History of Hive

- Initially developed at Facebook in 2007 to handle massive amounts of growth.
  - Dataset growth from 15TB to 700TB over a few years
  - RDBMS Data Warehouse was taking too long to process daily jobs
  - Moved their data into scalable open-source Hadoop environment
  - Using Hadoop / creating MapReduce programs was not easy for many users
  - Vision: Bring familiar database concepts to the unstructured world of Hadoop, while still maintaining Hadoop's extensibility and flexibility
  - Hive was open sourced in August 2008
  - Currently used at Facebook for reporting dashboards and ad-hoc analysis

What is Hive?

- Data Warehouse system built on top of Hadoop
  - Takes advantage of Hadoop distributed processing power
- Facilitates easy data summarization, ad-hoc queries, analysis of large datasets stored in Hadoop
- Hive provides a SQL interface (HQL) for data stored in Hadoop
  - Familiar, Widely known syntax
  - Data Definition Language and Data Manipulation Language
- HQL queries implicitly translated to one or more Hadoop MapReduce job(s) for execution
  - Saves you from having to write the MapReduce programs!
  - Clear separation of defining the *what* (you want) vs. the *how* (to get it)
- Hive provides mechanism to project structure onto Hadoop datasets
  - Catalog ("metastore") maps file structure to a tabular form

What Hive is not…

- **Hive is not a full database - but it fits alongside your RDBMS.**
- Is not a real-time processing system
  - Best for heavy analytics and large aggregations – Think Data Warehousing.
  - Latencies are often much higher than RDBMS
  - Schema on Read
    - Fast loads and flexibility – at the cost of query time
    - Use RDBMS for fast run queries.
- Not SQL-92 compliant
  - Does not provide row level inserts, updates or deletes
  - Doesn't support transactions
  - Limited subquery support
- Query optimization still a work in progress
- See HBase for rapid queries and row-level updates and transactions

Hive versus Java and Pig

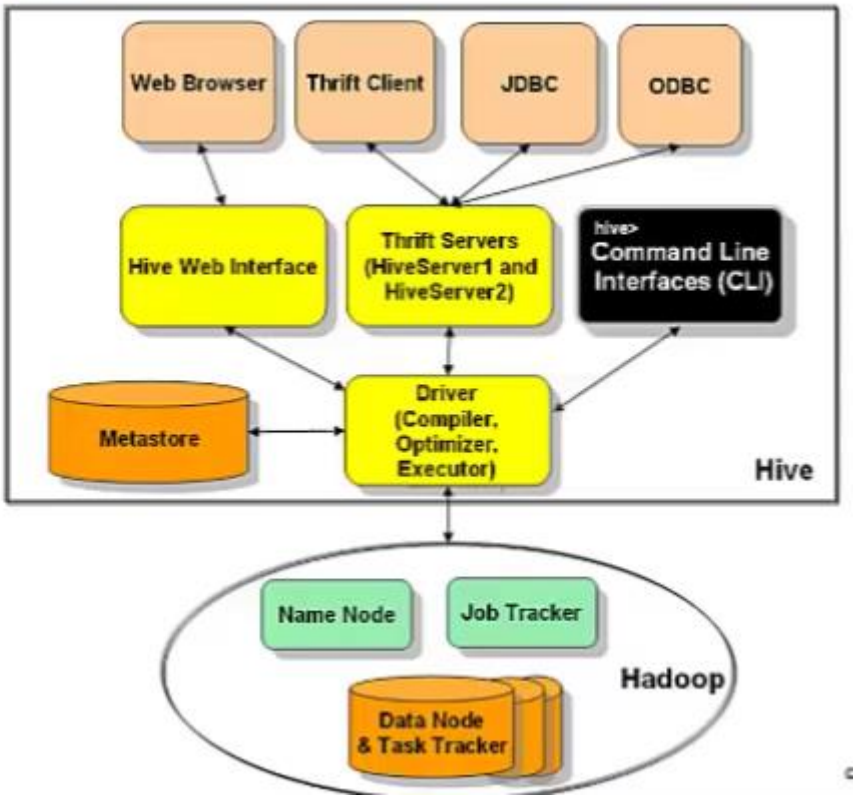
Java

- Word Count MapReduce example
  - Lists words and number of occurrences in a document
  - Takes 63 lines of Java code to write this.
  - Hive solution only takes 7 easy lines of code!

Pig

- High level programming language ("data flow language")
  - Higher learning curve for SQL programmers
- Good for ETL, not as good for ad-hoc querying
- Powerful transformation capabilities
- Often used in combination with Hive

Hive Components



Hive Directory Structure

- Lib directory
  - $HIVE_HOME/lib
  - Location of Hive JAR files
  - Contain the actual Java code that implement the Hive functionality
- Bin directory
  - $HIVE_HOME/bin
  - Location of Hive Scripts/Services
- Conf directory
  - HIVE_HOME/conf
  - Location of configuration files

CLI (Command Line Interface)

- Most common way to interact with Hive
- From the shell you can
  - Perform queries, DML, and DDL
  - View and manipulate table metadata
  - Retrieve query explain plans (execution strategy)
- The Hive Beeline shell and original CLI are located in **$HIVE_HOME**/bin/hive

Beeline CLI

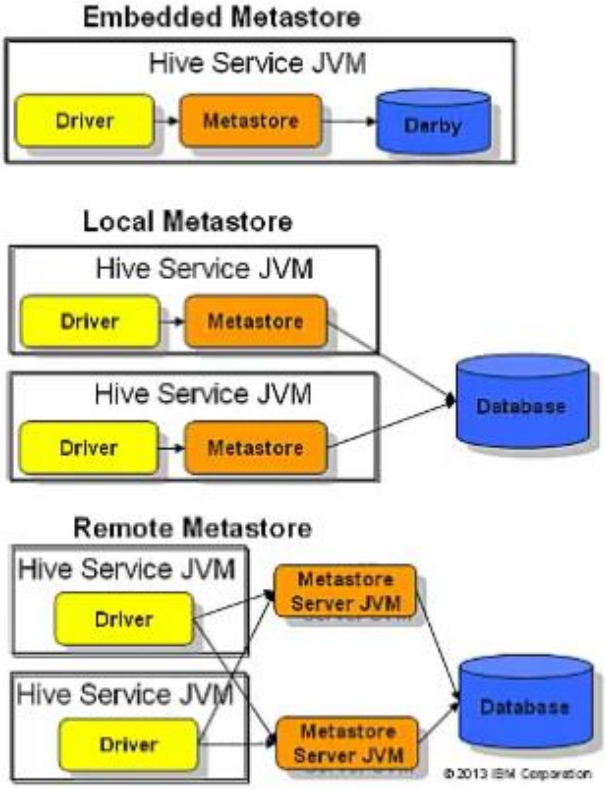
Original Hive CLI

```
$ $HIVE_HOME/bin/hive
2013-01-14 23:36:52.153 GMT : Connection obtained for host: master-
Logging initialized using configuration in file:/opt/ibm/biginsight
Hive history file=/var/ibm/biginsights/hive/query/biadmin/hive_job

hive> show tables;
mytab1
mytab2
mytab3
OK
Time taken: 2.987 seconds
hive> quit;
```


Metastore

- 2 pieces – Service & Datastore
- Stores Hive Metadata in 1 of 3 configs:
  - **Embedded:** in-process metastore, in-process database
  - **Local:** in-process metastore, out-of-process database
  - **Remote:** out-of-process metastore, out-of-process database
  - If metastore not configured - Derby database is used
    - Derby metastore allows only one user at a time
  - Can be configured to use a wide variety of storage options (DB2, MySQL, Oracle, XML files, etc.) for more robust metastore



Real world use cases

- **CNET: "We use Hive for data mining, internal log analysis and ad hoc queries."**
- **Digg: "We use Hive for data mining, internal log analysis, R&D, and reporting/analytics."**
- **Grooveshark: "We use Hive for user analytics, dataset cleaning, and machine learning R&D."**
- **Papertrail: "We use Hive as a customer-facing analysis destination for our hosted syslog and app log management service."**
- **Scribd: "We use hive for machine learning, data mining, ad-hoc querying, and both internal and user-facing analytics."**
- **VideoEgg: "We use Hive as the core database for our data warehouse where we track and analyze all the usage data of the ads across our network."**

Hive Data Units

- Organization of Hive data (order of granularity)

```
Database
  ->Table
    ->Partition
      ->Buckets
```

  - **Databases**: Namespaces that separate tables and other data units from naming confliction.
  - **Tables**: Homogeneous units of data which have the same schema.
  - **Partitions**: A virtual column which defines how data is stored on the file system based on its values. Each table can have one or more partitions (and one or more levels of partition).
  - **Buckets** (or **Clusters**): In each partition, data can be divided into buckets based on the hash value of a column in the table (useful for sampling, join optimization).
  - Note that it is not necessary for tables to be partitioned or bucketed, but these abstractions allow the system to prune large quantities of data during query processing, resulting in faster query execution.

Physical Layout – Data in Hive

Data files are just regular HDFS files
- Variety of storage and record formats can be used
- Internal format can vary table-to-table (delimited, sequence, etc.)

- Warehouse directory in HDFS
  - Specified by "*hive.metastore.warehouse.dir*" in hive-site.xml (can be overrid
  - E.g. /user/hive/warehouse
- One can think tables, partitions and buckets as directories, subdirectories and fil respectively

Hive Entity	Sample	Sample location *In this example $WH is a variable that holds warehouse path*
database	testdb	$WH/testdb.db
table	T	$WH/testdb.db/T
partition	date='01012013'	$WH/testdb.db/T/date=01012013

We could also Bucket...

Bucketing column	userid	$WH/testdb.db/T/date=01012013/000000_0 ... ... $WH/testdb.db/T/date=01012013/000032_0

Creating Databases

- **Create a database named "mydatabase"**

```
hive> CREATE DATABASE mydatabase;
```

- **Create a database named "mydatabase" and override the Hive warehouse configured location**

```
hive> CREATE DATABASE mydatabase
    > LOCATION '/myfavorite/folder/';
```

- **Create a database and add a descriptive comment**

```
hive> CREATE DATABASE mydatabase
    > COMMENT 'This is my database';
```

- **Create a database and some descriptive properties**

```
hive> CREATE DATABASE mydatabase
    > WITH DBPROPERTIES ('createdby' = 'bigdatauser', 'date' =
'2014-01-01');
```

Database Show/Describe

- List the Database(s) in the Hive system

```
hive> SHOW DATABASES;
```

- Show some basic information about a Database

```
hive> DESCRIBE DATABASE mydatabase;
```

- Show more detailed information about a Database

```
hive> DESCRIBE DATABASE EXTENDED mydatabase;
```

Database Use/Drop/Alter

- Hive "default" database is used if database is not specified

- Tell Hive that your following statements will use a specific database

```
hive> USE mydatabase;
```

- Delete a database

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

  – Note: Database directory is also deleted

- Alter a database

```
hive> ALTER DATABASE mydatabase SET DBPROPERTIES ('createdby' =
'bigdatauser2');
```

  – Note: Only possible to update the DBPROPRTIES metadata

Primitive Data Types

- Types are associated with the columns in the tables.
- Primitive types in Hive (subset of typical RDBMS types):
  - **Integers**
    - TINYINT - 1 byte integer
    - SMALLINT - 2 byte integer
    - INT - 4 byte integer
    - BIGINT - 8 byte integer
  - **Boolean type**
    - BOOLEAN - TRUE/FALSE
  - **Floating point numbers**
    - FLOAT - single precision
    - DOUBLE - Double precision
    - DECIMAL - provide precise values and greater range than Double
  - **String types**
    - STRING - sequence of characters in a specified character set
    - VARCHAR – specify length of character string (between 1 and 65,355)
  - **Date/Time types**
    - TIMESTAMP - YYYY-MM-DD HH:MM:SS.fffffffff
    - DATE - YYYY-MM-DD
  - **Binary type**
    - Binary - array of bytes (similar to VARBINARY in RDBMS)

  - On comparison, Hive does some implicit casting
    - Any Integer to larger of two Integer types
    - FLOAT to DOUBLE
    - Any integer to DOUBLE

Complex Data Types

- Complex Types can be built up from primitive types and other composite types.
- **Arrays** - Indexable lists containing elements of the same type.
  - Format: ARRAY<data_type>
  - Literal syntax example: array('user1', 'user2')
  - Accessing Elements: [n] notation where n is an index into the array. E.g. *arrayname[0]*
- **Structs** - Collection of elements of different types.
  - Format: STRUCT<col_name : data_type, ...>
  - Literal syntax example: struct('Jake', '213')
  - Accessing Elements: DOT (.) notation. E.g. *structname.firstname*
- **Maps** – Collection of key-value tuples.
  - Format: MAP<primitive_type, data_type>
  - Literal syntax example: map('business_title', 'CEO', 'rank', '1')
  - Accessing Elements: ['element name'] notation. E.g. *mapname['business_title']*
- **Union** – at any one point can hold exactly one of their specified data types.
  - Format: UNIONTYPE<data_type, data_type, ...>
  - Accessing Elements: Use the create_union UDF (see Hive docs for more info)

Creating a Table

- Creating a delimited table

```
hive> CREATE TABLE users
(
  id         INT,
  office_id  INT,
  name       STRING,
  children   ARRAY<STRING>
)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ':'
STORED AS TEXTFILE;
```

file: users.dat

```
1|1|Bob Smith|Mary
2|1|Frank Barney|James:Liz:Karen
3|2|Ellen Lacy|Randy:Martin
4|3|Jake Gray|
5|4|Sally Fields|John:Fred:Sue:Hank:Robert
```

- Inspecting tables:

```
hive> SHOW TABLES;
OK
users
Time taken: 2.542 seconds

hive> DESCRIBE users;
OK
id         int
office_id  int
name       string
children   array<string>
Time taken: 0.129 seconds

Note: Can also use "DESCRIBE EXTENDED tablename" for more metadata
```

Table partitioning

- Creating a partitioned table:

```
hive> CREATE TABLE users
(
  id         INT,
  office_id  INT,
  name       STRING,
  children   ARRAY<STRING>
)
PARTITIONED BY (division STRING)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ':'
STORED AS TEXTFILE;
```

Directories on file system would reflect the partitions:

$WH/mydatabase.db/users/division=div123

$WH/mydatabase.db/users/division=div567

$WH/mydatabase.db/users/division=div890

Database (directory) | Users table (directory) | Partition (directory)

- Table partitioned on "Division".
- PARTITION BY clause defines the virtual columns which are different from the data columns and are actually not stored with the data
- Our Hive queries can now take advantage of the partitioned data (each partition is a separate directory that stores the data for that partition)
  - Better performance for certain queries (WHERE clauses)

Managed Vs External Tables

Managed Tables

- By default Hive tables are "Managed".
  - Hive controls the metadata AND the lifecycle of the data
  - Data stored in subdirectories within the hive.metastore.warehouse.dir location
  - Dropping a managed table deletes the data in the table

External Tables

- Store table in a directory outside of Hive
- Useful if sharing your data with other tools
- Hive does not assume it owns the data in the table
  - Dropping table deletes the tables metadata - NOT the actual data
- Must add the EXTERNAL and LOCATION keywords to CREATE statement

```
CREATE EXTERNAL TABLE users
...
LOCATION '/path/to/your/data';
```

Drop/Alter Table

- **Delete a table**

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

- **Change name of table**

```
hive> ALTER TABLE users RENAME TO employees;
```

- **Add two columns to end of table**

```
hive> ALTER TABLE users ADD COLUMNS (
    location  STRING,
    age       INT);
```

- **Variety of other Alter statements**
  - Delete columns
  - Alter table properties
  - Alter storage properties
  - Alter partitions

Indexes

- Goal of Hive indexing: improve speed of query lookup on certain columns of a table.

- Speed improved at cost of processing and disk space (index data stored in another table)

- **Create an Index**

```
hive> CREATE INDEX table01_index ON TABLE table01 (column2) AS
  'COMPACT';
```

- **Show index**

```
hive> SHOW INDEX ON table01;
```

- **Delete an index**

```
hive> DROP INDEX table01_index ON table01;
```

- Variety of other Indexing topics including Bitmap indexes

Loading Data into Hive – From a File

- **Loading data from input file (Schema on Read)**

```
hive> LOAD DATA LOCAL INPATH '/tmp/data/users.dat'
      OVERWRITE INTO TABLE users;
```

  - The "LOCAL" indicates the source data is on the local filesystem
  - Local data is **copied** to final location
  - Otherwise file is assumed to be on HDFS and is **moved** to final location
  - Hive does not do any transformation while loading data into tables

- **Loading data into a partition requires PARTITION clause**

```
hive> LOAD DATA LOCAL INPATH '/tmp/data/usersny.dat'
      OVERWRITE INTO TABLE users;
      PARTITION (country = 'US', state = 'NY')
```

  - HDFS directory is created:
    /user/hive/warehouse/mydb.db/users/country=US/state=NY
    - usersny.dat file is copied to this HDFS directory

Loading Data from a Directory

- **Load data from an HDFS directory instead of single file**

```
hive> LOAD DATA INPATH '/user/biadmin/userdatafiles'
      OVERWRITE INTO TABLE users;
```

  - Lack of "LOCAL" keyword means source data is on the HDFS file system
  - Data is **moved** to final location
  - All of the files in the /user/biadmin/userdatafiles directory are copied over into Hive
  - OVERWRITE keyword causes contents of target table to be deleted and replaced.
    - Leaving out the OVERWRITE means files will be added to the table
    - If target has file name collision then new file will overwrite existing Hive file

Loading Data into Hive – From a Query

Tables may be created using queries on other tables

```
CREATE TABLE emps_by_state
AS
SELECT o.state AS state, count(*) AS employees
  FROM office o LEFT OUTER JOIN users  u
     ON u.office_id = o.office_id
GROUP BY o.state;
```

Or using INSERT OVERWRITE ...

```
CREATE table emps_by_state ... STORED AS textfile;

INSERT OVERWRITE TABLE emps_by_state
SELECT o.state AS state, count(*) AS employees
  FROM office o LEFT OUTER JOIN users u
     ON u.office_id = o.office_id
GROUP BY o.state;
```

Exporting Data out of Hive

- If data files are already format you like, can just copy them out of HDFS
- Query results can be inserted into file system directories (local or HDFS)
- If LOCAL keyword is used, Hive will write data to the directory on the local file system.

```
INSERT OVERWRITE LOCAL DIRECTORY '/mydirectory/dataexports'
SELECT sale_id, product, date
FROM sales
WHERE date='2014-01-01';
```

- Data written to the file system is by default serialized as text with columns separated by ^A and rows separated by newlines.
  - Non-primitive columns serialized to JSON format.
  - Delimiters and file format may be specified.

- INSERT OVERWRITE statements to HDFS is good way to extract large amounts of data from Hive. Hive can write to HDFS directories in parallel from within a MapReduce job.

- **Warning: The directory is OVERWRITTEN!**
  - if the specified path exists, it is clobbered and replaced with output.

SELECT FROM

users
id office_id name children

- If you know SQL, then HiveQL's DML has few surprises

- Simple SELECT all query:

```
hive> SELECT * FROM users LIMIT 3;
1       1       Bob Smith       ["Mary"]
2       1       Frank Barney    ["James","Liz","Karen"]
3       2       Ellen Lacy      ["Randy","Martin"]
```

  – Note the last column is ARRAY data type. Hive outputs the array elements in brackets.
  – LIMIT puts an upper limit on number rows returned

- SELECT query (ARRAY indexing):

```
hive> SELECT name, children[0] FROM users;
Bob Smith       Mary
Frank Barney    James
Ellen Lacy      Randy
```

  – First element from children ARRAY is selected.

Customer
id cust_name phone Address ytd_sales

- SELECT query with STRUCT column (address):

```
hive> SELECT cust_name, address FROM customer;
Bruce Smith     {"city":"Burlington","zipcode":05401}
Chuck Barney    {"city":"Jericho","zipcode":05465}
```

  – The address column (type STRUCT) prints out in JSON format

WHERE

- Predicate Operators
  =, <>, !=, <, <=, >, >=, IS NULL, IS NOT NULL, LIKE, RLIKE

- SELECT WHERE query with STRUCT column (address):

```
hive> SELECT cust_name, address.city FROM customer
      WHERE address.city = 'Burlington';
      Bruce Smith     Burlington
```

  – Dot notation used to access struct data

- SELECT WHERE query with GROUP BY clause:

```
hive> SELECT address.city, sum(ytd_sales) FROM customer
      WHERE address.city = 'Burlington'
      GROUP BY address.city;
```

  – GROUP BY usually used with aggregate functions
  – Can also use the HAVING clause with GROUP BY

Column Alias and Nested Select

- SELECT query with column alias

```
hive> SELECT cust_name, address.city as city
       FROM customer;
```

  – Address.city is given the column alias "city"

- Column alias comes in handy when doing nested SELECT...

```
hive> FROM(
           SELECT cust_name, round(ytd_sales * .01) as rewards
           FROM customer;
      ) c
      SELECT c.name, c.rewards
      WHERE c.rewards > 25;
```

  – The first query is aliased as "c". We then select from the results of that query.

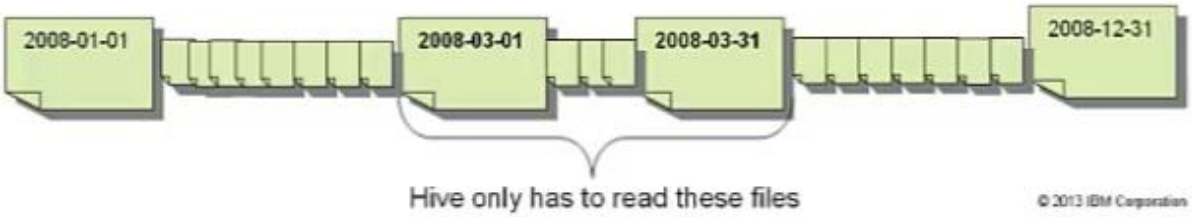
Selecting from Partitions

Taking advantage of partitions

- Partition pruning – Hive scans only a fraction of the table relevant to the partitions specified by the query.
- Hive does partition pruning if the partition predicates are specified in the WHERE clause or the ON clause in a JOIN.
- Example: If table `page_views` is partitioned on `log_date`, the following query retrieves rows for just days between `2008-03-01` and `2008-03-31`.

```
hive> SELECT * FROM page_views
       WHERE log_date >= '2008-03-01' AND log_date <= '2008-03-31';
```

  – Imagine we had a year of page_view data partitioned into 365 partition files. Hive only needs to open and read the 31 partitioned data files in the above example.

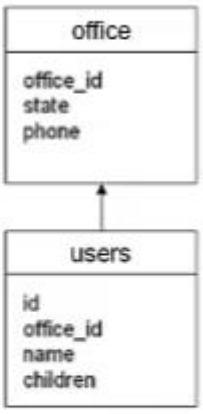


Joins

- Hive supports joins via ANSI join syntax only

```
select ...
  from T1, T2
 where T1.a = T2.b
```

```
select ...
  from T1 JOIN T2
       on T1.a = T2.b
```



- Example join

```
hive> SELECT o.state as state, count(*) as employees
  FROM office o LEFT OUTER JOIN users  u
     on u.office_id = o.office_id
GROUP BY o.state
ORDER BY state;
```

- Note Hive only supports equi-joins
- Inner Join, Left Outer, Right Outer, Full Outer, Left Semi-Join are supported
- Largest table on right for performance
- Limited support for subqueries, only permitted in the FROM clause of a SELECT statement
  - Sometimes can be rewritten using Joins

Order BY / SORT BY

- ORDER BY clause is similar to other versions of SQL
  - Performs total ordering of result set - through single reducer (Hadoop)
  - Large data sets = high latency
- Hive's SORT BY clause
  - Data is ordered in each reducer - each reducer's output is sorted
  - If multiple reducers - final results may not be sorted as desired

Casting

- Implicit conversions when compare one type to another
- Use the cast( ) function for explicitly conversions
- If salary is a STRING data type, we can explicitly cast it to FLOAT as seen below

```
hive> SELECT first_name, team
      FROM players
      WHERE cast(salary AS FLOAT) > 100.0;
```

If "salary" not a string that could be converted to a floating point number, then Null

Views

- Allow query to be saved and treated like table
- Logical – doesn't store data
- Hides complexity of long query
- Earlier example of a "complex" query

```
hive> FROM(
        SELECT cust_name, round(ytd_sales * .01) as rewards FROM customer;
      ) c
      SELECT c.name, c.rewards
      WHERE c.rewards > 25;
```

Can be simplified using a View….

```
hive> CREATE VIEW rewards_view AS
       SELECT cust_name, round(ytd_sales * .01) as rewards FROM customer;

hive> SELECT name, rewards FROM rewards_view
      WHERE rewards > 25;
```

- Drop a view

```
hive> DROP VIEW IF EXISTS rewards_view;
```

Explain

- The explain keyword generates a Hive explain plan for the query

```
hive> EXPLAIN SELECT o.state as state, count(*) as employees
  FROM office o LEFT OUTER JOIN users  u
     on u.office_id = o.office_id
GROUP BY o.state
ORDER BY state;

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-3 depends on stages: Stage-2
  Stage-0 is a root stage
...
```

- Query requires three MapReduce jobs to implement
  - One to join the two inputs
  - One to perform the GROUP BY
  - One to perform the final sort

- FYI – there also is the EXPLAIN EXTENDED keyword which gives even more detail

- Details are beyond the scope of this presentation

Relational Operators

- Passed operands compared, generates a TRUE or FALSE value

Operator	Operand types	Description
A = B	all primitive types	TRUE if expression A is equivalent to expression B otherwise FALSE
A != B	all primitive types	TRUE if expression A is *not* equivalent to expression B otherwise FALSE
A < B	all primitive types	TRUE if expression A is less than expression B otherwise FALSE
A <= B	all primitive types	TRUE if expression A is less than or equal to expression B otherwise FALSE
A > B	all primitive types	TRUE if expression A is greater than expression B otherwise FALSE
A >= B	all primitive types	TRUE if expression A is greater than or equal to expression B otherwise FALSE

More Relational Operators...

Operator	Operand types	Description
A IS NULL	all types	TRUE if expression A evaluates to NULL otherwise FALSE
A IS NOT NULL	all types	FALSE if expression A evaluates to NULL otherwise TRUE
A LIKE B	strings	TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. Comparison done character by character.
A RLIKE B	strings	NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B (otherwise FALSE).
A REGEXP B	strings	Same as RLIKE

Relational Operator example

LIKE / RLIKE

- Very useful when searching STRING fields

- **Find all users with first name beginning with "Je"**

```
hive> SELECT id, name FROM users WHERE name LIKE 'Je%';

     19 Jessica
     24 Job
     56 Jenn
     ...
```

- RLIKE similar – allows search using regular expressions

Arithmetic Operators

Operator	Operand types	Description
A + B	all number types	A and B added together.
A – B	all number types	B subtracted from A,
A * B	all number types	A multiplied by B.
A / B	all number types	A divided by B.
A % B	all number types	Remainder of A divided by B.
A & B	all number types	Bitwise AND of A and B.
A | B	all number types	Bitwise OR of A and B.
A ^ B	all number types	Bitwise XOR of A and B.
~A	all number types	Bitwise NOT of A.

Logical Operators

Operator	Operand types	Description
A AND B A && B	boolean	If A and B are both TRUE, then this returns TRUE – else returns FALSE.
A OR B A || B	boolean	If either A or B or both are TRUE, then this returns TRUE – else returns FALSE.
NOT A !A	boolean	If A is FALSE, then this returns TRUE – else returns FALSE .

Operators on Complex Types

Operator	Operand types	Description
A[n]	A is an Array and n is an int	Returns the nth element in the array A. The first element has index 0.
M[key]	M is a Map<Key, Value> and key has type K	Returns the value corresponding to the key in the map.
S.x	S is a struct	Returns the x field of struct S.

Built-in Functions

- Wide variety of functions built into Hive

Return Type	Function name (Signature)	Returns….
BIGINT	round(double a)	rounded BIGINT value of the double
BIGINT	floor(double a)	maximum BIGINT value that is equal or less than the double
BIGINT	ceil(double a)	minimum BIGINT value that is equal or greater than the double
STRING	substr(string A, Int start)	substring of A starting from start position till the end of string A
STRING	upper(string A)	string resulting from converting all characters of A to upper case
INT	Length(string s)	length of the string
INT	year(string timestamp)	year part of a timestamp string
STRING	get_json_object(string json_string, string path)	Extracts JSON object from a JSON string based on JSON path specified, and return JSON string of the extracted JSON object.

More Built-in Functions…

Return Type	Function name (Signature)	Description
BIGINT	count(*)	Returns the total number of retrieved rows, including rows containing NULL values.
DOUBLE	sum(col)	Returns the sum of the elements in the group.
DOUBLE	avg(col)	Returns the average of the elements in the group.
DOUBLE	min(col) or max(col)	Returns the minimum or maximum value of the column in the group
N rows	explode(array)	Return zero or more rows – one row for each element from the input array.
N rows	explode(map)	Return zero or more rows – one row for each map key-value pair, with field for each map key and field for the map value

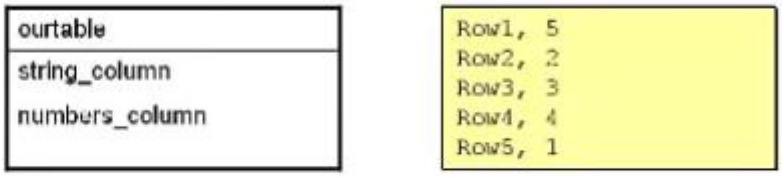
- Run the "SHOW FUNCTIONS" and "DESCRIBE FUNCTION" commands to see more!

Windowing and Analytic Functions

Hive includes Windowing and Analytic Functions

- **Windowing functions**
  - LEAD
  - LAG
  - FIRST_VALUE
  - LAST_VALUE
- **The OVER clause**
  - OVER with standard aggregates:
    - COUNT
    - SUM
    - MIN
    - MAX
    - AVG
  - PARTITION BY, ORDER BY and more
- **Analytics functions**
  - RANK
  - ROW_NUMBER
  - DENSE_RANK
  - CUME_DIST
  - PERCENT_RANK
  - NTILE

Built-in Functions in Action




```
hive> SELECT sum(numbers_column) FROM ourtable;
        15
```

- As expected, Hive outputs the sum of the numbers_column

```
hive> SELECT count(*) FROM ourtable;
        5
```

- Hive outputs the count of the rows in the table

Extending Hive's Functionality

- **Custom User Defined Functions**
  - Create custom functions – implement your own logic
  - Implemented in Java
  - Add to Hive session and use like a built-in function
  - UDF, UDAF, UDTF

- **Streaming – Custom Map/Reduce Scripts**
  - Alternative way of transforming data
  - I/O pipe to external process
  - Data passed to process, operates, writes out
  - MAP(), REDUCE(), TRANSFORM() clauses provided

```
hive> SELECT TRANSFORM (foo, bar)
      USING '/bin/cat' AS newFoo, newBar
      FROM mydatabase.mytable;
```