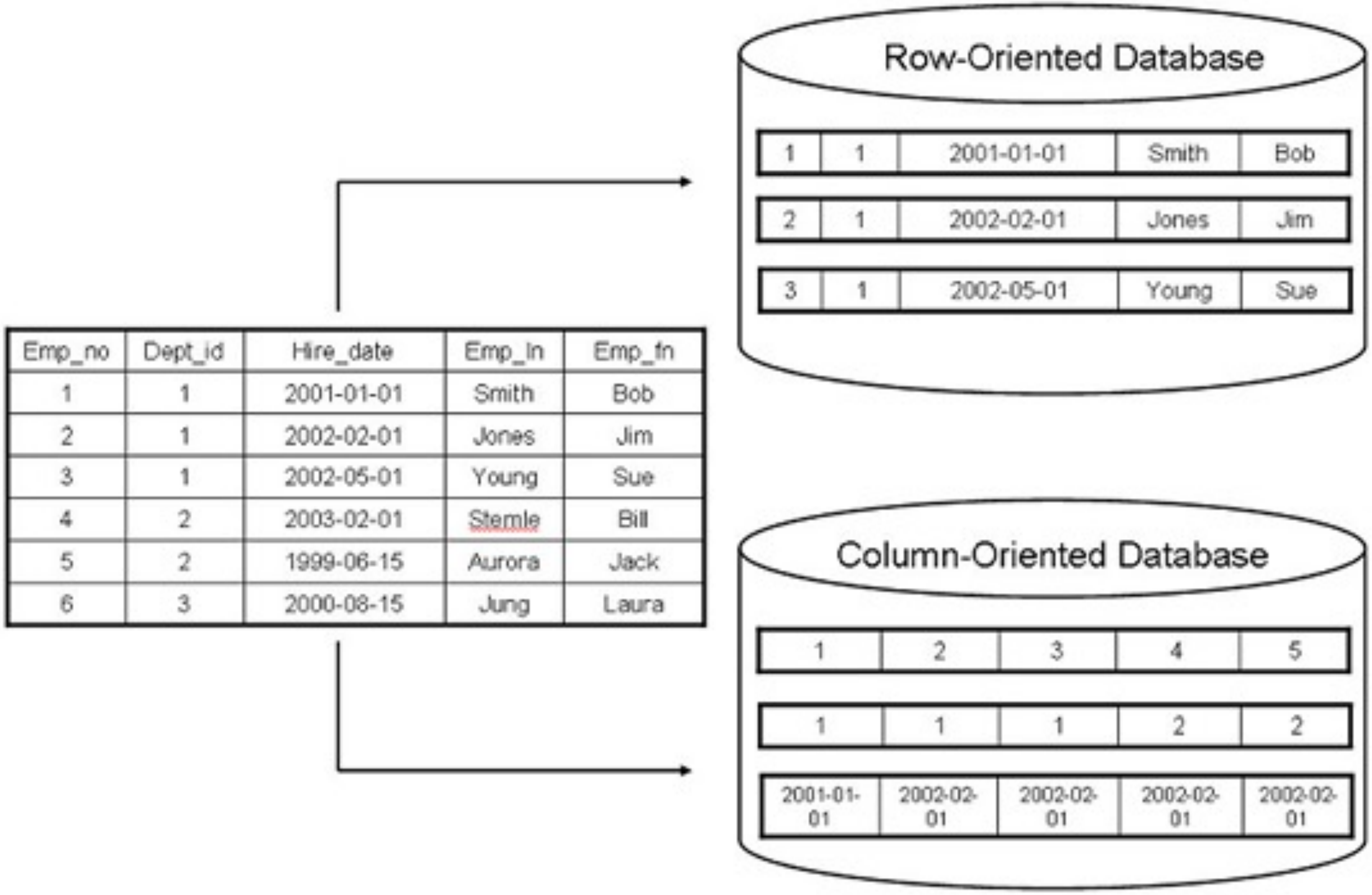
What Are Hive Storage Formats?

- Some storage formats are
  - TEXTFILE
    - STORE AS TEXTFILE
    - Default if no other is declared on table creation)
    - Use the DELIMITED clause to read delimited files
  - ORC, PARQUET, RCFILE
    - STORE AS ORCFILE | PARQUET, | RCFILE
    - Stores file rows in ways that optimizes query performance
    - Some of these formats also optimize for data compression
    - Not human readable and only created by Hive operations
  - Other formats (we will not discuss)
    - AVRO
    - SEQENTIALFILE

What is ORC Storage Format?

- ORC is a columnar file format designed for Hadoop workloads
- In row file format all the columns for a single record are stored together
- In column file format all the values for a column for all records are stored together

What is ORC Storage Format?



Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

Row-Oriented Database

1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue

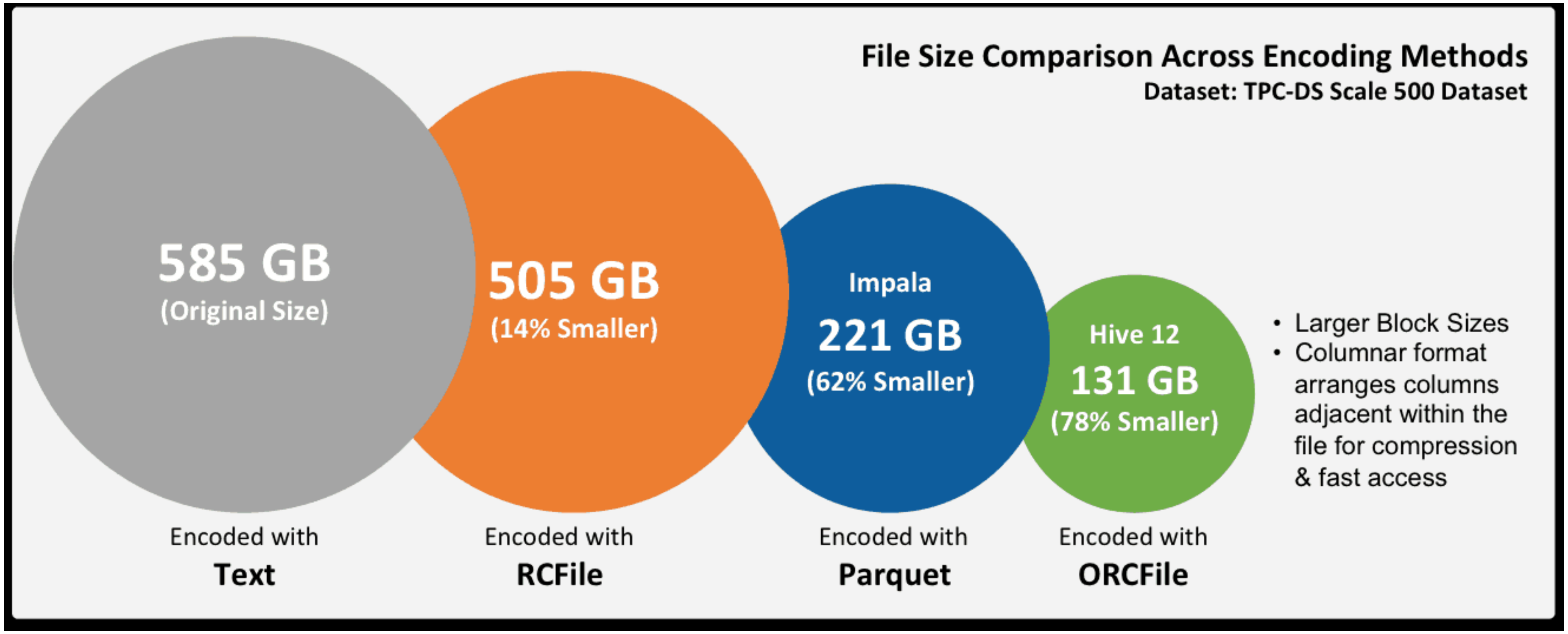
Column-Oriented Database

1	2	3	4	5
1	1	1	2	2
2001-01-01	2002-02-01	2002-02-01	2002-02-01	2002-02-01

What is ORC Storage Format?

- ORC format is useful if your query only requires values from a subset of columns
  - In row format you still need to read data one record (all columns)
  - In column format you can read a column in a sequential operation
- It is optimized for large streaming reads, but with indexes for finding required rows quickly
- Organizing data into columns also offers opportunities for compression
  - Imagine a column having two value “M” and “F”
  - This could be compressed using a bit map with 0 for “M” and 1 for “F”
  - Here eight columns of data could then be packed into one byte making reading that column require much less I/O

What is ORC Storage Format?



- Larger Block Sizes
- Columnar format arranges columns adjacent within the file for compression & fast access

What is the Text File Encoding of Data Values?

- You are familiar with text files delimited by commas, the so-called comma-separated values (CSVs)
- Or text files delimited by tabs, the so-called tab-separated values (TSVs)
- However, there is a drawback to both formats…
  - You have to be careful about commas or tabs embedded in text and not intended as field or column delimiters
- For this reason, Hive uses various control characters by default, which are less likely to appear in value strings

What is the Text File Encoding of Data Values?

Delimiter	Description
\n	For text files, each line is a record, so the line feed character separates records
^A (control-A)	Separates all fields (columns). Written using the octal code \001 when explicitly specified in CREATE TABLE statements
^B	Separate the elements in an ARRAY or STRUCT, or the key-value pairs in a MAP. Written using the octal code \002 when explicitly specified in CREATE TABLE statements
^C	Separate the key from the corresponding value in MAP key-value pairs. Written using the octal code \003 when explicitly specified in CREATE TABLE statements

What is the Text File Encoding of Data Values?

- Here is a table declaration with all the format defaults explicitly specified:

```
CREATE TABLE employees (
name STRING,
salary FLOAT,
subordinates ARRAY<STRING>,
deductions MAP<STRING, FLOAT>,
address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

What is the Text File Encoding of Data Values?

- Here is the format of one record we might find in the declared table

```
John Doe^A100000.0^AMary Smith^BTodd Jones^AFederal
Taxes^C.2^Bstate Taxes^C.05^BInsurance^C.1^A1 Michigan
Ave.^BChicago^BIL^B60600
```

What is the Text File Encoding of Data Values?

- Here is a table definition where the data will contain the more common comma-delimited fields
- Note that we do not indicate the data is stored in TEXTFILE format as this is the default

```
CREATE TABLE some_data (
first FLOAT,
second FLOAT,
third FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

- Use '\t' for tab-delimited fields.

What are Hive Data Partitions?

- Lets assume we have a file of the following format

ID	Name	Dept	Year
1	Joe	IT	2012
2	Jane	ECON	2012
3	Jill	FIN	2011
4	Jack	HR	2011
5	Sam	IT	2010

- Based on experience we know that the majority of our queries of this data will have the following form
    - SELECT ID, Name, Dept FROM table WHERE Year = 'someYear'

What are Hive Data Partitions?

- Now consider if the table were Terabytes long; we would still need to scan records for years of no interest
- But what if could organize our table into tables based on the value of the Year column; call these table partitions

ID	Name	Dept	Year
1	Joe	IT	2012
2	Jane	ECON	2012

ID	Name	Dept	Year
3	Jill	FIN	2011
4	Jack	HR	2011

ID	Name	Dept	Year
5	Sam	IT	2010

- Then each data based query would need to scan many few records and therefore execute more quickly

What are Hive Data Partitions?

- Recall these partition tables could still each be very large
- Is there a way to reduce the size of data we need to store for each partition?
- Recall that each partition holds records for some specific year… each value in the Year column is the same
- So a simple optimization is to name each partition something like: Year=2012 or Year=2011
- Hive will look at the name of each partition (kept as a subdirectory name) and use that in place of the repeated year value

Year=2012

ID	Name	Dept	~~Year~~
1	Joe	IT	~~2012~~
2	Jane	ECON	~~2012~~

Year=2011

ID	Name	Dept	~~Year~~
3	Jill	FIN	~~2011~~
4	Jack	HR	~~2011~~

Year=2010

ID	Name	Dept	~~Year~~
5	Sam	IT	~~2010~~

Some Intuition About Hive Partitions

- Let’s imagine that we work for a very large corporation…
- Our HR people often run queries with WHERE clauses that restrict the results to a state or city
- For example to query the names of employees in the state of Illinois

```
SELECT name from Employee
WHERE state = “Illinois”
```

- Or for example to query the names of employees in the state of Illinois in the city of Skokie

```
SELECT name from Employee
WHERE state = “Illinois” and city = ”Skokie”
```

Some Intuition About Hive Partitions

- If the Employee table is contained in one HDFS file or S3 object, Hive will have to scan and records for employees in other states than Illinois and/or other cities than Skokie
- As a result, imagining the table has hundreds of millions of records, is that queries are slower than necessary
- Why slower than necessary… because Hive mostly scans and skips records of no importance to users
- So the amount of I/O operations between the data and storage nodes is unnecessarily high
- And, some amount of compute activity, even occurring in parallel, is also wasted on comparing irrelevant records

What are Hive Data Partitions?

- Up to this point a table is stored in a file with the table’s name the database directory for that table
  - So if we have table T1 in database D1 we would see the directory
    /user/hive/warehouse/D1/T1
- In Hive partitioning, a each partition of a table is stored in a subdirectory where the parent directory has the Table’s name
- Each partition subdirectory is named corresponding to a particular value of partition column(s)
- When the table is queried, where possible, only required partitions of the table are searched reducing I/O and the time required by the query

Some Intuition About Hive Partitions

- An sensible optimization would be to divide the Employee table into multiple tables, one per state, or even one table per city within a state
  - Employee-Illinois
  - Employee-Maine
  - Employee-Illinois-Chicago
  - Employee-Illinois-Skokie
- Let’s call these new tables partitions, because that divide the original table into didtinct and separate parts
- Then, if we were interested in the names of employees from Illinois we could issue a query like this
  - SELECT name FROM Employee-Illinois
  - SELECT name FROM Employee-Illinois-Skokie

Some Intuition About Hive Partitions

- But this leads to one major problem… The query system is not aware of this more efficient grouping
- We need to use a different table name depending on our query…

  SELECT name from Employee-Illinois

  SELECT name from Employee-Maine
- Instead of the more general

  SELECT name from Employee

  WHERE state = <someStateName>
- That is, our query system is not partition aware and can't translate

  SELECT name from Employee WHERE state = "Illinois"

  Into SELECT name from Employee-Illinois
- So our queries are less general, but they are certainly more efficient

What are Hive Data Partitions?

- As an example consider the following table definition

```
CREATE TABLE employees (
   name        STRING,
   salary      FLOAT,
   city        STRING,
   state       STRING
 )
```

- Assume that the file holding data for this table is held in the HDFS directory

```
/user/hive/warehouse/some-database-name/employees
```

What are Hive Data Partitions?

- Now consider the following table definition where we partition on country and then state

```
CREATE TABLE employees (
  name        STRING,
  salary      FLOAT,
)
PARTITIONED BY (country STRING, state STRING);
```

Note, we do not mention partition attributes twice.
IF these attributes are arguments of the PARTITION BY keyword then the don't appear again in the schema definition

- You can see the country and state columns are no longer included in the CREATE TABLE definition…
- But are included in the partition definition

What are Hive Data Partitions?

- Partitioning tables changes how Hive structures the data storage.
- If we create this table in the mydb database, there will still be an *employees* directory for the table

  /user/hive/warehouse/mydb.db/employees
- But Hive will now create subdirectories reflecting the partitioning structure. For example:

  */user/hive/warehouse/mydb.db/*employees/country=CA/state=AB
  */user/hive/warehouse/mydb.db/*employees/country=CA/state=BC
  */user/hive/warehouse/mydb.db/*employees/country=US/state=AL
- The state directories will contain zero or more files for the employees in those states
- Once created the partition keys (country and state) behave like actual columns
- Because the country and state values are encoded in directory names there is no reason to have this data in the data files themselves
- Since these partition fields are only mentioned in the directory path, and not in the HDFS files themselves, we call then virtual fields

What are Hive Data Partitions?

- For example, the following query selects all employees in the state of Illinois in the United States:

```
SELECT * FROM employees WHERE country = 'US' AND state = 'IL’;
```

What are Hive Data Partitions?

- Perhaps the most important reason to partition data is for faster queries.
- In the example query, which limits the results to employees in Illinois, it is only necessary to scan the contents of *one* directory
- Even if we have thousands of country and state directories, all but one can be ignored
- For very large data sets, partitioning can dramatically improve query performance
  - But *only* if the partitioning scheme reflects common *range* filtering (e.g., by locations, timestamp ranges).
- When we add predicates to WHERE clauses that filter on partition values, these predicates are called *partition filter*
- Of course, if you need to do a query for all employees around the globe, you can still do it.
  - Hive will have to read every directory, but hopefully these broader disk scans will be relatively rare.

What are Hive Data Partitions?

- You can see the partitions that exist with the SHOW PARTITIONS command:

```
hive> SHOW PARTITIONS employees;

country=CA/state=AB
country=CA/state=BC
country=US/state=AL
country=US/state=AK
...
```

What are Hive Data Partitions?

- You can use partitioning with external tables
- In fact, you may find that this is your most common scenario for managing large production data sets
- The combination gives you a way to “share” data with other tools, while still optimizing query performance
- You also have more flexibility in the directory structure used, as you define it yourself
- We’ll see a particularly useful example in a moment.

What are Hive Data Partitions?

- Here’s how we might define an external partitioned Hive table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS log_messages (
  hms            INT,
  severity       STRING,
  server         STRING,
  process_id     INT,
  message        STRING)
PARTITIONED BY (year INT, month INT, day INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- We assume a day’s worth of log data is the correct size for a useful partition
- And finer grain queries over a day’s data will be fast enough

What are Hive Data Partitions?

- Recall that when we created the non-partitioned external table, a LOCATION … clause was required
- It isn't used for external partitioned tables
- Instead, an ALTER TABLE statement is used to add *each* partition separately
- It must specify a value for each partition key, the year, month, and day, in this case
- Here is an example, where we add a partition for January 2nd, 2012:

```
ALTER TABLE log_messages ADD PARTITION(year = 2012, month = 1, day
= 2) LOCATION '/data/log_messages/2012/01/02';
```

- Notice the partition directory naming conventions are up to us

What are Hive Data Partitions?

- If used incorrectly, partitioning can lead to performance degradation
- The key thing with Hive partitioning is not to over partition
  - Partitions increase the overhead in data loading and data retrieval
  - If you create a very large number of partitions with small chunk of data in each partition, you are more likely to have small files
  - Large number of small files is generally much slower in Hadoop than fewer, larger files
- Some of the best practices to consider when partitioning tables
  - Pick a column for partition key with low to medium Number of Distinct Values (NDVs)
  - Avoid partitions that are less than 1 GB (bigger is better)
  - You should avoid deep nesting as it can cause too many partitions and hence create very small files
    - When you use multiple columns for partition key, it will create a nested tree of subdirectories for each combination of partition key columns

DML

Modifying Data in Hive Tables

- LOAD
  - Append data into a Hive table from a (HDFS, local) file or S3 object
- INSERT … SELECT
  - Append data into a Hive table from a SELECT operation
- INSERT...VALUES (only transactional/ACID tables)
  - Used to insert data into Hive tables directly from SQL
- UPDATE (only transactional/ACID tables)
  - Used to update existing data in Hive tables directly from HQL
- DELETE (only transactional/ACID tables)
  - Used to delete existing data from Hive tables directly in HQL

DML

LOAD …

- Load data into a table from a Linux or HDFS file or S3 bucket into a Hive table

- Syntax

  LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

filepath – Supports absolute and relative paths. If you use optional clause LOCAL the specified filepath would be referred from the server where hive beeline is running otherwise it would use the HDFS path

LOCAL – Use LOCAL if you have a file in the server where beeline is running

OVERWRITE – It deletes the existing contents of the table and replaces with the new content

PARTITION – Loads data into specified Hive table partition

-

How Do We Load Data Into Hive Tables?

- The LOAD command has the following general syntax

```
LOAD DATA [LOCAL] INPATH 'filepath'
[OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)];
```

- The target being loaded to can be a table or a partition
- If the table is partitioned, then one must indicate a specific partition of the table
    - By specifying values for all of the partitioning column
- If the OVERWRITE keyword is used then the contents of the target table (or partition) will be deleted and replaced by the files referred to by *filepath*
    - Otherwise the files referred by *filepath* will be added to the table

How Do We Load Data Into Hive Tables?

- If the keyword LOCAL is specified, then:
  - A ***local (Linux, Windows, MacOS) file is copied*** to the table or partition
- If the keyword LOCAL is *not* specified
  - An ***HDFS or S3 file is moved*** to the table or partition

How Do We Load Data Into Hive Tables?

- Here is an example of loading a file from the local file system into a specific partition of a Hive database table

```
LOAD DATA LOCAL INPATH '$home/jrosen/california-employees'
OVERWRITE INTO TABLE employees
PARTITION (country = 'US', state = 'CA');
```

- This command will first create the directory for the partition if it doesn’t already exist…
- And then it will copy the data to it; the source file will not be moved as we use the LOCAL option
- If the target table is not partitioned, then you omit the PARTITION clause
- Hive does not verify that the data you are loading matches the schema for the table
- However, it will verify that the file format matches the table definition
- For example, if the table was created with SEQUENCEFILE storage, the loaded files must be sequence files.

How Do We Load Data Into Hive Tables?

- Notice that on limitation of the LOAD command is the source file is required to be of the same format as the target Hive table
- This raises a several questions
    - How can I load data into a table not in TEXTFILE format, say in ORC format
    - How can I load data from one Hive table to another
    - How can I load data into a partitioned table without needing to identify each partition manually
    - How can I load data from a non-partitioned file or table to a partitioned table
        - The non-partitioned data source includes partition columns but the partition table must not include them

How Do We Load Data Into Hive Tables?

- The INSERT statement lets you load data into a table from a query

```
INSERT OVERWRITE TABLE employees
PARTITION (country = 'US', state = 'OR')
SELECT * FROM staged_employees se
WHERE se.cnty = 'US' AND se.st = 'OR';
```

- With OVERWRITE, any previous contents of the partition (or whole table if not partitioned) are replaced
- If you drop the keyword OVERWRITE or replace it with INTO, Hive appends the data

- This example suggests one common scenario where this feature is useful…
- Data has been staged in a directory, exposed to Hive as an external table, and now you want to put it into the final, partitioned table
- A workflow like this is also useful if you want the target table to have a different record format than the source table (e.g., a different field delimiter)
- It is also the primary way data can be moved from a table or file having one storage format to one having a different storage format

How Do We Load Data Into Hive Tables?

- Hive supports two partitioning models: static and dynamic
- When either is used, queries are run against only a portion of the data, providing significant performance gains.
- But what type of partitioning should you use, and when?
- **Static Partitioning**—Used when the values for partition columns are known well in advance of loading the data into a Hive table
- **Dynamic Partitioning**—Used when the values for partition columns are known only during loading of the data into a Hive table

DML

INSERT … SELECT

- INSERT OVERWRITE will overwrite any existing data in the table or partition
- NSERT INTO will append to the table or partition, keeping the existing data intact

- Syntax

```
INSERT OVERWRITE TABLE tablename1
[PARTITION (partcol1=val1, partcol2=val2 ...)
[IF NOT EXISTS]] select_statement1 FROM from_statement;

INSERT INTO TABLE tablename1
[PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1 FROM from_statement;
```

How Do We Load Data Into Hive Tables?

- For example, let's say you have a huge dataset accumulated over many years
- You might be concerned about query performance against the Hive table created for this dataset
- You could use one or more columns to partition the table created for this underlying data
- One possibility would be to partition the data by year
- If you already know that the data is nicely segregated by year before it's loaded into Hive, static partitioning can be used

How Do We Load Data Into Hive Tables?

- However, what if the year values are not known in advance of loading the data, and you want to avoid digging into the data to extract year values
- In this case, dynamic partitioning is the best approach.
- The values of dynamic partition columns are known only at execution time of the data-load query
- Hive automatically takes care of not only creating the partitions, but also loading the data into the correct partitions
- No manual user intervention is required.
- Partition columns are specified by a “PARTITIONED BY” clause in “CREATE TABLE” query
- For each distinct set of partition column values, Hive creates a unique HDFS path and loads data into it

How Do We Load Data Into Hive Tables?

- To enable dynamic partitioning you must set some Hive properties to non-default value

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=non-strict;
```

How Do We Load Data Into Hive Tables?

- Now let’s assume we have a table defined as follows:
- Create the table with partitions:

```
CREATE TABLE patents (
citing_patent          INT,
cited_patent           INT,
assignee               STRING,
companyname            STRING,
publication_date       STRING)

PARTITIONED BY (
year  INT,
month INT,
day   INT);
```

How Do We Load Data Into Hive Tables?

- To load the data into the "patents" table (and overwrite existing records) we specify an INSERT command as follows for some external table

```
INSERT OVERWRITE TABLE patents
PARTITION (year, month, day)
SELECT citing, cited, name, company, year, month, day
FROM patents_raw_data;
```

- You could append to the table as follows

```
INSERT INTO TABLE patents
PARTITION (year, month, day)
SELECT citing, cited, name, company, year, month, day
FROM patents_raw_data;
```

Note, this SELECT must (1) include the partition key columns, (2) these columns must be in the same order as they are listed in the PARTITION and (3) these columns must be listed last

How Do We Load Data Into Hive Tables?

- The table, “patents_raw_data”, is an external table, which points to patent raw data
- The order of the partition columns specified in the “SELECT” clause is in exactly the same order as the partition columns specified in the “PARTITIONED BY” clause in create table query
- Also, the columns year, month, and day are purposefully specified at the very end in the “SELECT” clause
  - The is required for dynamic partitioning to work
- Hive splits the data into multiple partitions by year, month, and day values
- It also updates the Hive metastore automatically without explicit user intervention.

How Query Data From Hive Tables?

- The SELECT operator outlined here supports queries of Hive databases

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
 [LIMIT number]
```

- It has many aspects that are similar to the equivalent operator defined in the SQL-92 standard for relational databases
- This facilitates its use by analysts who are familiar with that paradigm

How Query Data From Hive Tables?

Examples

SELECT * FROM sales WHERE amount > 10 AND region = "US”

The following query returns 5 arbitrary customers

SELECT * FROM customers LIMIT 5

How Query Data From Hive Tables?

- In general, a SELECT query scans the entire table
- If a table created using the PARTITIONED BY clause, a query can do **partition pruning** and scan only a fraction of the table related to partitions specified by the query
- Hive currently does partition pruning if the partition predicates are specified in the WHERE clause
  - Or the ON clause in a JOIN

How Query Data From Hive Tables?

- If table page_views is partitioned on column date…
- The following retrieves rows for just days between 2008-03-01 and 2008-03-31

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
```

How Query Data From Hive Tables?

- When you select columns that are one of the collection types, Hive uses JSON (JavaScript Object Notation) syntax for the output
- First, let's select an ARRAY, where a comma-separated list surrounded with […] is used
  - Note that STRING elements of the collection are quoted

Computing With Column Values

- Not only can you SELECT columns in a table…
- But you can manipulate column values using function calls and arithmetic expressions

```
SELECT upper(name), salary, deductions["Federal Taxes"],
round(salary * (1 - deductions["Federal Taxes"])) FROM employees;
```

How Query Data From Hive Tables?

- A special kind of function is the *aggregate* function that returns a single value resulting from some computation over many rows
- Perhaps the two best known examples are count, which counts the number of rows and avg, which returns the average value of the specified column values
- Here is a query that counts the number of employees, averages their salaries and provides the min and max salaries

```
SELECT count(*), avg(salary), min(salary), max(salary)
FROM employees;
```

How Query Data From Hive Tables?

- While SELECT clauses select columns, WHERE clauses are filters; they select which records to return

```
SELECT * FROM employees
WHERE country = 'US' AND state = 'CA';

SELECT e.* FROM
    (SELECT name, salary, deductions["Fed Taxes"] as ded,
      salary * (1 - deductions["Fed Taxes"]) as salary_minus_fed_taxes
      FROM employees) e
WHERE round(e.salary_minus_fed_taxes) > 70000;
```

How Query Data From Hive Tables?

- Hive supports the classic SQL JOIN statement, but only *equi-joins* are permitted
- INNER JOIN
- OUTER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN

Only equi-join allowed

```
SELECT a.ymd, a.price_close, b.price_close
FROM stocks a JOIN stocks b ON a.ymd = b.ymd
WHERE a.symbol = 'AAPL' AND b.symbol = 'IBM';
```

Row Level ACID Transactions in Hive

- Imagine that you want to update a row (record) in an HDFS file using Hive
- Lets refer to the updating of a single row as a “row level transaction”
- What is one significant problem that might occur during a row level transaction…
  - Assume the row is in process of being updated by a writer (but not completely updated)
  - Let a reader come along and before the row is completely updated with new values, read the row
  - So the reader is given a row the is inconsistent, and has some original and some new values
- So what properties should a transaction have to prevent inconsistent reads?

- These are Atomicity, Consistency, Isolation, and Durability or ACID properties…

1. Atomicity means, a transaction should complete successfully or else it should fail completely i.e. it should not be left partially
2. Consistency ensures that any transaction will bring the database from one valid state to another state
3. Isolation states that every transaction should be independent of each other i.e. one transaction should not affect another
4. Durability states that if a transaction is completed, it should be preserved in the database even if the machine state is lost or a system failure might occur

ACID Transaction Scope and Limits

The different row-level transactions available in Hive are as follows:

1. Insert
2. Delete
3. Update

There are numerous limitations with the present transactions available in Hive

1. Only ORC file format is supported
2. Tables need to be bucketed in order to support transactions
3. External tables cannot be made ACID tables

How Do We Insert Data into a Hive Table

The INSERT...VALUES statement can be used to insert data into tables directly

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

Where values_row is:
  (value [, value ...])
  where a value is either null or any valid SQL literal

- Values must be provided for every column in the table. The standard SQL syntax that allows the user to insert values into only some columns is not yet supported
- To mimic the standard SQL, nulls can be provided for columns the user does not wish to assign a value to
- Dynamic partitioning is supported
- If the table being inserted into supports ACID and a transaction manager that supports ACID is in use, this operation will be auto-committed upon successful completion

How Do We Insert Data into a Hive Table

```
CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3, 2))
CLUSTERED BY (age) INTO 2 BUCKETS STORED AS ORC;


INSERT INTO TABLE students
VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32, 2.32);

---
CREATE TABLE pageviews (userid VARCHAR(64), link STRING, came_from STRING)
PARTITIONED BY (datestamp STRING)
CLUSTERED BY (userid) INTO 256 BUCKETS
STORED AS ORC;


INSERT INTO TABLE pageviews PARTITION (datestamp = '2014-09-23')
VALUES ('jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);


INSERT INTO TABLE pageviews PARTITION (datestamp)
VALUES ('tjohnson', 'sports.com', 'finance.com', '2014-09-23'),
('tlee', 'finance.com', null, '2014-09-21');
```

How Do We Update Data into a Hive Table

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

Updates can only be performed on tables that support ACID

- The referenced column must be a column of the table being updated
- Only rows that match the WHERE clause will be updated
- Partitioning columns cannot be updated
- Bucketing columns cannot be updated

How Do We Update Data into a Hive Table

```
CREATE TABLE hello_acid (key int, value int) PARTITIONED BY
(load_date date)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC
TBLPROPERTIES ('transactional'='true’);

INSERT INTO hello_acid partition (load_date='2016-03-03') VALUES
(3, 3);

UPDATE hello_acid SET value = 10 WHERE key = 3;
```