Hive

Querying and managing large datasets

Today’s lecture

- Basics
- Loading data into Hive
- Hive queries
- Text processing

Introduction to Hive

- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive
- Hive vs. Pig

What is Hive?

- Data warehousing framework
  - Designed to warehouse **petabytes**
  - Facilitates **queries** against data stored in Hadoop
- Developed by the Data Infrastructure Team at Facebook
- Facebook engineering note:

  http://www.facebook.com/note.php?note_id=89508453919

General interest: https://www.facebook.com/Engineering/notes

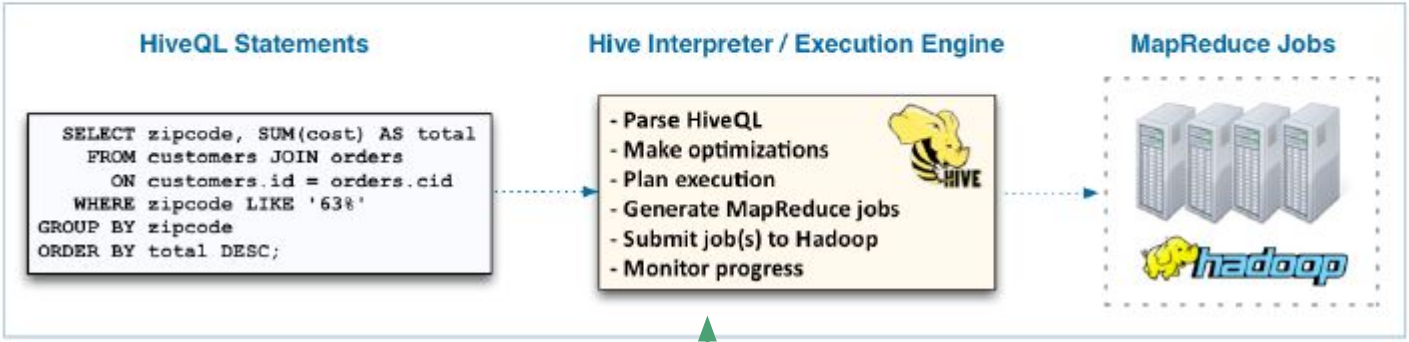
Overview of Apache Hive

- **Apache Hive is a high-level abstraction on top of MapReduce**
  - Uses a SQL-like language called HiveQL
  - Generates MapReduce jobs that run on the Hadoop cluster
  - Originally developed by Facebook for data warehousing
    - Now an open-source Apache project


```
  SELECT zipcode, SUM(cost) AS total
    FROM customers
    JOIN orders
      ON customers.cust_id = orders.cust_id
   WHERE zipcode LIKE '63%'
GROUP BY zipcode
ORDER BY total DESC;
```

Overview for Hive Job Submission

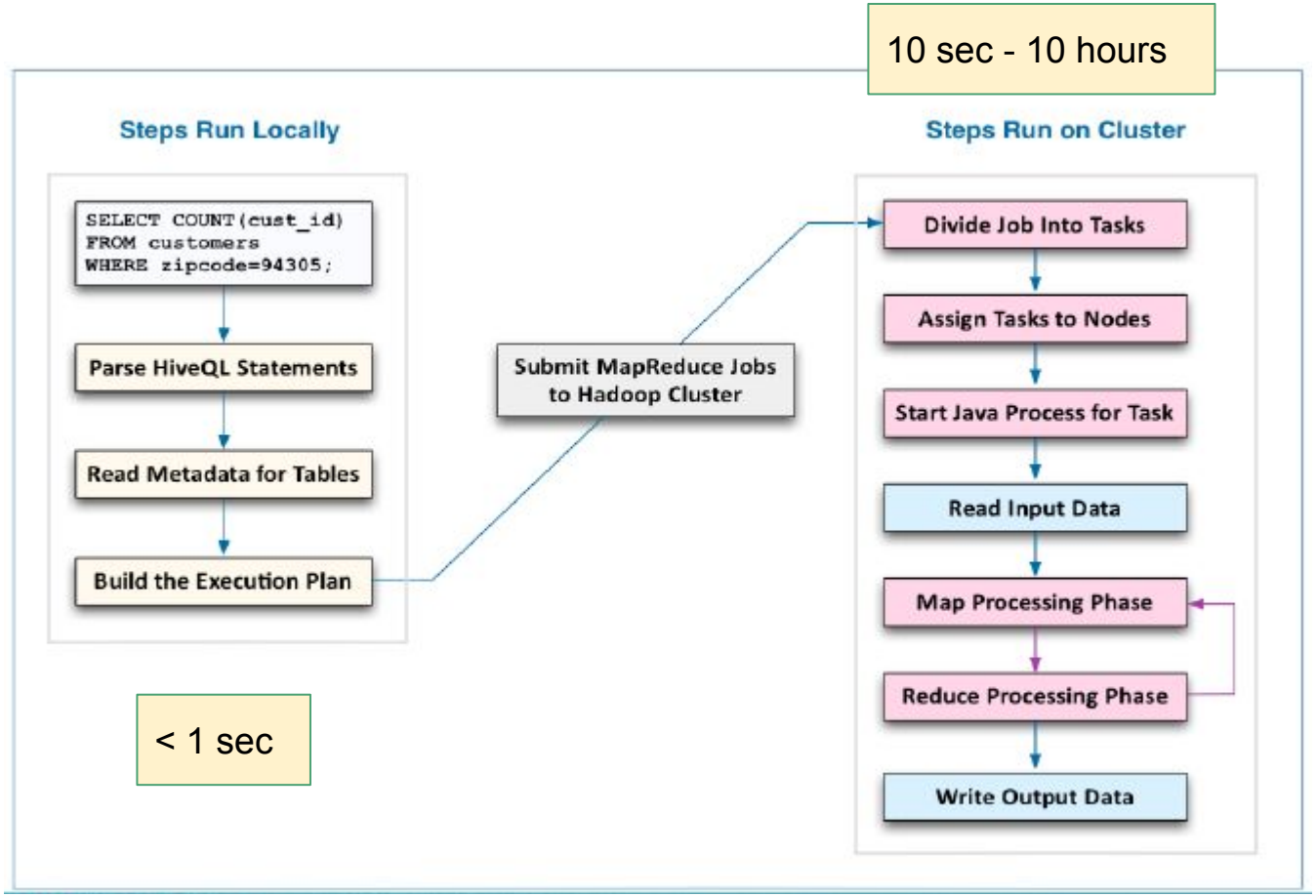
- **Hive runs on the client machine**
  - Turns HiveQL queries into MapReduce jobs
  - Submits those jobs to the cluster

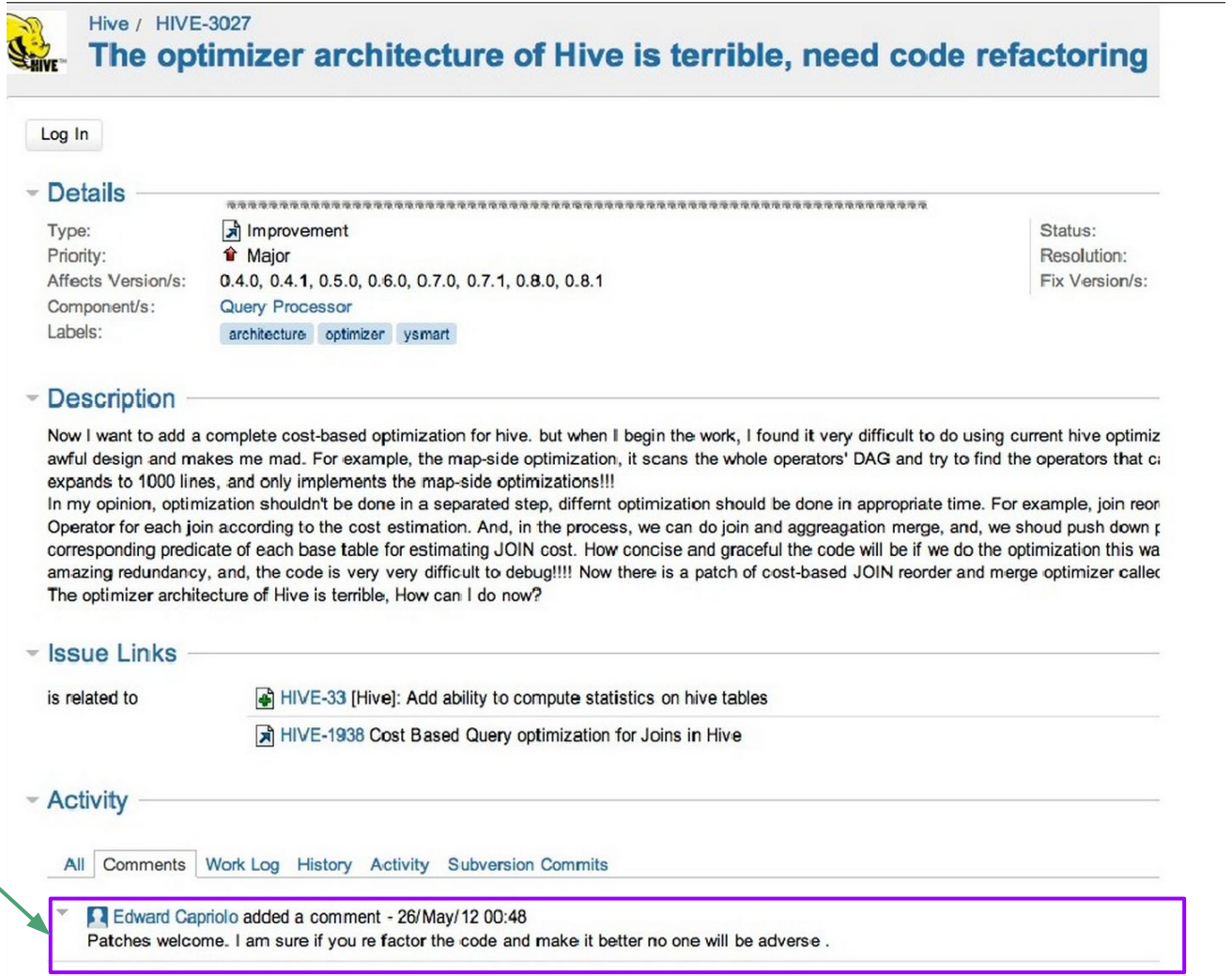


more on Hive execution architecture

Plan building and execution:
http://www.slideshare.net/nzhang/hive-anatomy
http://www.slideshare.net/recruitcojp/internal-hive

How Hive Processes Data





contributors, please start here...

Why Use Apache Hive?

- **More productive than writing MapReduce directly**
  - Five lines of HiveQL might be equivalent to 100 lines or more of Java
- **Brings large-scale data analysis to a broader audience**
  - No software development experience required
  - Leverage existing knowledge of SQL
- **Offers interoperability with other systems**
  - Extensible through Java and external scripts
  - Many business intelligence (BI) tools support Hive

Introduction to Hive

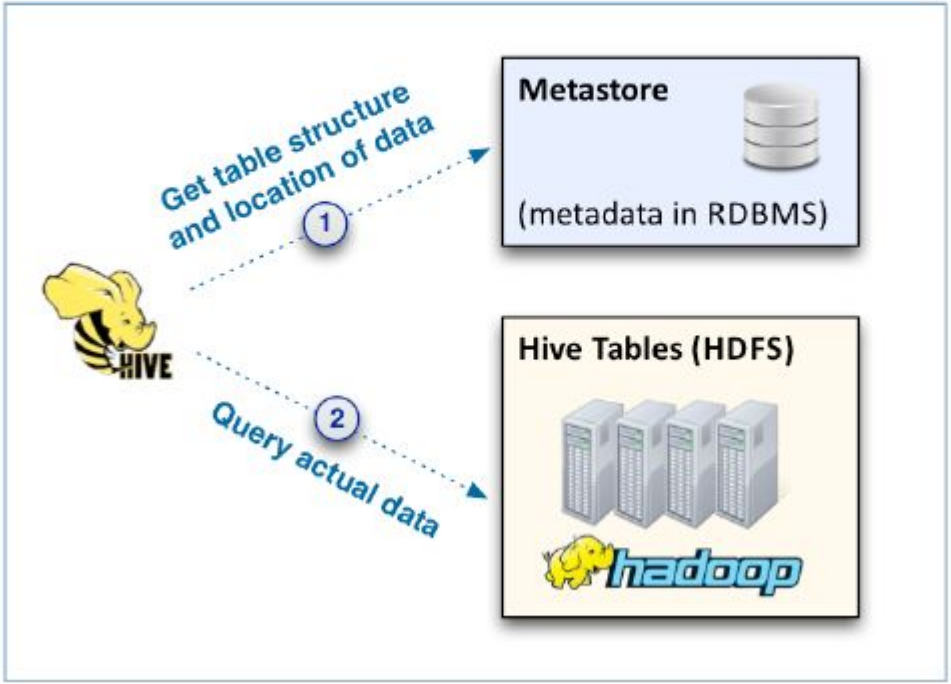
- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive
- Hive vs. Pig

How Hive Loads and Stores Data (1)

- **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

How Hive Loads and Stores Data (2)

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



Introduction to Hive

- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive

The Cluster is not a Database Server

- **Client-server database management systems have many strengths**
  - Very fast response time
  - Support for transactions
  - Allows modification of existing records
  - Can serve thousands of simultaneous clients
- **Hive does not turn your Hadoop cluster into an RDBMS**
  - It simply produces MapReduce jobs from HiveQL queries
  - Limitations of HDFS and MapReduce still apply

Comparing Hive to a Relational Database

	Relational Database	Hive
Query language	SQL	HiveQL
Update/delete records	Yes	Yes*
Transactions	Yes	Yes*
Index support	Extensive	Limited
Latency	Very low	High
Data size	Terabytes	Petabytes

***As of Hive 0.14 (Nov, 2014)**
Note: if a table is to be used for insert, update, delete then the table property "transactional" must be set. Without this value, inserts will be done in the old style; updates and deletes will be prohibited.

Introduction to Hive

- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive
- Hive vs. Pig

Use Case: Log File Analytics

- **Server log files are an important source of data**
- **Hive allows you to treat a directory of log files like a table**
  - Allows SQL-like queries against raw data

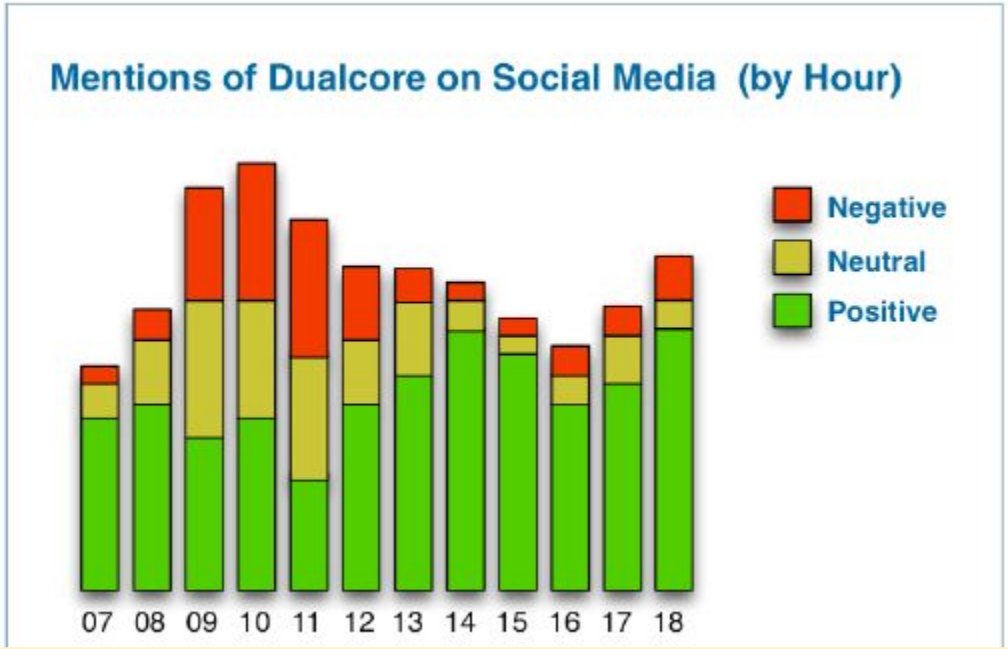
Dualcore Inc. Public Web Site (June 1 - 8)

Product	Unique Visitors	Page Views	Average Time on Page	Bounce Rate	Conversion Rate
Tablet	5,278	5,894	17 seconds	23%	65%
Notebook	4,139	4,375	23 seconds	47%	31%
Stereo	2,873	2,981	42 seconds	61%	12%
Monitor	1,749	1,862	26 seconds	74%	19%
Router	987	1,139	37 seconds	56%	17%
Server	314	504	53 seconds	48%	28%
Printer	86	97	34 seconds	27%	64%

- **Real-world Orbitz use case for processing log files:**
  - See: www.slideshare.net/jseidman/data-analysis-with-hadoop-and-hive-chicagodb-2212011
  - Start at slide 27, actual code on slide 34-36.

Use Case: Sentiment Analysis

- Many organizations use Hive to analyze social media coverage



see: http://blog.cloudera.com/blog/2013/03/how-to-analyze-twitter-data-with-hue/

Introduction to Hive

- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive
- Hive vs. Pig

Using the Hive Shell

- **You can execute HiveQL statements in the Hive Shell**
  - This interactive tool is similar to the MySQL shell
- **Run the `hive` command to start the Hive shell**
  - The Hive shell will display its `hive>` prompt
  - Each statement must be terminated with a semicolon
  - Use the `quit` command to exit the Hive shell

```
$ hive
hive> SELECT cust_id, fname, lname
      FROM customers WHERE zipcode=20525;
1000000    Quentin    Shepard
1000001    Brandon    Louis
1000002    Marilyn    Ham
hive> quit;
$
```

Accessing Hive from the Command Line

- You can also execute a file containing HiveQL code using the `-f` option

```
$ hive -f myquery.hql
```

- Or use HiveQL directly from the command line using the `-e` option

```
$ hive -e 'SELECT * FROM users'
```

- Use the `-S` (silent) option to suppress informational messages
  - Can also be used with `-e` or `-f` options

```
$ hive -S
```

More on configuration: Programming Hive, pp 24-34

Hive Properties

- **Many aspects of Hive's behavior are configured through properties**
  - Use `set -v` in Hive to see current values

```
hive> set -v;
```

- **You can also use set to specify property values**
  - The following enables columns headers in query results

```
hive> set hive.cli.print.header=true;
```

- **Hive runs the `.hiverc` file in your home directory at startup**
  - Useful for specifying per-user defaults

Hive properties: https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

Interacting with the Operating System and HDFS

- **Use ! to execute system commands from within Hive**
  - Neither pipes nor globs (wildcards) are supported

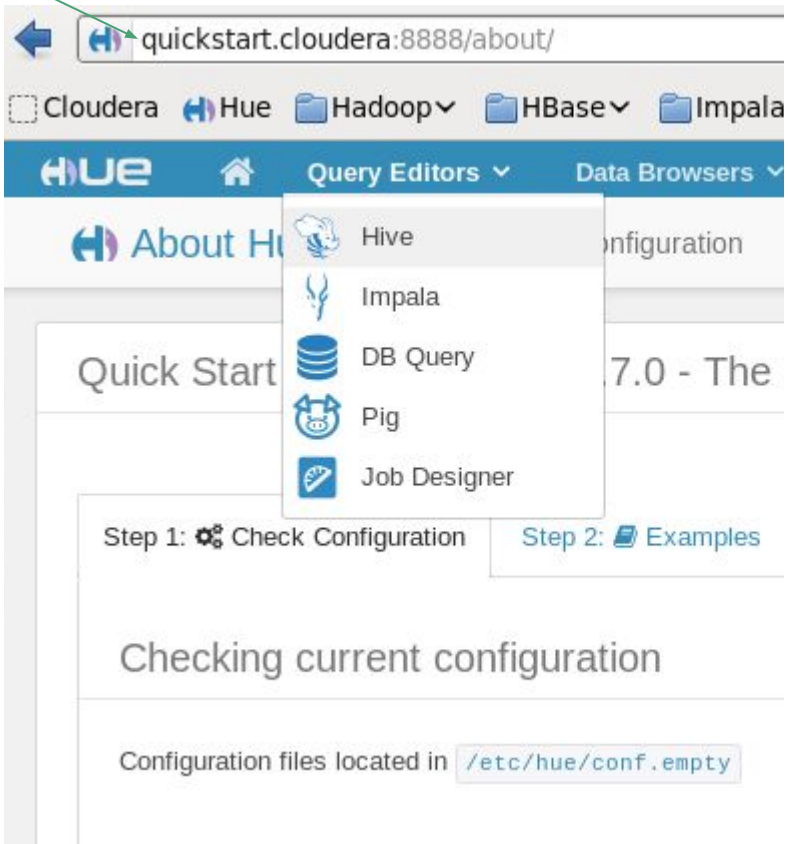
```
hive> ! date;
Mon May 20 16:44:35 PDT 2013
```

- **Prefix HDFS commands with `dfs` to use them from within Hive**

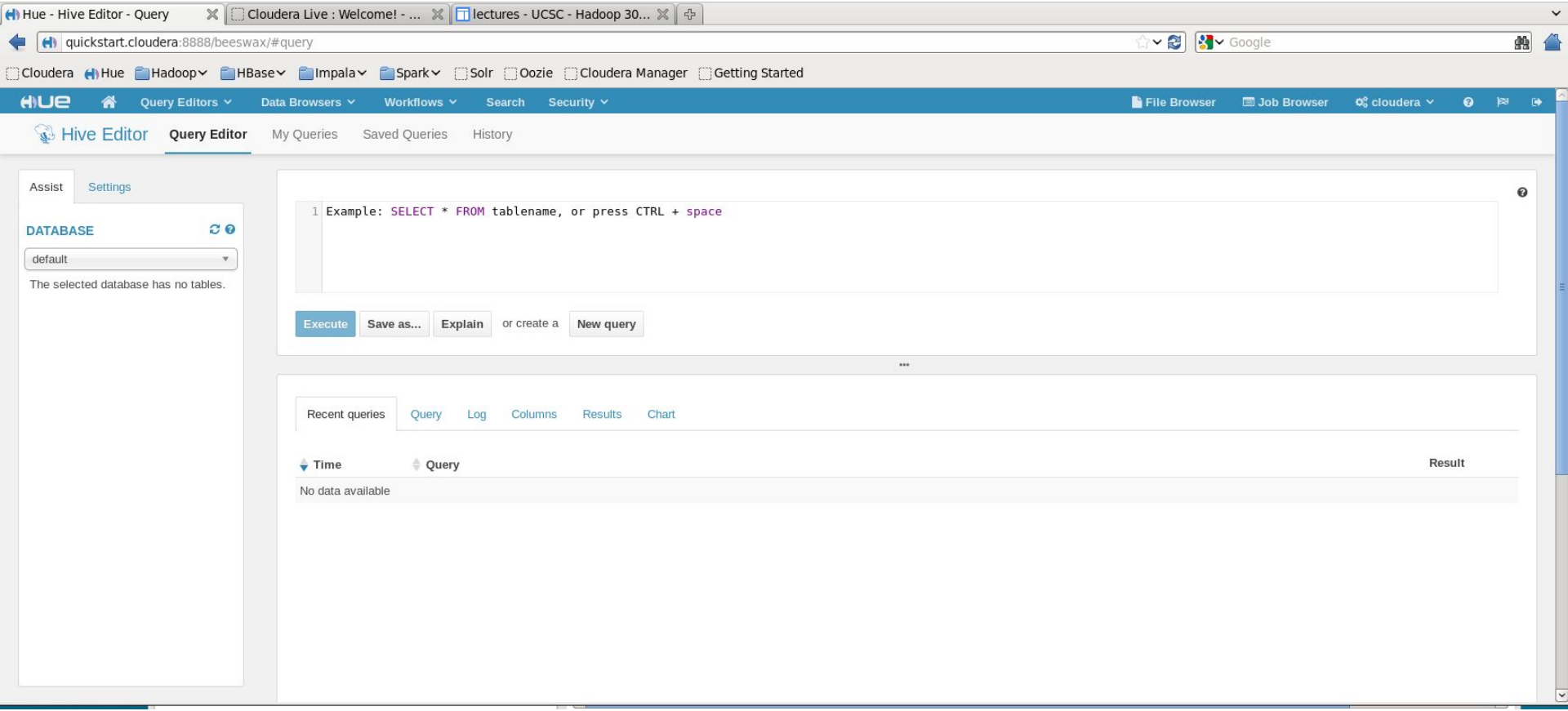
```
hive> dfs -mkdir /reports/sales/2013;
```

Accessing Hive using Hue

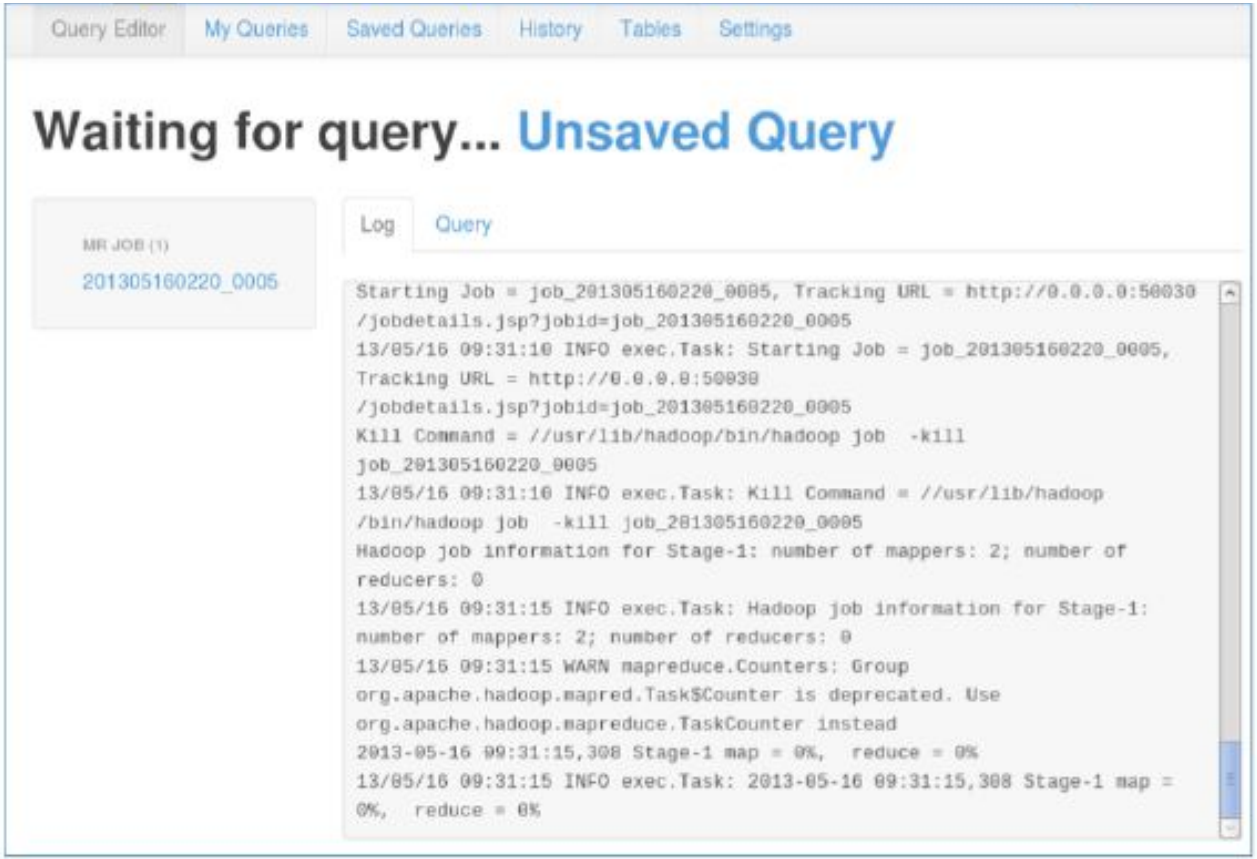
- May need to start Hue service
  - `sudo service hue start`
- Hive's UI on Hue is called Beeswax
- Launch by clicking on the icon
- What you can do in Hive on Hue:
  - Create tables
  - Run queries
  - Browse tables
  - Save queries



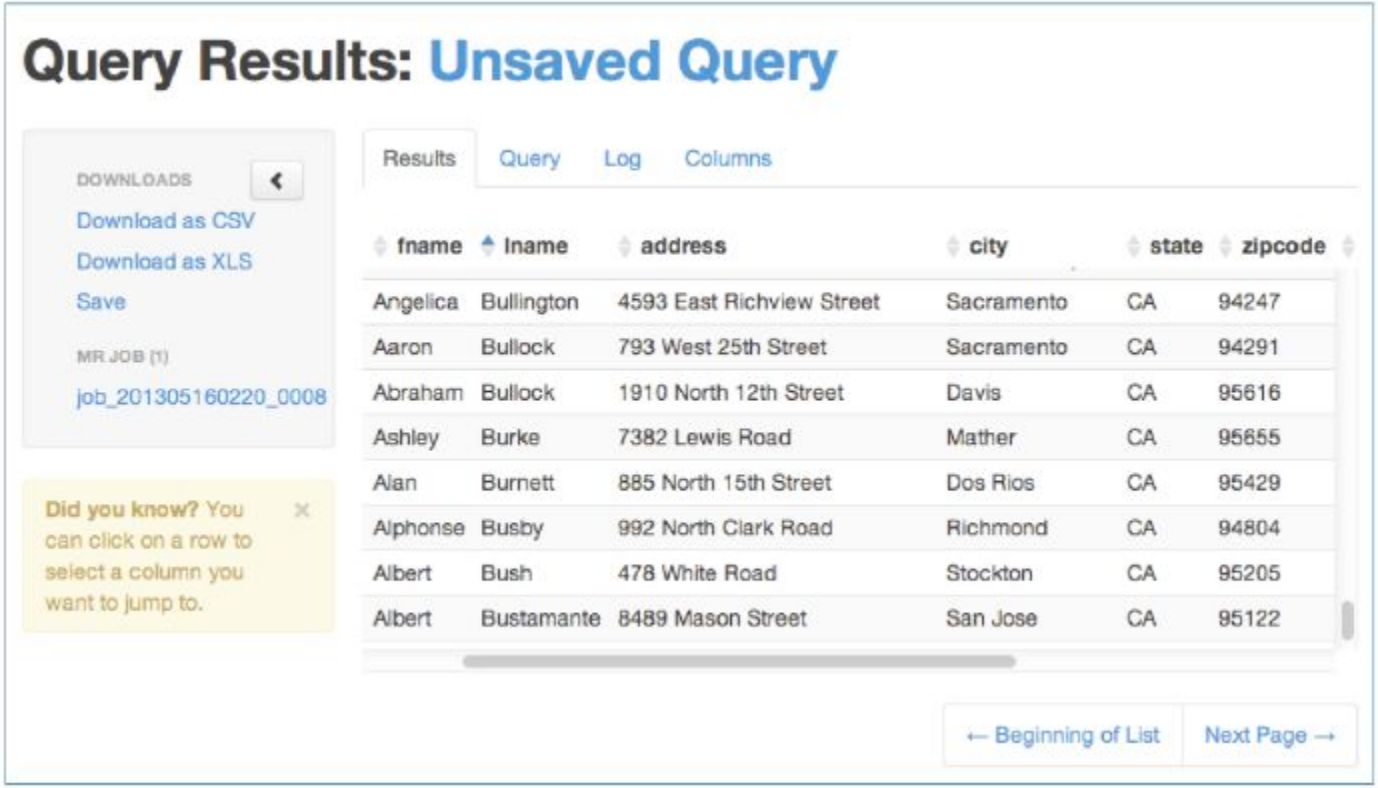
Hive query in Hue



Query Execution in Hue

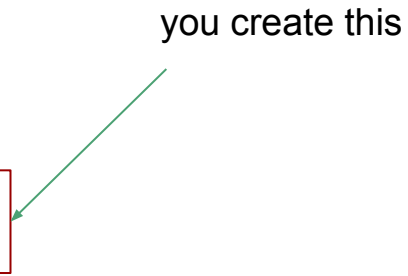


Query Results in Hue



Locations for Hive in your VM

- hive startup script: /usr/bin/hive
- HIVE_HOME: /usr/lib/hive
- hive configuration files: /etc/hive/conf
- log files: /tmp/cloudera
- query history: /home/cloudera /.hivehistory
- user-defined settings: /home/training/.hiverc



Introduction to Hive

- What is Hive?
- Hive Schema and Data Storage
- Comparing Hive to Traditional Databases
- Hive Use Cases
- Interacting with Hive
- Hive vs. Pig

Next week, we will spend some time on Pig

Unlike Hive's SQL-like interface, Pig uses a procedural programming language

Like Hive, Pig's programming language (grunt) compiles down to MapReduce code

Pig is used in processing pipelines or dataflows

- Allows one to specify and/or (re)write operations like joins and splits
- Supports data with partial or unknown schemas, and semi-structured or unstructured data
- Provides the ability to optimize operations "set in stone" in Hive

Hive vs Pig

	Pig	**Hive**
Usage	Processing pipelines Research on semi-structured data	Queries OLAP (on-line analysis)
Language	PigLatin - procedural language	HiveQL - SQL-based language
Processing	Interpreter ● runs on a client machine ● generates code for jobs Execution engine submits jobs to a cluster	
Jobs	Traditional MapReduce Spark interface is maturing	Traditional MapReduce ***or*** **Spark**
Data management	Tables are defined when a file is loaded	● Tables are warehoused as files ● Table metadata (schemas) are stored in the Metastore

Essential Points

- **Hive is a high-level abstraction on top of MapReduce**
  - Runs MapReduce jobs on Hadoop based on HiveQL statements
  - Originally developed for data warehousing at Facebook
- **HiveQL is very similar to SQL**
  - Easy to learn for those with relational database experience
  - However, Hive does *not* replace your RDBMS
- **Hive tables are really directories of files in HDFS**
  - Information about those tables is kept in Hive's metastore
- **Hive and Pig are similar in many ways**
  - But each also has its own distinct advantages

Bibliography

- Hive Web Site
  - hive.apache.org
- Programming Hive (book by O'Reilly)
  - shop.oreilly.com/product/0636920023555.do
- Original work from Facebook
  - Synopsis: www.facebook.com/note.php?note_id=89508453919
  - Actual paper: http://infolab.stanford.edu/~ragho/hive-icde2010.pdf

Bibliography

- Deeper into Hive internals
  - www.slideshare.net/recruitcojp/internal-hive
  - www.slideshare.net/nzhang/hive-anatomy (by Facebook engineer)
- Beeline CLI references
  - sqlline.sourceforge.net
  - cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients
- A couple of interesting use cases
  - www.slideshare.net/jseidman/data-analysis-with-hadoop-and-hive-chicagodb-2212011
    - Start at slide 27, actual code on slide 34-36.
  - blog.cloudera.com/blog/2013/03/how-to-analyze-twitter-data-with-hue/

Hive Data Management

Loading data and creating tables

Hive Data Management

- Basics
- How Hive encodes and stores data
- Creating Hive databases and tables
- How to load data into tables
- How to alter and remove tables
- Using SerDe to create and load tables

Basics

Exploring Hive Databases and Tables (1)

- **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

Exploring Hive Databases and Tables (2)

- **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
```

Exploring Hive Databases and Tables (3)

- 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...
```

Hive Databases

- **Each Hive table belongs to a specific database**
- **Early versions of Hive supported only a single database**
  - It placed all tables in the same database (named `default`)
  - This is still the default behavior
- **Hive supports multiple databases as of release 0.7.0**
  - Helpful for organization and authorization

Creating Databases 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;
```

Removing a Database

- **Removing a database is similar to creating it**
  - Just replace **CREATE** with **DROP**

```
hive> DROP DATABASE dualcore;
```

```
hive> DROP DATABASE IF EXISTS dualcore;
```

- **These commands will fail if the database contains tables**
  - Add the **CASCADE** keyword to force removal
  - Caution: this command removes data in HDFS!

```
hive> DROP DATABASE dualcore CASCADE;
```

Creating Tables

- Data Types
- File Formats
- Create statements

Hive’s Data Types

- **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

Hive's Integer Types

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

	Name	Description	Example Value
byte →	TINYINT	Range: -128 to 127	17
short →	SMALLINT	Range: -32,768 to 32,767	5842
int →	INT	Range: -2,147,483,648 to 2,147,483,647	84127213
long →	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

Name	Description	Example Value
FLOAT	Decimals	3.14159
DOUBLE	Very precise decimals	3.141592653589793238 46

DECIMAL(n,m) - Decimals where n is the number of significant digits (the precision) and m is the number of digits behind the decimal point (the scale).

Caution: avoid using when exact values are required!

- see seminal paper on floating point errors: goldberg.pdf

Other Simple Types in Hive

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

Recently added:
- VARCHAR (maximum length)
- CHAR (fixed length)

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`

Creating Tables

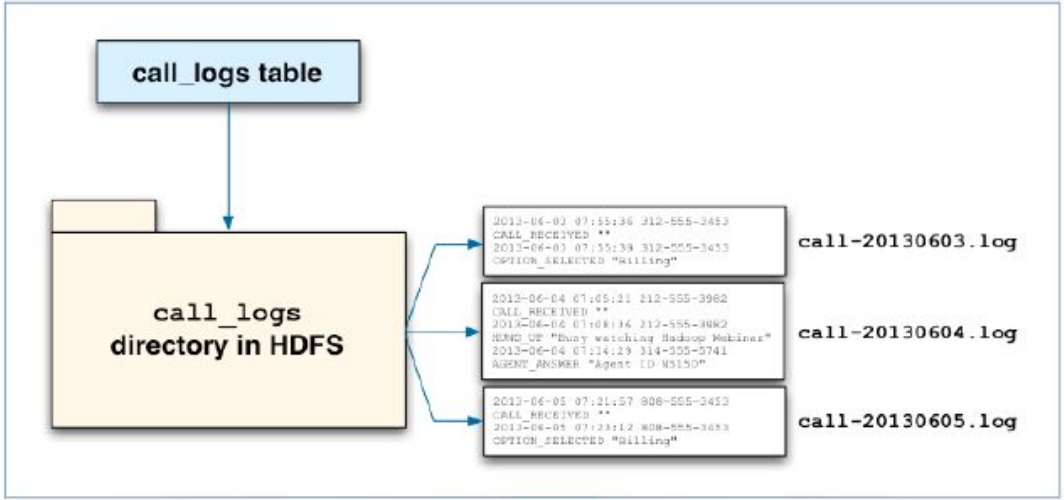
- Data Types
- File Formats
- Create statements

Data storage in Hive

- **Data for Hive's tables is stored on the filesystem (typically HDFS)**
  - Each table maps to a single directory
- **A table's directory may contain multiple files**
  - Typically delimited text files, but Hive supports many formats
  - Subdirectories are not allowed
- **Hive uses the metastore to give context to this data**
  - Helps map raw data in HDFS to named columns of specific types

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**



Hive Text File Format

- **Each Hive table maps to a directory of data, typically in HDFS**
  - Data stored in one or more files
- **Default data format is plain text**
  - One record per line (record separator is `\n`)
  - Columns are delimited by `^A` (field separator is Control-A)
  - Complex type elements are separated by `^B` (Control-B)
  - Map keys/values are separated by `^C` (Control-C)
- **Hive allows you to override these delimiters**
  - Specify alternate values during table creation

Hive binary file formats - splittable and compressible

- Avro - support block compression, evolvable schema
  - Schema for companylistNASDAQ
  - Code for writing an Avro file
- Parquet - columnar, evolvable schema
- SequenceFile - support block and record compression, self-aware
- ORC - columnar, similar to RCFile except better compression, faster queries
- RCFile - columnar, optimized for queries, slower for writes

Creating Tables

- Data Types
- File Formats
- Create statements

Creating a Table in Hive (1)

- **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*

Can store a table as:

- **TEXTFILE**
- **SEQUENCEFILE**
- **ORC**
- **PARQUET**
- **AVRO**
- **RCFILE**

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
```

Creating Tables with Complex Column Types

- **Complex columns are typed**
  - Arrays have a single type
  - Maps have a key type and a value type
  - Structs have a type for each attribute
- **These types are specified with angle brackets**

```
CREATE TABLE stores
   (store_id SMALLINT,
    departments ARRAY<STRING>,
    staff MAP<STRING, STRING>,
    address STRUCT<street:STRING,
                   city:STRING,
                   state:STRING,
                   zipcode:STRING>
);
```

Customizing Complex Type Storage

- We can control which delimiters are used for complex types

```
CREATE TABLE stores
   (store_id SMALLINT,
    departments ARRAY<STRING>,
    staff MAP<STRING, STRING>,
    address STRUCT<street:STRING,
                   city:STRING,
                   state:STRING,
                   zipcode:STRING>
   )
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ','
   MAP KEYS TERMINATED BY ':';
```

Row Format Example for Complex Types

- **The following is an example of a record in that table**

```
1|Audio,Photo|A123:Abe,B456:Bob|123 Oak St.,Ely,MN,55731
```

- **Examples queries on this record**

```
hive> SELECT departments[0] FROM stores;
Audio

hive> SELECT staff['B456'] FROM stores;
Bob

hive> SELECT address.city FROM stores;
Ely
```

Next week: Partitioning tables

Loading data

Data Validation in Hive

- **Hadoop and its ecosystem are 'schema on read'**
  - Unlike an RDBMS, Hive does not validate data on insert
    - Files are simply moved into place
  - Loading data into tables is therefore very fast
  - Errors in file format will be discovered when queries are performed
- **Missing or invalid data in Hive will be represented as `NULL`**

Loading Data From Files (1)

- **To load data, simply add files to the table's directory in HDFS**
  - Can be done directly using `hadoop fs` commands
  - This example loads data from HDFS into Hive's `sales` table

```
$ hadoop fs -mv sales.txt /user/hive/warehouse/sales/
```

- **Alternatively, use Hive's `LOAD DATA INPATH` command**
  - Done from within the Hive shell (or a Hive script)
  - This *moves* data within HDFS, just like the command above
  - Source can be either a file or directory

```
hive> LOAD DATA INPATH 'sales.txt' INTO TABLE sales;
```

Loading Data From Files (2)

- **Add the LOCAL keyword to load data from the local disk**
  - Copies a local file (or directory) to the table's directory in HDFS

```
hive> LOAD DATA LOCAL INPATH '/home/bob/sales.txt'
      INTO TABLE sales;
```

  - This is equivalent to the following **hadoop fs** command

```
$ hadoop fs -put /home/bob/sales.txt \
 /user/hive/warehouse/sales
```

Loading Data From Files (3)

- **Add the `OVERWRITE` keyword to delete all records before import**
  - Removes all files within the table's directory
  - Then moves the new files into that directory

```
hive> LOAD DATA INPATH '/depts/finance/salesdata'
      OVERWRITE INTO TABLE sales;
```

Loading Data From a Relational Database

- **Sqoop has built-in support for importing data into Hive**
- **Just add the `--hive-import` option to your Sqoop command**
  - Creates the table in Hive (metastore)
  - Imports data from RDBMS to table's directory in HDFS

```
$ sqoop import \
  --connect jdbc:mysql://localhost/dualcore \
  --username training
  --password training \
  --fields-terminated-by '\t' \
  --table employees \
  --hive-import
```

Creating tables

using files outside of hive’s warehouse

Controlling a table's file location

- **Hive stores data associated with a table in its warehouse directory**
- **Storing data below Hive's warehouse is not always ideal**
  - Data might be shared by several users
- **Use `LOCATION` to specify directory where table data resides**

```
CREATE TABLE adclicks
    (campaign_id STRING,
     when TIMESTAMP,
     keyword STRING,
     site STRING,
     placement STRING,
     was_clicked BOOLEAN,
     cost SMALLINT)
  LOCATION '/dualcore/ad_data';
```

External Tables

- **Recall that dropping a table removes its data in HDFS**
- **Using `EXTERNAL` when creating the table avoids this behavior**
  - Dropping an external table removes only its *metadata*

```
CREATE EXTERNAL TABLE adclicks
    (campaign_id STRING,
     click_time TIMESTAMP,
     keyword STRING,
     site STRING,
     placement STRING,
     was_clicked BOOLEAN,
     cost SMALLINT)
  LOCATION '/dualcore/ad_data';
```

Altering tables

Removing a Table

- **Syntax is similar to database removal**

```
hive> DROP TABLE customers;
```

```
hive> DROP TABLE IF EXISTS customers;
```

- **Caution: These commands can remove data in HDFS!**
  - Hive does not have a rollback or undo feature

Renaming Tables and Columns

- **Use `ALTER TABLE` to modify a table's metadata**
  - This command does not change data in HDFS
- **Rename an existing table**

```
hive> ALTER TABLE customers RENAME TO clients;
```

- **Rename a column by specifying its old and new names**
  - Type must be specified even though it is unchanged

```
hive> ALTER TABLE clients
      CHANGE fname first_name STRING;
```

Old Name New Name Type

Modifying and Adding Columns

- **You can also modify a column's type**
  - The old and new column names will be the same
  - You must ensure data in HDFS conforms to the new type

```
hive> ALTER TABLE jobs CHANGE salary salary BIGINT;
```

Old Name New Name Type

- **Add new columns to a table**
  - Appended to the end of any existing columns
  - Existing data will have **NULL** values for new columns

```
hive> ALTER TABLE jobs
      ADD COLUMNS (city STRING, bonus INT);
```

Reordering and Removing Columns

- **Use AFTER or FIRST to reorder columns**
  - Ensure that your data in HDFS matches the new order

```
hive> ALTER TABLE jobs
      CHANGE bonus bonus INT AFTER salary;
```

```
hive> ALTER TABLE jobs
      CHANGE bonus bonus INT FIRST;
```

- **Use REPLACE COLUMNS to remove columns**
  - Any column not listed will be dropped from metadata

```
hive> ALTER TABLE jobs REPLACE COLUMNS
      (id INT,
       title STRING,
       salary INT);
```

File Formats and SerDe

Intro to SerDe in Hive

- SerDe classes are used when importing/exporting data into Hive
- org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
  - **default** - supports all datatypes in Hive, converts to Writables
  - ROW FORMATTED DELIMITED uses LazySimpleSerDe
- org.apache.hadoop.hive.contrib.serde2.RegexSerDe
  - parse file data into tables by using regular expressions

Regex SerDe

- **We sometimes need to analyze data that lacks consistent delimiters**
  - Log files are a common example of this

```
05/23/2013 19:45:19 312-555-7834 CALL_RECEIVED ""
05/23/2013 19:45:23 312-555-7834 OPTION_SELECTED "Shipping"
05/23/2013 19:46:23 312-555-7834 ON_HOLD ""
05/23/2013 19:47:51 312-555-7834 AGENT_ANSWER "Agent ID N7501"
05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received"
05/23/2013 19:48:41 312-555-7834 CALL_END "Duration: 3:22"
```

- **`RegexSerDe` will read records based on supplied regular expression**
  - Allows us to create a table from this log file

Creating a Table with Regex SerDe (1)

Log excerpt

```
05/23/2013 19:45:19 312-555-7834 CALL_RECEIVED ""
05/23/2013 19:48:37 312-555-7834 COMPLAINT "Item not received"
```

RegexSerDe

```
CREATE TABLE calls (
   event_date STRING,
   event_time STRING,
   phone_num STRING,
   event_type STRING,
   details STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" =
      "([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) \"([^\"]*)\"");
```

- **Each pair of parentheses denotes a field**
  - Field value is text matched by pattern within parentheses

Fixed-Width Formats in Hive

- **Many older applications produce data in fixed-width formats**

- **Unfortunately, Hive doesn't directly support these**
  - But you can overcome this limitation by using `RegexSerDe`
- **Caveat: all fields in `RegexSerDe` are of type `STRING`**
  - May need to cast numeric values in your queries

Fixed-Width Format Example

Input data

```
1030929610759620120829012215Oakland              CA94618
```

RegexSerDe

```
CREATE TABLE fixed (
   cust_id STRING,
   order_id STRING,
   order_dt STRING,
   order_tm STRING,
   city STRING,
   state STRING,
   zip STRING)
  ROW FORMAT SERDE
    'org.apache.hadoop.hive.serde2.RegexSerDe'
    WITH SERDEPROPERTIES ("input.regex" =
       "(\\d{7})(\\d{7})(\\d{8})(\\d{6})(.{20})(\\w{2})(\\d{5})");
```

cust_id	order_id	order_dt	order_tm	city	state	zipcode
1030929	6107596	20120829	012215	Oakland	CA	94618

ParquetSerDe

- parquet.hive.serde.ParquetHiveSerDe
  - Parquet is used to import/export Parquet files in/out of Hive.
  - Parquet is a compressed, column-store file format
  - Widely used, developed at Twitter
    - https://github.com/Parquet/parquet-format
  - Speed and space-saving advantages compared to regular files

Example: RegexSerDe to create apachelog

```
CREATE EXTERNAL TABLE apachelog (
        host STRING,
        identity STRING,
        user STRING,
        time STRING,
        request STRING,
        status STRING,
        size STRING,
        referrer STRING,
        agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^
\"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)
(?: ([^ \"]*|\"[^\"]*\") ([^\"]*|\"[^\"]*\"))?",
  "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s")
STORED AS TEXTFILE
LOCATION '/user/root/logs';
```

Example: ParquetHiveSerde - using CTAS from apachelog

```
CREATE TABLE apachelog_parquet
ROW FORMAT SERDE 'parquet.hive.serde.ParquetHiveSerDe'
    STORED AS
    INPUTFORMAT "parquet.hive.DeprecatedParquetInputFormat"
    OUTPUTFORMAT "parquet.hive.DeprecatedParquetOutputFormat"
AS SELECT
       host,
       identity,
       user,
       status,
       size,
       referrer,
       agent
FROM   apachelog;
```

AvroSerDe

The AvroSerde allows users to read or write Avro data as Hive tables.

- **Infers the schema of the Hive table from the Avro schema.**
- **Reads all Avro files within a table using the specified schema.**
- **Translates all Avro data types into equivalent Hive types.**
- **Understands compressed Avro files.**
- **Writes any Hive table to Avro files.**

Example: creating an external table with AvroSerDe

```
CREATE EXTERNAL TABLE categories
    ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
    STORED AS INPUTFORMAT
    'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
    OUTPUTFORMAT
    'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
    LOCATION 'hdfs:///user/hive/warehouse/categories'
    TBLPROPERTIES
    ('avro.schema.url'='hdfs:///user/examples/categories.avsc');
```

For general information about SerDes,

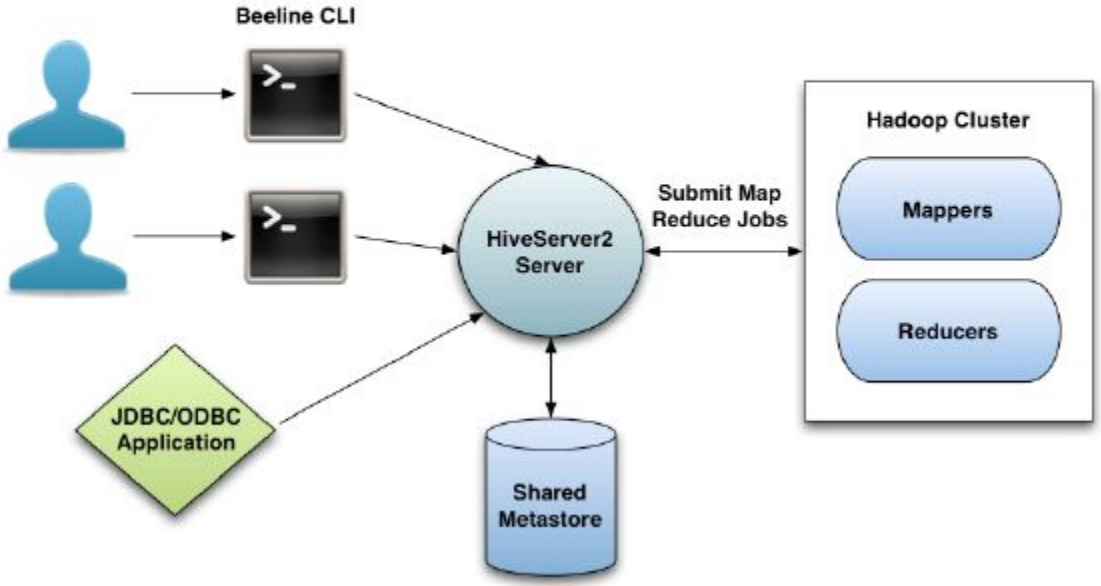
see Hive SerDe in the Developer Guide.

Also see SerDe for details about input and output processing.

Extra slides: secure Hive

Interacting with HiveServer2 - Hive as a Service (1)

- **HiveServer2 can be deployed to provide a centralized Hive service**
  - Uses a JDBC or ODBC connection
  - Supports Kerberos authentication



Interacting with HiveServer2 - Hive as a Service (2)

- **To connect to HiveServer2, use Hue or the Beeline CLI**
  - You cannot use the Hive shell
  - For secure deployments, supply your user ID and password
- **Example: starting Beeline and connecting to HiveServer2**

```
[training@localhost analyst]$ beeline
Beeline version 0.10.0-cdh4.2.1 by Apache Hive

beeline> !connect jdbc:hive2://localhost:10000
training mypwd org.apache.hive.jdbc.HiveDriver

Connecting to jdbc:hive2://localhost:10000
Connected to: Hive (version 0.10.0)
Driver: Hive (version 0.10.0-cdh4.2.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ

beeline>
```

Extra slides: avro

Avro file - companyInfo.avsc

```
{
  "type": "record",
  "name": "CompanyInfo",
  "fields": [
      {"name": "Symbol", "type": "string"},
      {"name": "Name", "type": "string"},
      {"name": "LastSale", "type": "string"},
      {"name": "MarketCap", "type": "string"},
      {"name": "IPOyear", "type": "string"},
      {"name": "Sector", "type": "string"},
      {"name": "Industry", "type": "string"},
      {"name": "SummaryQuoteURL", "type": "string"}
  ]
}
```

This file is in src/main/resources/avro

Read and parse schema (.avsc)

Create writer

Create record

Read text fields and write to record

```
private static void writeAvro(File inputFile, File outputFile) throws IOException {

    InputStream schemaIS = CompanyInfoAvroUtility.class
            .getResourceAsStream(SchemaConstants.COMPANYINFO_AVRO_SCHEMA);
    if (schemaIS == null) {
        throw new IllegalStateException("Unable to find " + SchemaConstants.COMPANYINFO_AVRO_SCHEMA);
    }
    Schema companyInfoSchema = new Parser().parse(schemaIS);

    DatumWriter<GenericRecord> writer = new GenericDatumWriter<GenericRecord>(companyInfoSchema);
    DataFileWriter<GenericRecord> dataFileWriter = new DataFileWriter<GenericRecord>(writer);
    dataFileWriter.setCodec(CodecFactory.deflateCodec(9));
    dataFileWriter.create(companyInfoSchema, outputFile);

    BufferedReader reader = new BufferedReader(new FileReader(inputFile));
    String line = null;
    GenericData.Record record = new GenericData.Record(companyInfoSchema);

    System.out.println("******** writing data in Avro format **********");
    while ((line = reader.readLine()) != null) {

        String[] tokens = line.split(regex1, -1);

        for (int i = 0; i < tokens.length; i++)
            tokens[i] = tokens[i].replace("\"", "");

        if (tokens.length == 9) {
            record.put("Symbol", new Utf8(tokens[0]));
            record.put("Name", new Utf8(tokens[1]));
            record.put("LastSale", new Utf8(tokens[2]));
            record.put("MarketCap", new Utf8(tokens[3]));
            record.put("IPOyear", new Utf8(tokens[4]));
            record.put("Sector", new Utf8(tokens[5]));
            record.put("Industry", new Utf8(tokens[6]));
            record.put("SummaryQuoteURL", new Utf8(tokens[7]));
            dataFileWriter.append(record);
        }
    }

    reader.close();
    dataFileWriter.close();

    System.out.println("******** finished writing data in Avro format to " + outputFile + " **********");
}
```