Hive

Training Session

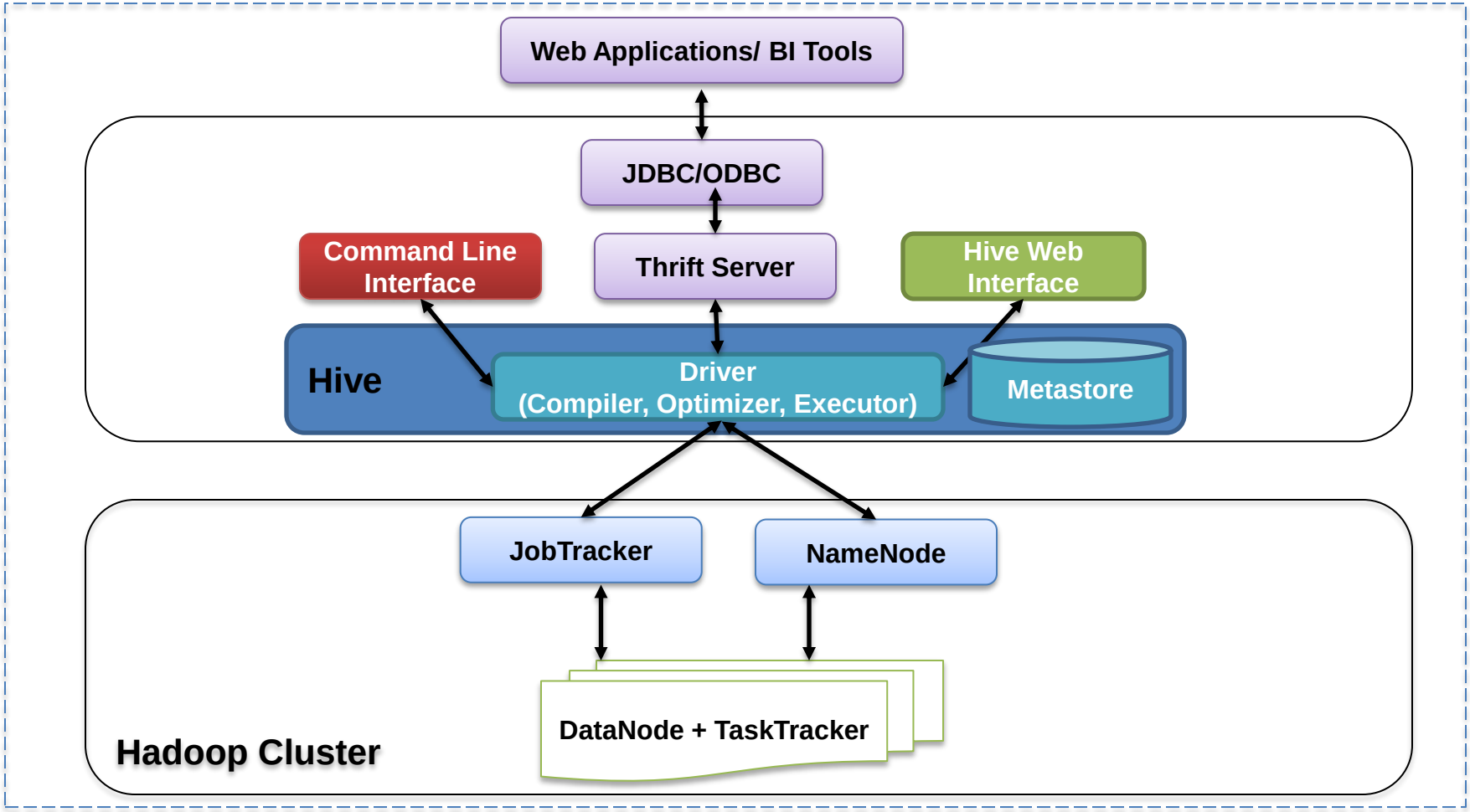
Contents

- History & Overview
- Architecture & Components
- Hive Query Language Features

History & Overview

- Apache Hive is a data warehouse software that facilitates reading, writing and managing large datasets on distributed storage systems using SQL
- It was built at Facebook by Jeff Hammerbacher & team as they needed a framework for data warehousing on top of Hadoop and was open-sourced in August, 2008
- Hive grew from a need to manage and process huge volumes of data that Facebook was producing every day from its fast-growing social network data
- It was developed with the intent of enabling analysts with strong SQL skills to run queries on the huge volumes of data that Facebook stored in HDFS
- Hive today is being used by many organizations as a general-purpose, scalable data processing platform
- SQL is not suitable for all big data problems, for instance it cannot be used for building complex machine-learning algorithms; but it is well-suited for many types of analysis and is widely known in the industry
- Hive is also quite suitable to integrate with various business intelligence products by means like ODBC bridge

Architecture & Components



Using Hive

- Hive shell – primary means of using Hive, by issuing commands in *HiveQL*

```
$ hive
hive>
$ hive -e 'SELECT * FROM dummy'
$ hive -f script.q
```

- Hive query example:

```
CREATE TABLE book(word STRING)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ' '
   LINES TERMINATED BY '\n';

LOAD DATA INPATH '/user/hduser/<sub-dir>/War_and_Peace.txt' INTO TABLE book;

SELECT LOWER(word), count(*) as count
   FROM book
   GROUP BY word
   SORT BY count DESC;
```

MapReduce Code in Java

Word Count


```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException
     {
      StringTokenizer itr = new
     StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

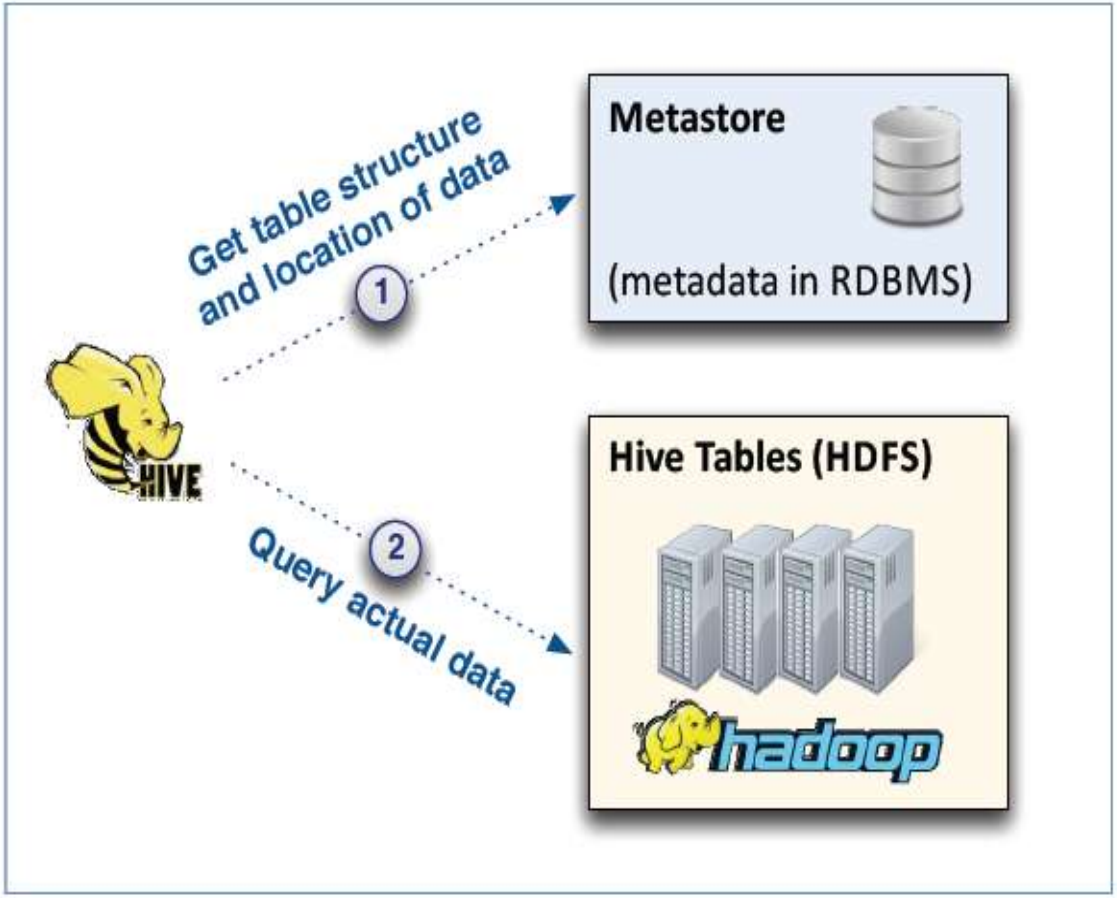
How Hive Loads & Stores Data?

- **Hive's queries operate on tables, just like in an RDBMS**
  - A table is simply an HDFS directory containing one or more files
  - Default path: `/user/hive/warehouse/<table_name>`
  - Hive supports many formats for data storage and retrieval
- **How does Hive know the structure and location of tables?**
  - These are specified when tables are created
  - This metadata is stored in Hive's *metastore*
    - Contained in an RDBMS such as MySQL

By default an embedded Derby database instance on local disk is used for metastore

How Hive Loads & Stores Data?

- **Hive consults the metastore to determine data format and location**
  - The query itself operates on data stored on a filesystem (typically HDFS)



Hive's Data Type

- **Each column in Hive is associated with a data type**
- **Hive supports more than a dozen types**
  - Most are similar to ones found in relational databases
  - Hive also supports three complex types
- **Use the `DESCRIBE` command to see a table's column types**

```
hive> DESCRIBE products;
prod_id       int
brand         string
name          string
price         int
cost          int
shipping_wt   int
```

Hive's Integer Type

- **Integer types are appropriate for whole numbers**
  - Both positive and negative values allowed

Name	Description	Example Value
TINYINT	Range: -128 to 127	17
SMALLINT	Range: -32,768 to 32,767	5842
INT	Range: -2,147,483,648 to 2,147,483,647	84127213
BIGINT	Range: ~ -9.2 quintillion to ~ 9.2 quintillion	632197432180964

Hive's Decimal Types

- **Decimal types are appropriate for floating point numbers**
  - Both positive and negative values allowed
  - **Caution**: avoid using when exact values are required!

Name	Description	Example Value
FLOAT	Decimals	3.14159
DOUBLE	Very precise decimals	3.14159265358979323846

Other Simple Types In Hive

- **Hive can also store several other types of information**
  - Only one character type (variable length)

Name	Description	Example Value
STRING	Character sequence	Betty F. Smith
BOOLEAN	True or False	TRUE
TIMESTAMP*	Instant in time	2013-06-14 16:51:05
BINARY*	Raw bytes	*N/A*

* Not available in older versions of Hive

Complex Column Types in HIVE

- **Hive also has a few complex data types**
  - These are capable of holding multiple values

Column Type	Usage in Query
ARRAY Ordered list of values, all of the same type	`departments[0]`
MAP Key-value pairs, each of the same type	`employees['BF5314']`
STRUCT Named fields, of possibly mixed types	`address.street`

Basic HiveQL Syntax

- **Hive keywords are not case-sensitive**
  - Though they are often capitalized by convention
- **Statements are terminated by a semicolon**
  - A statement may span multiple lines
- **Comments begin with --** ***(double hyphen)***
  - Only supported in Hive scripts
  - There are no multi-line comments in Hive

```
$ cat myscript.hql

SELECT cust_id, fname, lname
   FROM customers
   WHERE zipcode='60601';     -- downtown Chicago
```

Exploring Hive Databases & Tables

Exploring Hive Databases & Tables

- Hive concept of a Database is just a catalog or namespace.
- If Database not specified “default” database is used.
- Default Metastore database “Derby”
- Creating database
  - Creates directory for each database
  - Default database is “default”
  - Adding descriptive comments to the database.

Creating Database In Hive

- **Hive databases are simply namespaces**
  - Helps to organize your tables
- **To create a new database**

```
hive> CREATE DATABASE dualcore;
```

- **To conditionally create a new database**
  - Avoids error in case database already exists (useful for scripting)

```
hive> CREATE DATABASE IF NOT EXISTS dualcore;
```

Exploring Hive Databases

- **Which databases are available?**

```
hive> SHOW DATABASES;
accounting
default
sales
```

- **Switch between databases with the `USE` command**

```
$ hive
hive> SELECT * FROM customers;
hive> USE sales;
hive> SELECT * FROM customers;
```

Queries table in the `default` database

Queries table in the `sales` database

Creating Table In Hive

- **Basic syntax for creating a table:**

```
CREATE TABLE tablename (colname DATATYPE, ...)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY char
       STORED AS {TEXTFILE|SEQUENCEFILE|RCFILE}
```

- **Creates a subdirectory under `/user/hive/warehouse` in HDFS**
  - This is Hive's *warehouse directory*

Creating Table In Hive


```
CREATE TABLE tablename (colname DATATYPE, ...)
       ROW FORMAT DELIMITED
```

Specify a name for the table, and list the column names and datatypes (see later)

Creating Table In Hive


```
CREATE TABLE tablename (colname DATATYPE, ...)
       ROW FORMAT DELIMITED
```

This line states that fields in each file in the table's directory are delimited by some character. Hive's default delimiter is Control-A, but you may specify an alternate delimiter...

Creating Table In Hive


```
CREATE TABLE tablename (colname DATATYPE, ...)
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY char
```

…for example, tab-delimited data would require that you specify `FIELDS TERMINATED BY '\t'`

Creating Table In Hive


```
CREATE TABLE tablename (colname DATATYPE, ...)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY char
       STORED AS {TEXTFILE|SEQUENCEFILE|RCFILE}
```

Finally, you may declare the file format. `STORED AS TEXTFILE` is the default and does not need to be specified

Example Table Definition

- **The following example creates a new table named `jobs`**
  - Data stored as text with four comma-separated fields per line

```
CREATE TABLE jobs
    (id INT,
     title STRING,
     salary INT,
     posted TIMESTAMP
    )
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ',';
```

  - Example of corresponding record for the table above

```
1,Data Analyst,100000,2013-06-21 15:52:03
```

Exploring Hive Tables

- The `DESCRIBE` command displays basic structure for a table

```
hive> DESCRIBE orders;
order_id     int
cust_id      int
order_date   timestamp
```

- `DESCRIBE FORMATTED` shows more detailed information

```
hive> DESCRIBE FORMATTED orders;
# col_name          data_type          comment
order_id            int                None
cust_id             int                None
order_date          timestamp          None
# Detailed Table Information    ... More follows...
```

Exploring Hive Databases

- Which tables does the current database contain?

```
hive> USE accounting;
hive> SHOW TABLES;
invoices
taxes
```

- Which tables are contained in a different database?

```
hive> SHOW TABLES IN sales;
customers
prospects
```

Basic HiveQL Syntax

- **The `SELECT` statement retrieves data from Hive tables**
  - Can specify an ordered list of individual columns

```
hive> SELECT cust_id, fname, lname FROM customers;
```

  - An asterisk matches all columns in the table

```
hive> SELECT * FROM customers;
```

Basic HiveQL Syntax

Select is a PROJECTION OPERATOR.

```
hive> select  name, salary FROM employees;
```

Used for specifying regular expressions

```
hive> select * from stocks;
```

Limit Clause:

```
hive>Select name, salary from employees LIMIT 2;
```

Limiting & Sorting Query Results

- The `LIMIT` clause sets the maximum number of rows returned

```
hive> SELECT fname, lname FROM customers LIMIT 10;
```

- Caution: no guarantee regarding *which* 10 results are returned
  - Use `ORDER BY` for top-N queries
  - The field(s) you `ORDER BY` must be selected

```
hive> SELECT cust_id, fname, lname FROM customers
      ORDER BY cust_id DESC LIMIT 10;
```

Using WHERE clause To Restrict Results

- **WHERE clauses restrict rows to those matching specified criteria**
  - String comparisons are case-sensitive

```
hive> SELECT * FROM orders WHERE order_id=1287;
```

```
hive> SELECT * FROM customers WHERE state
      IN ('CA', 'OR', 'WA', 'NV', 'AZ');
```

- **You can combine expressions using AND or OR**

```
hive> SELECT * FROM customers
      WHERE fname LIKE 'Ann%'
      AND (city='Seattle' OR city='Portland');
```

Table Aliases

- **Table aliases can help simplify complex queries**

```
hive> SELECT o.order_date, c.fname, c.lname
      FROM customers c JOIN orders o
      ON c.cust_id = o.cust_id
      WHERE c.zipcode='94306';
```

  - Note: Using `AS` to specify table aliases is *not* supported

Exploring Hive Tables

- Types of tables:
  1. Managed tables (Internal Tables)
  2. External tables.

     creating table

     dropping table

     Adding data into the table.

Record Grouping & Aggregated Function

- GROUP BY groups selected data by one or more columns
  - **Caution**: Columns not part of aggregation must be listed in GROUP BY

stores table

id	city	state	region
a	Albany	NY	EAST
b	Boston	MA	EAST
c	Chicago	IL	NORTH
d	Detroit	MI	NORTH
e	Elgin	IL	NORTH

```
SELECT region, state,
       COUNT(id) AS num
  FROM stores
  GROUP BY region, state;
```

Result of query

region	state	num
EAST	MA	1
EAST	NY	1
NORTH	IL	2
NORTH	MI	1

Built In Aggregated Function

- **Hive offers many aggregate functions, including**

Function Description	Example Invocation
Count all rows	`COUNT(*)`
Count all rows where field is not null	`COUNT(fname)`
Count all rows where field is unique and not null	`COUNT(DISTINCT fname)`
Returns the largest value	`MAX(salary)`
Returns the smallest value	`MIN(salary)`
Adds all supplied values and returns result	`SUM(price)`
Returns the average of all supplied values	`AVG(salary)`

JOINING DATA SETS

Joins In HIVE

- **Joining disparate data sets is a common operation in Hive**
- **Hive supports several types of joins**
  - Inner joins
  - Outer joins (left, right, and full)
  - Cross joins (supported in Hive 0.10 and later)
  - Left semi joins
- **Only equality conditions are allowed in joins**
  - Valid: `customers.cust_id = orders.cust_id`
  - Invalid: `customers.cust_id <> orders.cust_id`
  - Outputs records where the specified key is found in each table
- **For best performance, list the largest table last in your query**

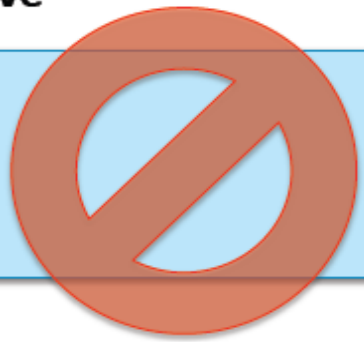
Join Syntax

- **Hive requires the following syntax for joins**

```
SELECT c.cust_id, name, total
FROM customers c
JOIN orders o ON (c.cust_id = o.cust_id);
```

- **The above example is an inner join**
  - Can replace `JOIN` with another type (e.g. `RIGHT OUTER JOIN`)
- **The following join syntax is not valid in Hive**

```
SELECT c.cust_id, name, total
FROM customers c, orders o
WHERE (c.cust_id = o.cust_id);
```



Common Built–in Functions

Hive Functions

- **Hive offers dozens of built-in functions**
  - Many are identical to those found in SQL
  - Others are Hive-specific
- **Example function invocation**
  - Function names are not case-sensitive

```
hive> SELECT CONCAT(fname, ' ', lname) AS fullname
      FROM customers;
```

- **To see information about a function**

```
hive> DESCRIBE FUNCTION UPPER;
      UPPER(str) - Returns str with all characters
      changed to uppercase
```

Example Built-In Function

- These functions operate on numeric values

Function Description	Example Invocation	Input	Output
Rounds to specified # of decimals	`ROUND(total_price, 2)`	23.492	23.49
Returns nearest integer above	`CEIL(total_price)`	23.492	24
Returns nearest integer below	`FLOOR(total_price)`	23.492	23
Return absolute value	`ABS(temperature)`	-49	49
Returns square root	`SQRT(area)`	64	8
Returns a random number	`RAND()`		0.584977

Example Built-In Function

- These functions operate on timestamp values

Function Description	Example Invocation	Input	Output
Convert to UNIX format	`UNIX_TIMESTAMP(order_dt)`	2013-06-14 16:51:05	1371243065
Convert to string format	`FROM_UNIXTIME(mod_time)`	1371243065	2013-06-14 16:51:05
Extract date portion	`TO_DATE(order_dt)`	2013-06-14 16:51:05	2013-06-14
Extract year portion	`YEAR(order_dt)`	2013-06-14 16:51:05	2013
Returns # of days between dates	`DATEDIFF(order_dt, ship_dt)`	2013-06-14, 2013-06-17	3

Example Built-In Function

- Here are some other interesting functions

Function Description	Example Invocation	Input	Output
Converts to uppercase	`UPPER(fname)`	Bob	BOB
Extract portion of string	`SUBSTRING(name, 0, 2)`	Alice	Al
Selectively return value	`IF(price > 1000, 'A', 'B')`	1500	A
Convert to another type	`CAST(weight as INT)`	3.581	3
Returns size of array or map	`SIZE(array_field)`	N/A	6

Partitioning in Hive

- Notion of Partitioning Data is an Old Concept, And Hive has the notion of Partitioned Tables.
- Partitioning of Tables changes How HIVE Structures the Data Storage.
- Benefits:
  - Performance Benefits.
  - Organize data in Logical fashion.
- **Partition Filters:** Predicates to the WHERE clause that filter on partition values are called as Partition Filters.

Data Storage in Hive

- **The metastore maintains information about Hive tables**
  - A table simply points to a directory in HDFS
  - The table's data are files within that directory
- **All files in the directory are read during a query**

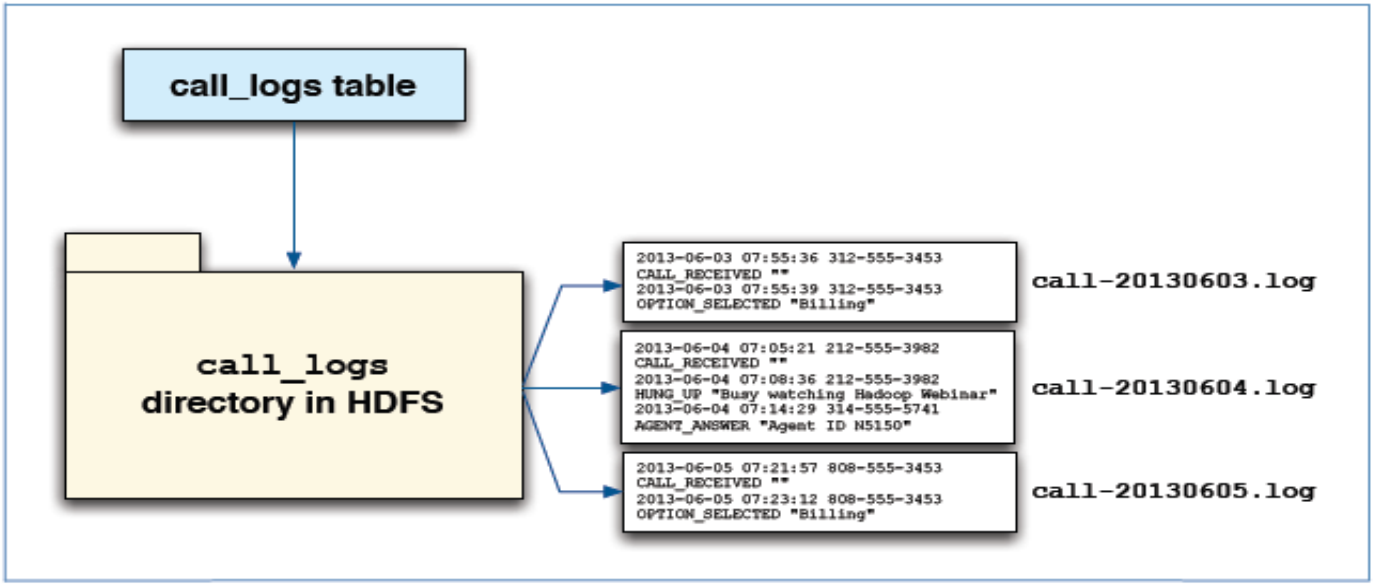
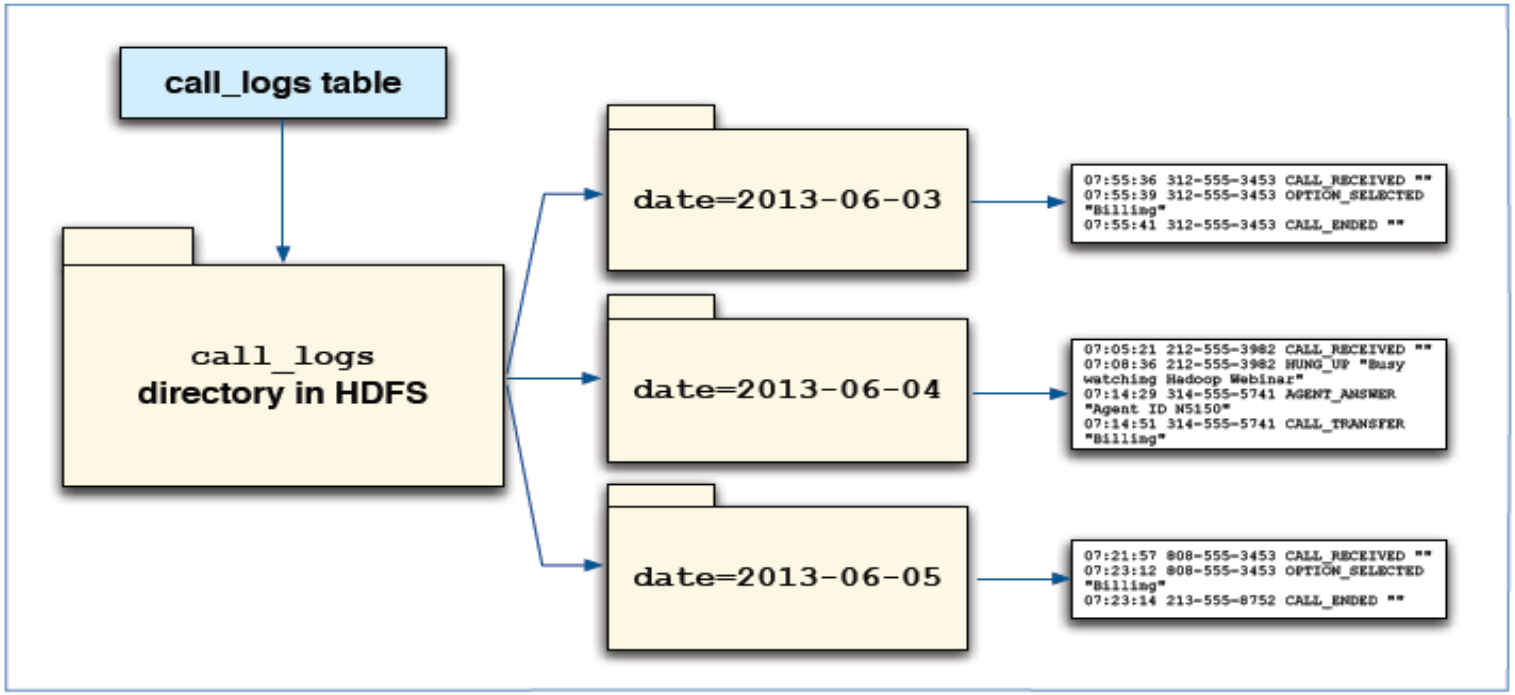


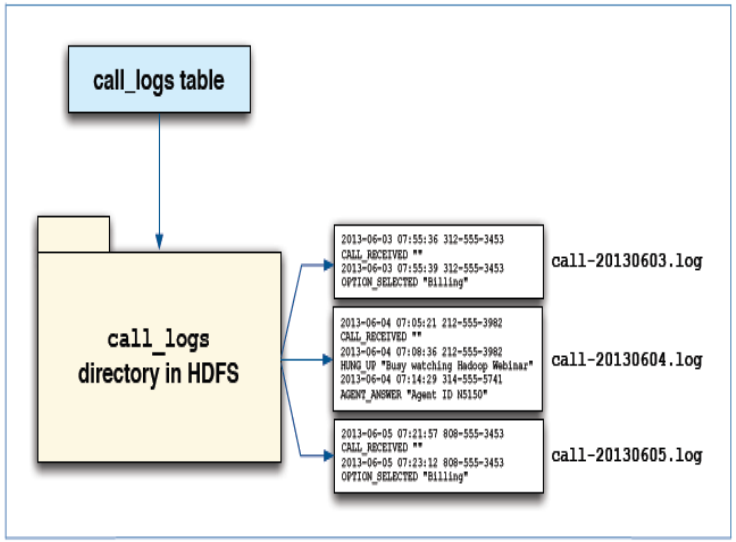
Table Partitioning

- **It is possible to create a table that partitions the data**
  - Queries that filter on partitioned fields limit amount of data read
  - Does not prevent you from running queries that span partitions

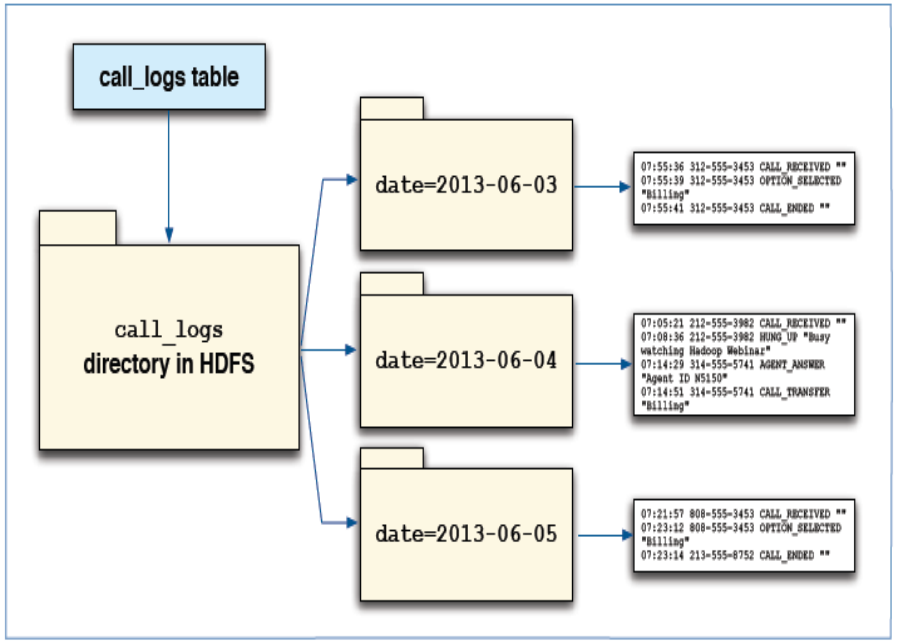


Comparison

- The metastore maintains information about Hive tables
  - A table simply points to a directory in HDFS
  - The table's data are files within that directory
- All files in the directory are read during a query



- It is possible to create a table that partitions the data
  - Queries that filter on partitioned fields limit amount of data read
  - Does not prevent you from running queries that span partitions



Creating A Partitioned Table In Hive

- Specify the partitioned column in the `PARTITION` clause

```
CREATE TABLE call_logs (
   call_time STRING,
   event_type STRING,
   phone STRING,
   details STRING)
  PARTITIONED BY (call_date STRING)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';
```

- You can also created nested partitions

```
PARTITIONED BY (call_date STRING, event_type STRING)
```

Creating A Partitioned Table In Hive

- **The partition column is displayed if you `DESCRIBE` the table:**

```
hive> DESCRIBE call_logs;
OK
call_time     string
event_type    string
phone  string
details       string
call_date     string
```

- **However, the partition is a "virtual column"**
  - The data does not exist in your incoming data
  - Instead, you specify the partition when loading the data

Loading Data Into Partition

- You must specify partition field value when loading data

```
hive> LOAD DATA INPATH 'call-20130603.log'
        INTO TABLE call_logs
        PARTITION(call_date='2013-06-03');

hive> LOAD DATA INPATH 'call-20130604.log'
        INTO TABLE call_logs
        PARTITION(call_date='2013-06-04');
```

- The above example would create two subdirectories:
  - /user/hive/warehouse/call_logs/call_date=2013-06-03
  - /user/hive/warehouse/call_logs/call_date=2013-06-04

Viewing, Adding or Altering Partition

- To view the current partitions in a table

```
hive> SHOW PARTITIONS call_logs;
```

- Use ALTER TABLE to add or drop partitions

```
ALTER TABLE call_logs
   ADD PARTITION (call_date='2013-06-05')
   LOCATION '/dualcore/call_logs/call_date=2013-06-05';
```

```
ALTER TABLE call_logs
   DROP PARTITION (call_date='2013-06-06');
```

Comparing Hive To Relational Database

	Relational Database	Hive
Query language	SQL	HiveQL
Update individual records	Yes	No
Delete individual records	Yes	No
Transactions	Yes	No
Index support	Extensive	Limited
Latency	Very low	High
Data size	Terabytes	Petabytes

Schema on Read* Versus *Schema on Write

- In a traditional database, 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. This design is sometimes called *schema on write because the data is checked against the schema when it is written into the* database.
- Hive, on the other hand, doesn't verify the data when it is loaded, but rather when a query is issued. This is called *schema on read.*

Bucketed Tables in Hive

- Bucketing in Hive is another way of giving more fine grained structure to Hive tables.
- Hive clusters the records together into a given number of buckets based on a specified column.
- Records with the same bucket column wll be saved in the same bucket.
- Physically each bucket is a file in a table directory or a partition directory.
- Bucketing can be done along wiht partitioning of the table or without partitioning.
- Unlike partitioned columns (which are not included in table columns definition) , Bucketed columns are included in table definition.
- ```
  create table customers_b (customer_id int, customer_name string, customer_city string, customer_zipcode string) PARTITIONED BY (customer_state string) CLUSTERED BY (customer_city) INTO 50 BUCKETS row format delimited fields terminated by "\t" lines terminated by "\n";
  ```
- Bucketing is used when the number of partitioning can reult in a very large number of parititions. In other words large number of paritions can make querying slow hence bucketing is used.
  ```