Hive

Agenda

- ✓ Motivation for Hive
  - What and Why
- ✓ Hive
  - Overview
  - Architecture
  - Hive Metastore
  - Features
  - HiveQL
  - UDF
- ✓ LAB

Hive: Motivation

- **MapReduce code is typically written in Java**
  - Although it can be written in other languages using Hadoop Streaming
- **Requires:**
  - A programmer
  - Who is a *good* Java programmer
  - Who understands how to think in terms of MapReduce
  - Who understands the problem they're trying to solve
  - Who has enough time to write and test the code
  - Who will be available to maintain and update the code in the future as requirements change

Hive: Motivation(Contd.)

- **Many organizations have only a few developers who can write good MapReduce code**
- **Meanwhile, many other people want to analyze data**
  - Business analysts
  - Data scientists
  - Statisticians
  - Data analysts
- **What's needed is a higher-level abstraction on top of MapReduce**
  - Providing the ability to query the data without needing to know MapReduce intimately
  - Hive and Pig address these needs

Hive: Introduction

- **Hive was originally developed at Facebook**
  - Provides a very SQL-like language
  - Can be used by people who know SQL
  - Under the covers, generates MapReduce jobs that run on the Hadoop cluster
  - Enabling Hive requires almost no extra work by the system administrator
- **Hive is now a top-level Apache Software Foundation project**

The Hive Data Model

- **Hive 'layers' table definitions on top of data in HDFS**
- **Tables**
  - Typed columns (int, float, string, boolean and so on)
  - Also array, struct, map (for JSON-like data)
- **Partitions**
  - e.g., to range-partition tables by date

Hive Data Types

- **Primitive types:**
  - `TINYINT`
  - `SMALLINT`
  - `INT`
  - `BIGINT`
  - `FLOAT`
  - `BOOLEAN`
  - `DOUBLE`
  - `STRING`
  - `BINARY` (available starting in CDH4)
  - `TIMESTAMP` (available starting in CDH4)
- **Type constructors:**
  - `ARRAY < primitive-type >`
  - `MAP < primitive-type, data-type >`
  - `STRUCT < col-name : data-type, ... >`

The Hive Metastore

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

Hive Data: Physical Layout

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

The Hive Shell

- To launch the Hive shell, start a terminal and run

```
$ hive
```

- Results in the Hive prompt:

```
hive>
```

Hive Basics: Creating Tables

```
hive> SHOW TABLES;

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

hive> DESCRIBE shakespeare;
```

Loading Data Into Hive

- **Data is loaded into Hive with the `LOAD DATA INPATH` statement**
  - Assumes that the data is already in HDFS

```
LOAD DATA INPATH "shakespeare_freq" INTO TABLE shakespeare;
```

- **If the data is on the local filesystem, use `LOAD DATA LOCAL INPATH`**
  - Automatically loads it into HDFS in the correct directory

Basic SELECT Queries

- Hive supports most familiar SELECT syntax

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

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

Joining Tables

- **Joining datasets is a complex operation in standard Java MapReduce**
  - We saw this earlier in the course
- **In Hive, it's easy!**

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

Storing Output Results

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

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

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

Using User-Defined Code

- **Hive supports manipulation of data via User-Defined Functions (UDFs)**
  - Written in Java
- **Also supports user-created scripts written in any language via the `TRANSFORM` operator**
  - Essentially leverages Hadoop Streaming
  - Example:

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

UDF - How to

```
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public class Strip extends UDF {
  private Text result = new Text();

  public Text evaluate(Text str) {
    if (str == null) {
      return null;
    }
    result.set(StringUtils.strip(str.toString()));
    return result;
  }

  public Text evaluate(Text str, String stripChars) {
    if (str == null) {
      return null;
    }
    result.set(StringUtils.strip(str.toString(), stripChars));
    return result;
  }
}
```

A UDF must satisfy the following two properties:

1. A UDF must be a subclass of `org.apache.hadoop.hive.ql.exec.UDF`.
2. A UDF must implement at least one `evaluate()` method.

UDF - How to

To use the UDF in Hive, we need to package the compiled Java class in a JAR file (you can do this by typing `ant hive` with the book's example code) and register the file with Hive:

```
ADD JAR /path/to/hive-examples.jar;
```

We also need to create an alias for the Java classname:

```
CREATE TEMPORARY FUNCTION strip AS 'com.hadoopbook.hive.Strip';
```

The `TEMPORARY` keyword here highlights the fact that UDFs are only defined for the duration of the Hive session (they are not persisted in the metastore). In practice, this means you need to add the JAR file, and define the function at the beginning of each script or session.

The UDF is now ready to be used, just like a built-in function:

```
hive> SELECT strip('  bee  ') FROM dummy;
bee
hive> SELECT strip('banana', 'ab') FROM dummy;
nan
```

Notice that the UDF's name is not case-sensitive:

```
hive> SELECT STRIP('  bee  ') FROM dummy;
bee
```

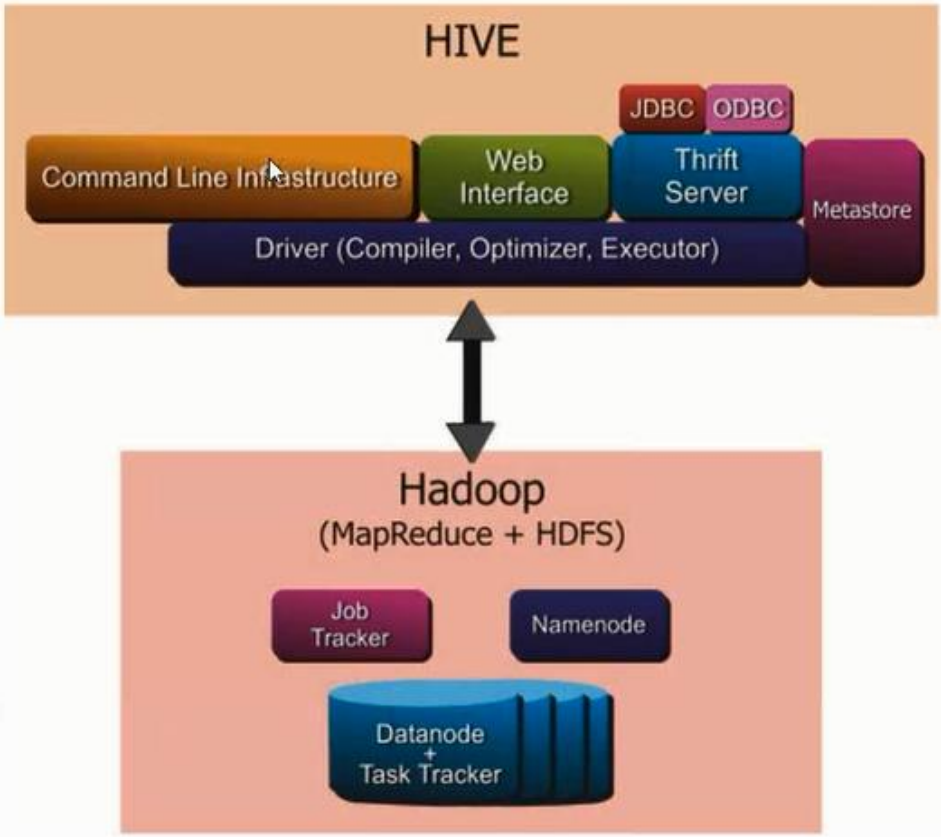
Hive - Limitations

- **Not all 'standard' SQL is supported**
  - Subqueries are only supported in the `FROM` clause
    - No correlated subqueries
- **No support for `UPDATE` or `DELETE`**
- **No support for `INSERT`ing single rows**

Choosing between Pig & Hive

- **Typically, organizations wanting an abstraction on top of standard MapReduce will choose to use either Hive or Pig**
- **Which one is chosen depends on the skillset of the target users**
  - Those with an SQL background will naturally gravitate towards Hive
  - Those who do not know SQL will often choose Pig
- **Each has strengths and weaknesses; it is worth spending some time investigating each so you can make an informed decision**
- **Some organizations are now choosing to use both**
  - Pig deals better with less-structured data, so Pig is used to manipulate the data into a more structured form, then Hive is used to query that structured data

HIVE - Architecture

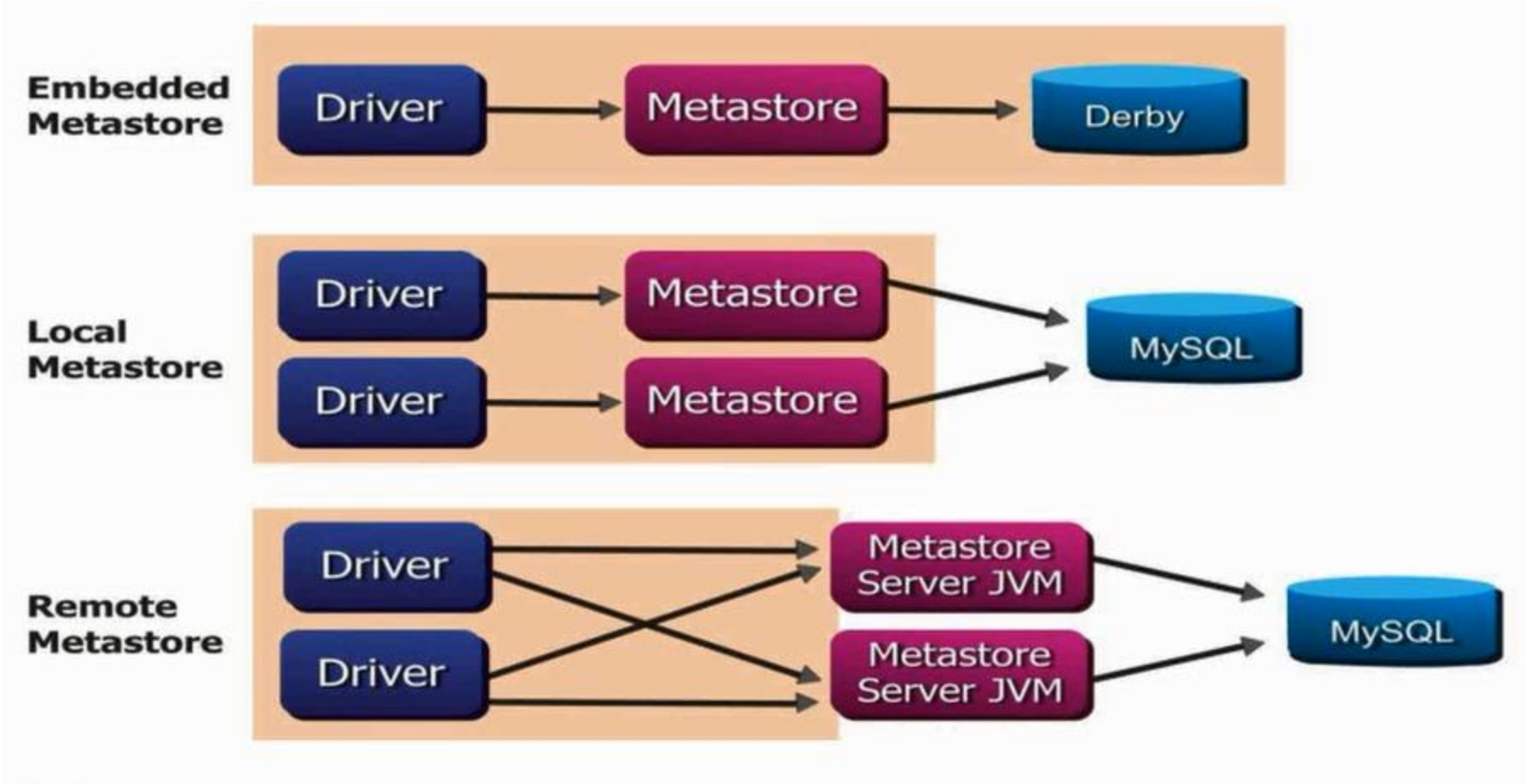


CLI – This is the default service and is most common way of accessing hive

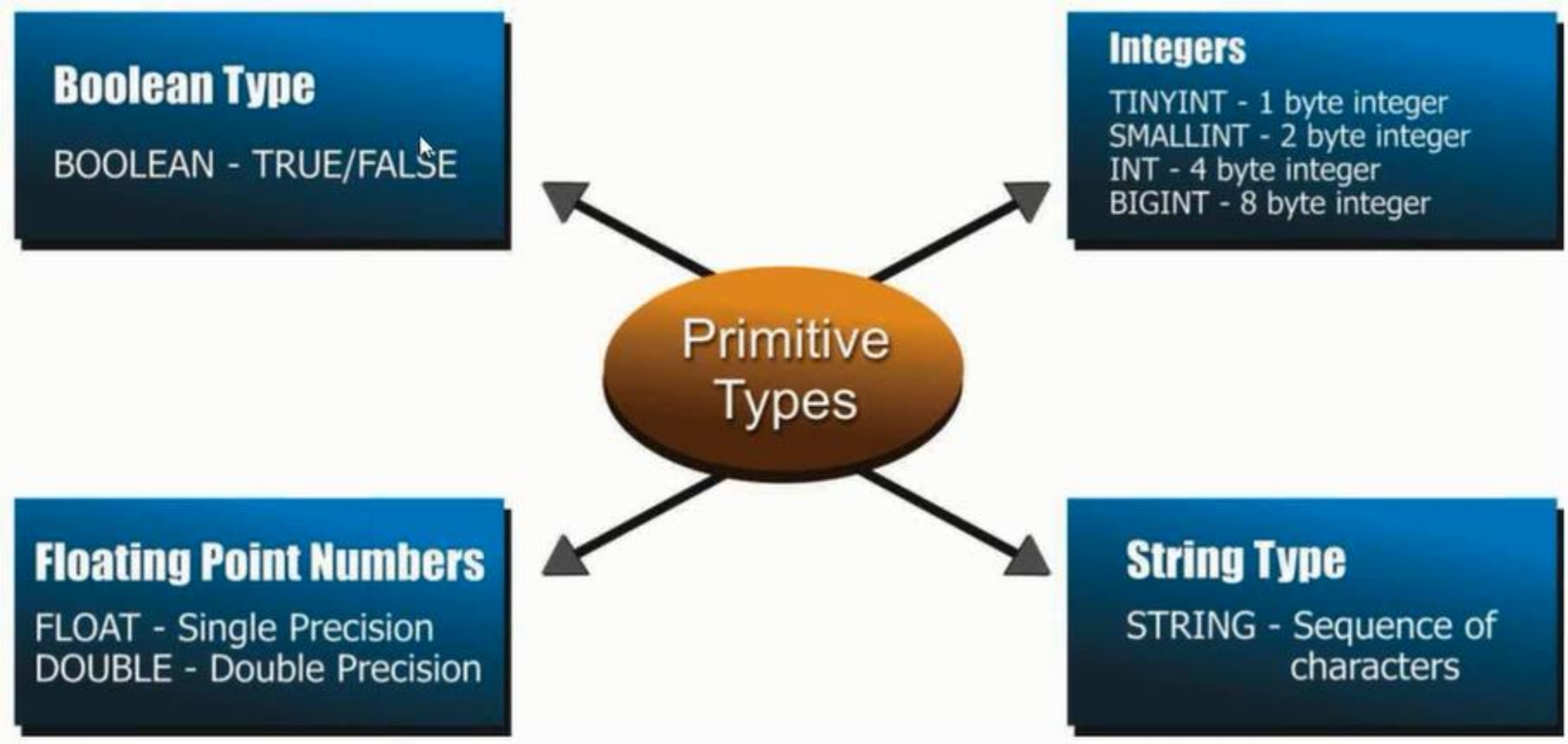
Hiveserver – Runs hive as a server exposing a Thrift Service, enabling access from a range of clients written in different language

hwi – Hive web interface

HIVE - Metastore



HIVE – Data Type



HIVE – Primitive Types

The Types are organized in the following hierarchy (where the parent is a super type of all the children instances):

Type

- Primitive Type
  - Number
    - DOUBLE
      - BIGINT
        - INT
          - TINYINT
      - FLOAT
        - INT
          - TINYINT
  - STRING
  - BOOLEAN

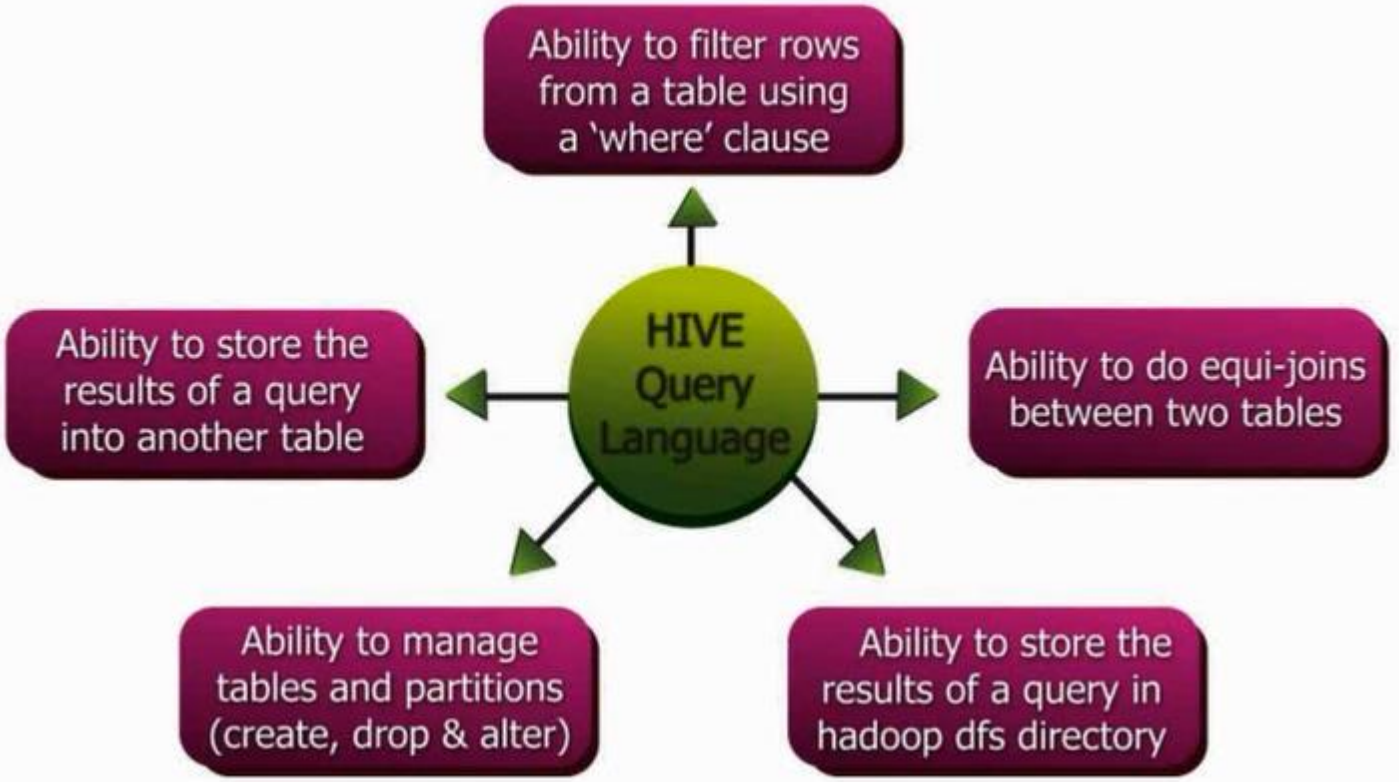
HIVE – Complex Types

Complex Types can be built up from primitive types and other composite types using the following three operators:

Type	Description	Literal syntax examples
STRUCT	Analogous to a C `struct` or an "object." Fields can be accessed using the "dot" notation. For example, if a column name is of type STRUCT {`first` STRING; `last` STRING}, then the first name field can be referenced using `name.first`.	`struct('John', 'Doe')`
MAP	A collection of key-value tuples, where the fields are accessed using array notation (e.g., ['key']). For example, if a column name is of type MAP with key→value pairs `'first'`→`'John'` and `'last'`→`'Doe'`, then the last name can be referenced using `name['last']`.	`map('first', 'John', 'last', 'Doe')`
ARRAY	Ordered sequences of the *same* type that are indexable using zero-based integers. For example, if a column name is of type ARRAY of strings with the value `['John', 'Doe']`, then the second element can be referenced using `name[1]`.	`array('John', 'Doe')`

HIVE – Ability of Hive Query Language

Hive Query Language provides the basic SQL-like operations



Differences with Traditional RDBMS

- ✓ **Schema on Read vs Schema on Write**
  - ✓ **Hive does not verifies the data when it is loaded**, but rather when a query is issued.
  - ✓ 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 file copy or move.
- ✓ **No Updates, Transactions and Indexes.**

Configuring HIVE

Hive automatically stores and manages data for users
–<install path>/hive/warehouse

HIVE-SITE.XML
- ConnectionURL
- WAREHOUSE.DIR

Metastore options
–Hive comes with Derby, a lightweight and embedded sql
–Can configure any other database as well e.g.MySQL

HIVE Data Models

- Databases
  - Namespaces
- Tables
  - Schemas in namespaces
- Partitions
  - How data is stored in HDFS
  - Grouping data bases on some column
  - Can have one or more columns
- Buckets or Clusters
  - Partitions divided further into buckets bases on some other column
  - Use for data sampling

Database

```
CREATE DATABASE IF NOT EXISTS FINANCIAL ;

USE FINANCIAL;

SHOW DATABASES;

SHOW DATABASES LIKE 'H.*';

DESCRIBE DATABASE FINANCIAL;
```

Type of Table

Managed Table

- Data is controlled by HIVE
- Create a directory for the data on HDFS
- Dropping the table will delete the data as well

External Table

- Hive does not delete the table (or hdfs files) even when the tables are dropped. It leaves the table untouched and only metadata about the tables are deleted

Working with HIVE

Managed Table

```
CREATE TABLE employees_managed(name STRING,salary INT,state STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

LOAD DATA LOCAL INPATH '/home/cloudera/Training/Hive/employees' INTO TABLE employees_managed

DESCRIBE employees_managed;

DESCRIBE extended employees_managed;

DROP TABLE employees_managed;
```

External Table

External Table

```
CREATE EXTERNAL TABLE employees_external(name STRING,salary INT,state
STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

LOAD DATA LOCAL INPATH '/home/cloudera/Training/Hive/employees' INTO TABLE
employees_managed

DROP TABLE employees_managed;
```

Load Data into Hive Table

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Training/Hive/Hive/employee' INTO TABLE EMPLOYEES;
Copying data from file:/home/cloudera/Training/Hive/Hive/employee
Copying file: file:/home/cloudera/Training/Hive/Hive/employee/000000_0
Loading data to table:hiveclass2feb.employees
```

Partition and Buckets

- Table organized into partitions
- Divides a table into coarse-grained parts based on partition column
- A table can be partitioned in multiple dimensions
- Partitioning helps to do queries faster on slices of the data

- Tables/Partitions can be further subdivided into Clusters
- Gives an additional structure to the data for more efficient queries

Create Table - Partitioned

Partitioned Table

```
CREATE TABLE logs(errortype STRING,errordesc STRING) PARTITIONED BY(dt
STRING, class STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' ;

LOAD DATA LOCAL  INPATH '/home/cloudera/Training/Hive/sampleclass6Feb3.log'
INTO TABLE logs PARTITION(dt='2012-02-03',class='sampleclass6');


SHOW PARTITIONS logs;

ALTER TABLE logs  add PARTITION(dt='2012-02-05',class='sampleclass6');
```

Bucketing

- Bucketing is helpful for 2 reasons
  - Enables more efficient queries
  - Makes sampling more efficient
- Hash(column) MOD(number of buckets) – evenly distributed

Bucketing

```
CREATE TABLE employees_bucket (name STRING,salary INT,state STRING)
PARTITIONED BY(country STRING) CLUSTERED BY  (state) INTO 4 BUCKETS
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

set hive.enforce.bucketing =true;
```

INSERTING

```
INSERT OVERWRITE TABLE employees_bucket PARTITION(country='USA')
SELECT name,salary,state from financials.employees_external;
```

SAMPLING

```
SELECT * FROM employees_bucket TABLESAMPLE(BUCKET 2 OUT OF 4);
```

Queries

Select

–Select count(*) from employees;

Aggregation

–Select count (DISTINCT name) from employees;

Grouping

–Select country, sum(salary) from employees group by country

Managing Outputs

- Inserting Output into another table

–INSERT OVERWRITE TABLE salary
SELECT country,salary
from employees_managed)

- Inserting into local file

–INSERT OVERWRITE LOCAL DIRECTORY'tmp/resuts' SELECT * from employees_managed

Joins

```
INSERT OVERWRITE TABLE pv_users
    SELECT pv.*,u.gender, u.age
    FROM user u JOIN page_view pv ON (pv.userid =u.id)
    WHERE pv.date = '200803-03';

INSERT OVERWRITE TABLE pv_users
    SELECT pv.* u.gender, u.age
    FROM user u FULL OUTER JOIN page _view pv ON (pv.userid =
    u.id)
    WHERE pv.date – '2008-03-03';
```

Hive Commands

```
create external table employee_external(name string,salary int,state string,country string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\t';


LOAD DATA LOCAL INPATH '/home/cloudera/Training/Hive/employees' INTO table employee_external;

CREATE TABLE SALARYTABLE (name STRING, salary INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

INSERT OVERWRITE TABLE SALARYTABLE select name,salary from employee_external;

INSERT OVERWRITE LOCAL DIRECTORY '/home/cloudera/Training/salary'select * from SALARYTABLE;

CREATE TABLE employeeloc (name STRING, state STRING, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

FROM employee_external emp
INSERT OVERWRITE TABLE SALARYTABLE select name,salary
INSERT OVERWRITE TABLE employeeloc select name,state,country
;

hive.cli.print.header
```

Multi Table Inserts

```
FROM employee_managed emp

    INSERT OVERWRITE TABLE employee_partition PARTITION
    (country='US")
    Select * where emp .country ='US'

    INSERT OVERWRITE TABLE employee_partition PARTITION
    (country='INDIA")
    Select * where emp .country ='INDIA'

    INSERT OVERWRITE TABLE salary select salary;
```

Managing Tables

- Altering a table

```
ALTER TABLE old_table_name RENAME TO new_table_name
ALTER TABLE tab1 ADD COLUMNS (c1 INT COMMENT ‘a new int
column’, c2 STRING DEFAULT ‘deL val’);
```

- Dropping a table or a parittion

```
DROP TABLE pv_users;
ALTER TABLE pv_users DROP PARTITION (ds=‘2008-0-08’)
```

Hive Script & Batch

```
Hive –f <script path>

Hive –e ‘select * from employees’
```