



Session 9

Motivation

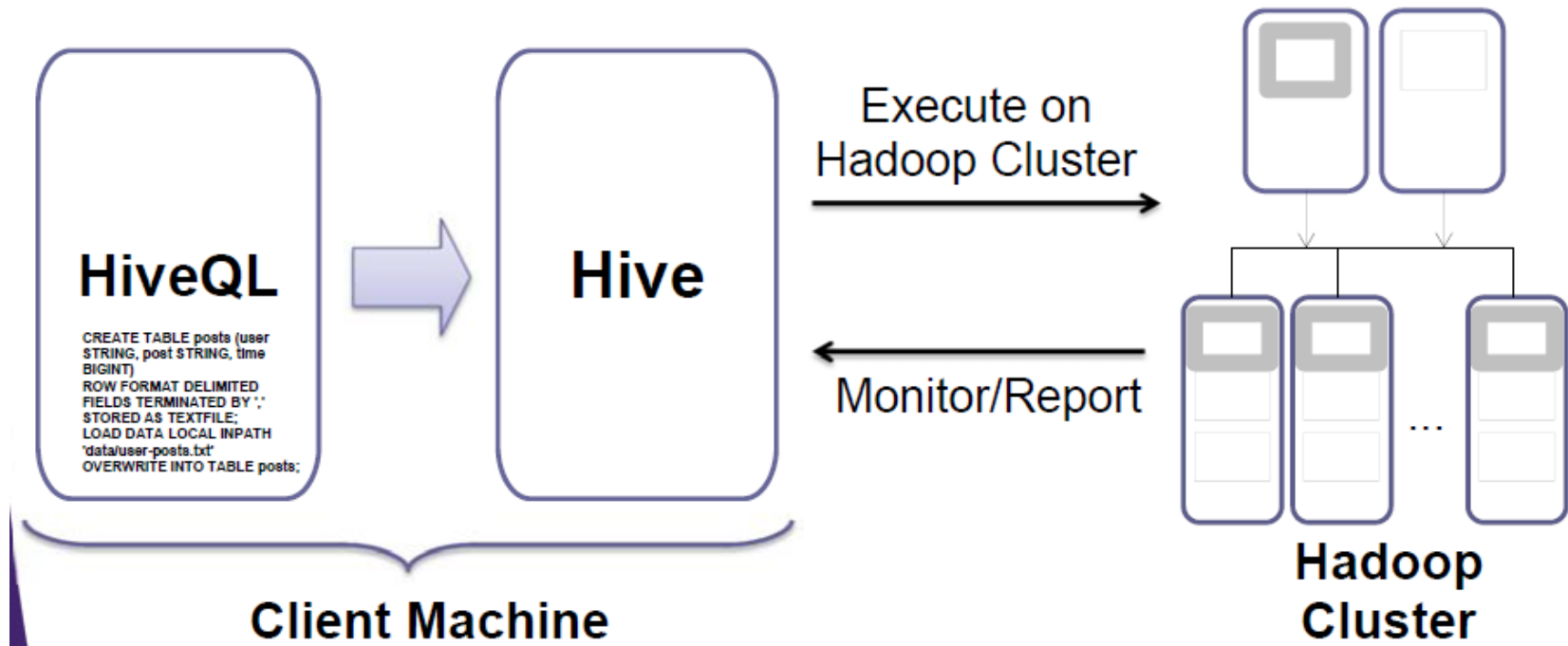
- **Limitation of MR**
 - Have to use M/R model
 - Java knowledge required (streaming ??)
 - Not Reusable
 - Error prone
 - For complex jobs:
 - Multiple stage of Map/Reduce functions
 - Just like ask dev to write specify physical execution plan in the database

Overview

- A data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
 - ETL.
 - Structure.
 - Access to different storage.
 - Query execution via MapReduce.
- Key Building Principles:
 - SQL is a familiar language
 - Extensibility – Types, Functions, Formats, Scripts
 - Performance
- Early Hive development work started at Facebook in 2007
- Today Hive is an Apache project under Hadoop

Transformation

- Translates HiveQL statements into a set of MapReduce Jobs which are then executed on a Hadoop Cluster



What NOT?

- Hive does NOT provide low latency or realtime queries
- Even querying small amounts of data may take minutes
- Designed for scalability and ease-of-use rather than low latency responses

Hive “One Shot” Commands

- CLI accepts a **-e command** in which CLI exits immediately as soon queries are executed

```
$ hive -e "SELECT * FROM mytable LIMIT 3";
```

- Adding the **-S for silent mode** removes the OK and Time taken ... lines, as well as other inessential output,

```
$ hive -S -e "select * FROM mytable LIMIT 3" > /tmp/myquery → can move the output to file
```

- Hive can execute one or more queries that were saved to a file using the **-f file argument**. By convention, saved Hive query files use the **.q or .hql extension**.

```
$ hive -f /path/to/file/withqueries.hql
```

- If you are already inside the Hive shell you can use the **SOURCE** command to execute a script file.

```
hive> source /path/to/file/withqueries.hql;
```

- **-i file option**, which lets you specify a file of commands for the CLI to run as it starts, before showing you the prompt.

.hiverc file

- Hive automatically looks for a file named *.hiverc* in your *HOME* directory and **runs the commands it contains**, if any.
 - Can be used to set user-specific properties → IMP
- These files are convenient for commands that you run frequently, such as setting system properties or adding Java archives (JAR files) of custom Hive extensions to Hadoop's distributed cache.
 - Use semicolon at the end of each line**
- ***Shell Execution::*** You don't need to leave the hive CLI to run simple bash shell commands. Simply type ! followed by the command and terminate the line with a semicolon (;)

Text File Encoding of Data Values

- Hive uses various control characters by default, which are less likely to appear in value strings.

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

- All format defaults:
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED BY '\002'
MAP KEYS TERMINATED BY '\003'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

Schema on Read

- When you write data to a traditional database, the database has total control over the storage. An important implication of this control is that the database can enforce the schema as data is *written*. This is called ***schema on write***.
- Hive has no such control over the underlying storage. There are many ways to create, modify, and even damage the data that Hive will query. Therefore, Hive can only enforce queries on *read*. This is called ***schema on read***.
- **So what if the schema doesn't match the file contents?** Hive does the best that it can to read the data else it will replace the NULL where doesn't matches (i.e. during schema violation)

Customizing Table Storage Formats

IMP → Hive uses,

- an ***input format*** to split input streams into records,
- an ***output format*** to format records into output streams (i.e., the output of queries),
- and a ***SerDe*** to parse records into columns, when **reading**, **AND**
- **encodes columns** into records, when **writing**.

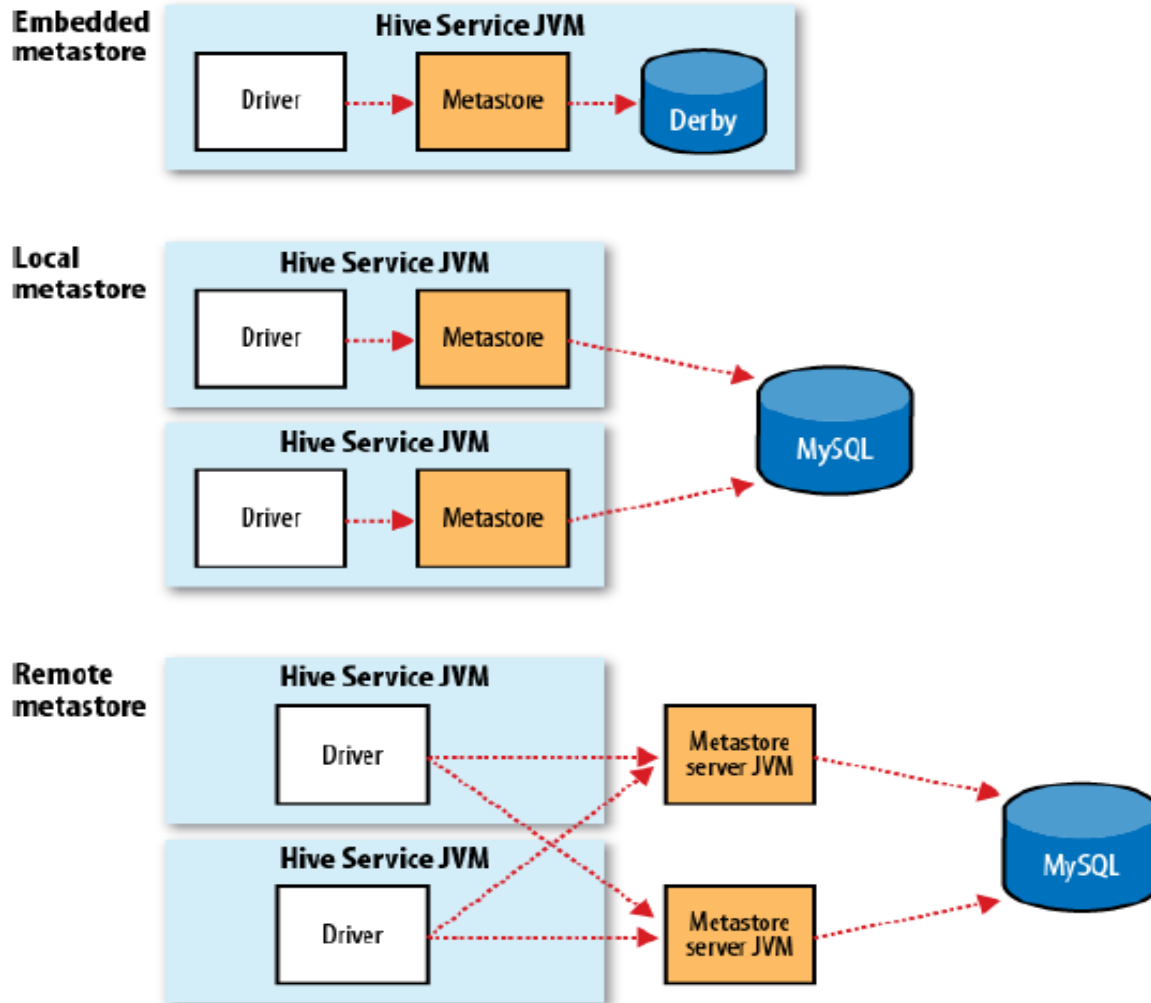
Example– Using a custom SerDe, i/p and o/p format:

```
CREATE TABLE kst
PARTITIONED BY (ds string)
ROW FORMAT SERDE 'com.linkedin.haivvreo.AvroSerDe'
WITH SERDEPROPERTIES ('schema.url'='http://schema_provider/kst.avsc')
STORED AS
INPUTFORMAT 'com.linkedin.haivvreo.AvroContainerInputFormat'
OUTPUTFORMAT 'com.linkedin.haivvreo.AvroContainerOutputFormat';
```

Hive MetaStore

- To support features like schema(s) and data partitioning Hive keeps its metadata in a Relational Database
 - Packaged with Derby, a lightweight embedded SQL DB
 - Default Derby based is good for evaluation and testing
 - Schema is not shared between users as each user has their own instance of embedded Derby
 - Stored in metastore_db directory which resides in the directory that hive was started from
- Can easily switch another SQL installation such as MySQL

MetaStore Configurations



Hive Installation

- **Set \$HADOOP_HOME environment variable**
 - Was done as a part of HDFS installation
- **Set \$HIVE_HOME and add hive to the PATH**
- **Hive will store its tables on HDFS and those locations needs to be bootstrapped**

```
export HIVE_HOME=$CDH_HOME/hive-0.8.1-cdh4.0.0
```

```
export PATH=$PATH:$HIVE_HOME/bin
```

```
$ hdfs dfs -mkdir /tmp
```

```
$ hdfs dfs -mkdir /user/hive/warehouse
```

```
$ hdfs dfs -chmod g+w /tmp
```

```
$ hdfs dfs -chmod g+w /user/hive/warehouse
```

- **Similar to other Hadoop's projects Hive's configuration is in \$HIVE_HOME/conf/hivesite.xml**

Type System

- Primitive types
 - Integers: TINYINT, SMALLINT, INT, BIGINT.
 - Boolean: BOOLEAN.
 - Floating point numbers: FLOAT, DOUBLE .
 - String: STRING.
- Complex types
 - Structs: {a INT; b INT}.
 - Maps: M['group'].
 - Arrays: ['a', 'b', 'c'], A[1] returns 'b'.

Built In Functions

- Mathematical: round, floor, ceil, rand, exp...
- Collection: size, map_keys, map_values, array_contains.
- Type Conversion: cast.
- Date: from_unixtime, to_date, year, datediff...
- Conditional: if, case, coalesce.
- String: length, reverse, upper, trim...

Serialization/DeSerialization

- Ser/De
 - Describe how to load the data from the file into a representation that make it looks like a table;
- Lazy load (Default – LazySerDe)
 - Create the field object when necessary
 - Reduce the overhead to create unnecessary objects in Hive
 - Java is expensive to create objects
 - Increase performance

Hive File Format

- Hive lets users store different file formats
- Helps in performance improvements
- Default – TEXTFILE FORMAT
- SQL Example:

```
CREATE TABLE dest1(key INT, value STRING)
STORED AS
INPUTFORMAT
'org.apache.hadoop.mapred.SequenceFileInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.mapred.SequenceFileOutputFormat'
```

Simple Example

1. Create a Table
2. Load Data into a Table
3. Query Data
4. Drop the Table

1. Create a Table

Launch Hive Command Line Interface (CLI)

```
$ cd $PLAY_AREA
```

Location of the session's log file

```
$ hive
```

```
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201208022144_2014345460.txt
```

```
hive> !cat data/user-posts.txt;
```

```
user1,Funny Story,1343182026191
```

```
user2,Cool Deal,1343182133839
```

```
user4,Interesting Post,1343182154633
```

```
user5,Yet Another Blog,13431839394
```

```
hive>
```

Can execute local commands within CLI, place a command in between ! and ;

Values are separate by ',' and each row represents a record; first value is user name, second is post content and third is timestamp

1. Create a Table

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
OK
Time taken: 10.606 seconds
```

1st line: creates a table with 3 columns
2nd and 3rd line: how the underlying file should be parsed
4th line: how to store data

Statements must end with a semicolon and can span multiple rows

```
hive> show tables;
OK
posts
Time taken: 0.221 seconds
```

Display all of the tables

Result is displayed between "OK" and "Time taken..."

```
hive> describe posts;
OK
user      string
post      string
time      bigint
Time taken: 0.212 seconds
```

Display schema for `posts` table

2. Load Data Into a Table

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'  
      > OVERWRITE INTO TABLE posts;  
Copying data from file:/home/hadoop/Training/play_area/data/user-posts.txt  
Copying file: file:/home/hadoop/Training/play_area/data/user-posts.txt  
Loading data to table default.posts  
Deleted /user/hive/warehouse/posts  
OK  
Time taken: 5.818 seconds  
hive>
```

Existing records the table *posts* are deleted; data in *user-posts.txt* is loaded into Hive's *posts* table

```
$ hdfs dfs -cat /user/hive/warehouse/posts/user-posts.txt  
user1,Funny Story,1343182026191  
user2,Cool Deal,1343182133839  
user4,Interesting Post,1343182154633  
user5,Yet Another Blog,13431839394
```

Under the covers Hive stores its tables in */user/hive/warehouse* (unless configured differently)

3. Query Data

```
hive> select count (1) from posts; ← Count number of records in posts table
Total MapReduce jobs = 1 ← Transformed HiveQL into 1 MapReduce Job
Launching Job 1 out of 1
...
Starting Job = job_1343957512459_0004, Tracking URL =
http://localhost:8088/proxy/application_1343957512459_0004/
Kill Command = hadoop job -Dmapred.job.tracker=localhost:10040 -kill
job_1343957512459_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-08-02 22:37:24,962 Stage-1 map = 0%, reduce = 0%
2012-08-02 22:37:30,497 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:31,577 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-08-02 22:37:32,664 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.64 sec
MapReduce Total cumulative CPU time: 2 seconds 640 msec
Ended Job = job_1343957512459_0004
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.64 sec HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 640 msec
OK
4 ← Result is 4 records
Time taken: 14.204 seconds
```

3. Query Data

```
hive> select * from posts where user="user2";  
...  
...  
OK  
user2 Cool Deal 1343182133839  
Time taken: 12.184 seconds
```

Select records for "user2"

Select records whose timestamp is less or equals to the provided value

```
hive> select * from posts where time<=1343182133839 limit 2;  
...  
...  
OK  
user1 Funny Story 1343182026191  
user2 Cool Deal 1343182133839  
Time taken: 12.003 seconds  
hive>
```

Usually there are too many results to display, then one could utilize `limit` command to bound the display

4. Drop The Table

```
hive> DROP TABLE posts; ← Remove the table; use with caution
OK
Time taken: 2.182 seconds
```

```
hive> exit;
```

```
$ hdfs dfs -ls /user/hive/warehouse/
$
```

←
If hive was managing underlying file then it will be removed

Loading Data

- **Several options to start using data in HIVE**

- Load data from HDFS location

```
hive> LOAD DATA INPATH '/training/hive/user-posts.txt'  
> OVERWRITE INTO TABLE posts;
```

- File is copied from the provided location to /user/hive/warehouse/ (or configured location)

- Load data from a local file system

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts.txt'  
> OVERWRITE INTO TABLE posts;
```

- File is copied from the provided location to /user/hive/warehouse/ (or configured location)

- Utilize an existing location on HDFS

- Just point to an existing location when creating a table

External Table


```
hive> CREATE EXTERNAL TABLE posts
  > (user STRING, post STRING, time BIGINT)
  > ROW FORMAT DELIMITED
  > FIELDS TERMINATED BY ','
  > STORED AS TEXTFILE
  > LOCATION '/training/hive/';
```

OK

Time taken: 0.077 seconds

hive>

Hive will load all the files under
/training/hive directory in posts table



Partitions

- **To increase performance Hive has the capability to partition data**
 - The values of partitioned column divide a table into segments
 - Entire partitions can be ignored at query time
 - Similar to relational databases' indexes but not as granular
- **Partitions have to be properly created by users**
 - When inserting data must specify a partition
- **At query time, whenever appropriate, Hive will automatically filter out partitions**

Creating Partitioned Tables

```
hive> CREATE TABLE posts (user STRING, post STRING, time BIGINT)
> PARTITIONED BY(country STRING)
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
```

Partition table based on the value of a country.

```
OK
Time taken: 0.116 seconds
```

```
hive> describe posts;
```

```
OK
user      string
post      string
time      bigint
country   string
Time taken: 0.111 seconds
```

There is no difference in schema between "partition" columns and "data" columns

```
hive> show partitions posts;
```

```
OK
Time taken: 0.102 seconds
hive>
```

Load Data Into Partitioned Table

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'  
      > OVERWRITE INTO TABLE posts;  
FAILED: Error in semantic analysis: Need to specify partition  
columns because the destination table is partitioned
```

Since the posts table was defined to be partitioned
any insert statement must specify the partition

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-US.txt'  
      > OVERWRITE INTO TABLE posts PARTITION(country='US');
```

OK

Time taken: 0.225 seconds

```
hive> LOAD DATA LOCAL INPATH 'data/user-posts-AUSTRALIA.txt'  
      > OVERWRITE INTO TABLE posts PARTITION(country='AUSTRALIA');
```

OK

Time taken: 0.236 seconds

hive>

Each file is loaded into separate partition;
data is separated by country

Partitioned Table

- **Partitions are physically stored under separate directories**
-

```
hive> show partitions posts;  
OK  
country=AUSTRALIA  
country=US  
Time taken: 0.095 seconds  
hive> exit;
```


There is a directory for
each partition value

```
$ hdfs dfs -ls -R /user/hive/warehouse/posts  
/user/hive/warehouse/posts/country=AUSTRALIA  
/user/hive/warehouse/posts/country=AUSTRALIA/user-posts-AUSTRALIA.txt  
/user/hive/warehouse/posts/country=US  
/user/hive/warehouse/posts/country=US/user-posts-US.txt
```

Querying Partitioned Table

- There is no difference in syntax
 - When partitioned column is specified in the where clause entire directories/partitions could be ignored
-

Only "COUNTRY=US" partition will be queried,
"COUNTRY=AUSTRALIA" partition will be ignored



```
hive> select * from posts where country='US' limit 10;
OK
user1 Funny Story 1343182026191      US
user2 Cool Deal 1343182133839      US
user2 Great Interesting Note 13431821339485      US
user4 Interesting Post 1343182154633      US
user1 Humor is good 1343182039586      US
user2 Hi I am user #2 1343182133839      US
Time taken: 0.197 seconds
```

Bucketing

- **Mechanism to query and examine random samples of data**
 - Break data into a set of buckets based on a hash function of a "bucket column"
 - Capability to execute queries on a sub-set of random data
- **Doesn't automatically enforce bucketing**
 - User is required to specify the number of buckets by setting # of reducer

```
hive> mapred.reduce.tasks = 256;  
OR  
hive> hive.enforce.bucketing = true;
```



Either manually set the # of reducers to be the number of buckets or you can use 'hive.enforce.bucketing' which will set it on your behalf

Create and Use Table with Buckets

```
hive> CREATE TABLE post_count (user STRING, count INT)
      > CLUSTERED BY (user) INTO 5 BUCKETS;
```

OK
Time taken: 0.076 seconds

← Declare table with 5 buckets for user column

```
hive> set hive.enforce.bucketing = true; ← # of reducer will get set 5
hive> insert overwrite table post_count
      > select user, count(post) from posts group by user;
```

Total MapReduce jobs = 2
Launching Job 1 out of 2
...
Launching Job 2 out of 2
...
OK
Time taken: 42.304 seconds
hive> exit;

← Insert data into post_count bucketed table; number of posts are counted up for each user

```
$ hdfs dfs -ls -R /user/hive/warehouse/post_count/
/user/hive/warehouse/post_count/000000_0
/user/hive/warehouse/post_count/000001_0
/user/hive/warehouse/post_count/000002_0
/user/hive/warehouse/post_count/000003_0
/user/hive/warehouse/post_count/000004_0
```

← A file per bucket is created; now only a sub-set of buckets can be sampled

Random Sample of Bucketed Table

```
hive> select * from post_count TABLESAMPLE(BUCKET 1 OUT OF 2);  
OK  
user5 1  
user1 2  
Time taken: 11.758 seconds  
hive>
```

Sample approximately 1 for every 2 buckets



Joins

- **Joins in Hive are trivial**
- **Supports outer joins**
 - left, right and full joins
- **Can join multiple tables**
- **Default Join is Inner Join**
 - Rows are joined where the keys match
 - Rows that do not have matches are not included in the result

Simple Inner Join

- **Let's say we have 2 tables: posts and likes**
-

```
hive> select * from posts limit 10;
```

```
OK
```

```
user1    Funny Story      1343182026191
user2    Cool Deal        1343182133839
user4    Interesting Post  1343182154633
user5    Yet Another Blog 1343183939434
```

```
Time taken: 0.108 seconds
```

```
hive> select * from likes limit 10;
```

```
OK
```

```
user1    12      1343182026191
user2    7       1343182139394
user3    0       1343182154633
user4    50      1343182147364
```

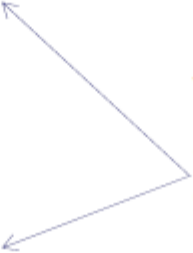
```
Time taken: 0.103 seconds
```

```
hive> CREATE TABLE posts_likes (user STRING, post STRING, likes_count INT);
```

```
OK
```

```
Time taken: 0.06 seconds
```

We want to join these 2 data-sets and produce a single table that contains user, post and count of likes



Simple Inner Join

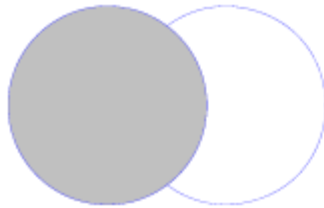
```
hive> INSERT OVERWRITE TABLE posts_likes  
      > SELECT p.user, p.post, l.count  
      > FROM posts p JOIN likes l ON (p.user = l.user);  
OK  
Time taken: 17.901 seconds
```

Two tables are joined based on user column; 3 columns are selected and stored in posts_likes table

```
hive> select * from posts_likes limit 10;  
OK  
user1 Funny Story          12  
user2 Cool Deal            7  
user4 Interesting Post     50  
Time taken: 0.082 seconds  
hive>
```

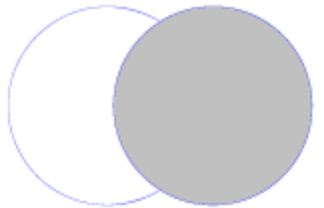
Outer Join

- Rows which will not join with the 'other' table are still included in the result



Left Outer

- Row from the first table are included whether they have a match or not. Columns from the unmatched (second) table are set to null.



Right Outer

- The opposite of Left Outer Join: Rows from the second table are included no matter what. Columns from the unmatched (first) table are set to null.



Full Outer

- Rows from both sides are included. For unmatched rows the columns from the 'other' table are set to null.

Outer Join Examples

```
SELECT p.*, l.*  
FROM posts p LEFT OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```

```
SELECT p.*, l.*  
FROM posts p RIGHT OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```

```
SELECT p.*, l.*  
FROM posts p FULL OUTER JOIN likes l ON (p.user = l.user)  
limit 10;
```

Pros & Cons

- Pros
 - A easy way to process large scale data
 - Support SQL-based queries
 - Provide more user defined interfaces to extend
 - Programmability
 - Interoperability with other database tools
- Cons
 - No easy way to append data
 - Files in HDFS are immutable
 - Accepts only a subset of SQL queries

Application

- Log processing
 - Daily Report
 - User Activity Measurement
- Data/Text mining
 - Machine learning (Training Data)
- Business intelligence
 - Advertising Delivery
 - Spam Detection

Thank You