Hive:
Hive Programing

Week 05-06

Marilson Campos

UCSC Extension - 2022

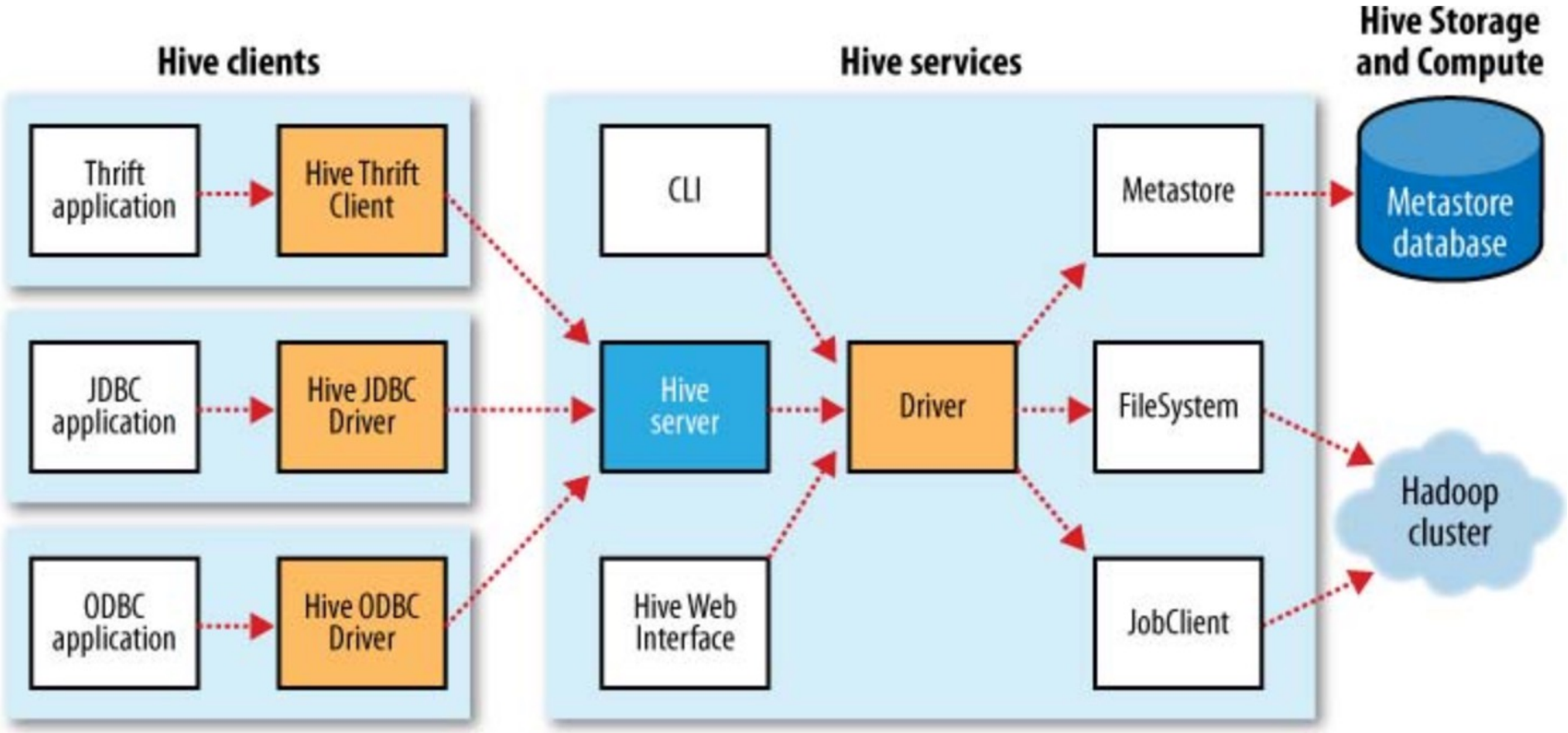
Hive
Hive Origins

Hive has been developed at Facebook.

It provides an environment using SQL like language.

Allows to manipulate data stored in Hadoop Clusters.

Hive
Hive Architecture



Hive
The Hive Environment

1. Command line shell

Can be installed in an external computer (“gateway”)
Requires the most security access to use Hive.

This is the most common way to connect when developing data applications.

Typical user: Data Engineer

Hive
The Hive Environment

2. JDBC/ODBC Database connection

Allow databases IDEs to connect to it and also allow external applications limited access.
Typical user: Data Engineer, Analyst, External app.

3. Thrift API

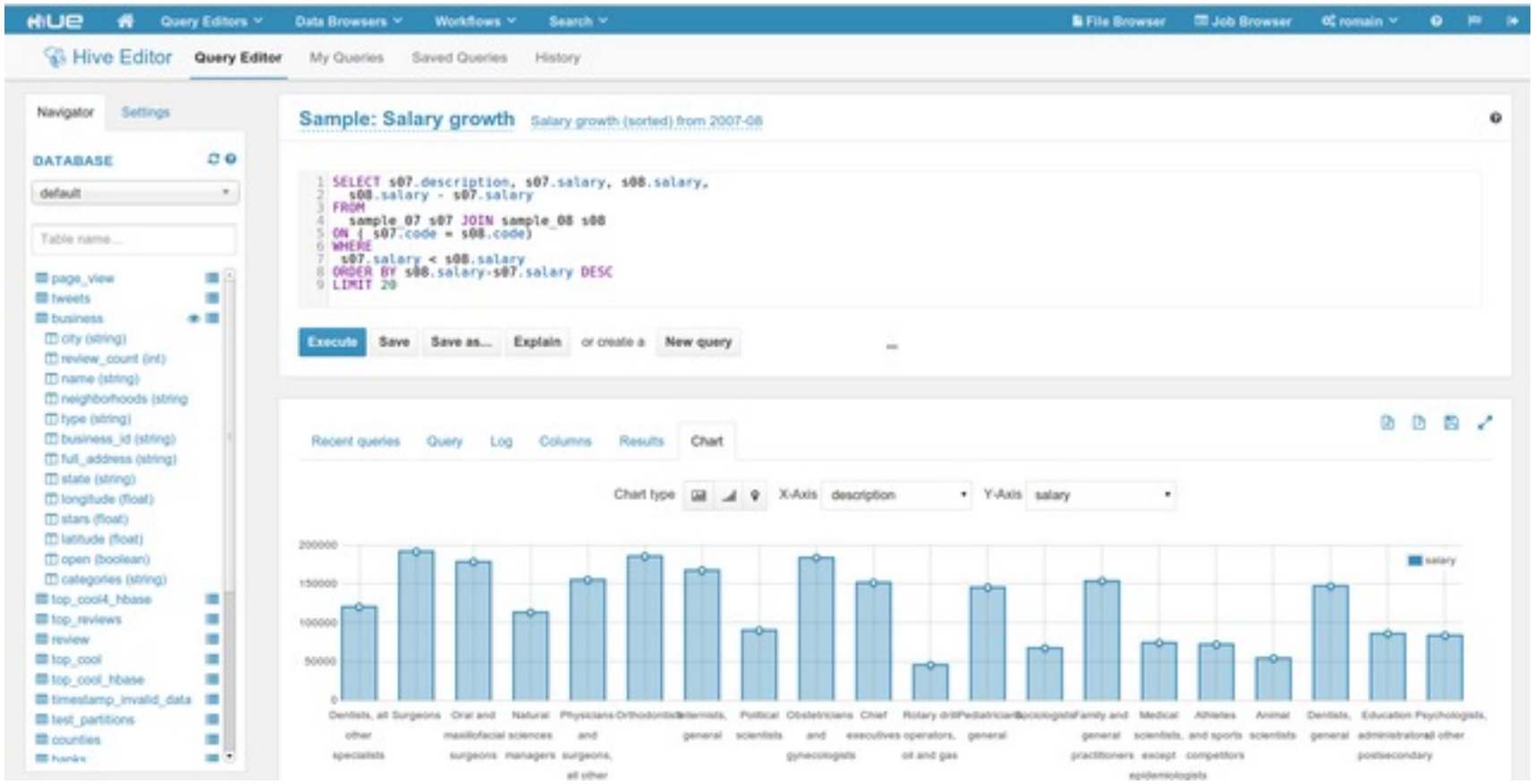
Allows applications to connect to a secure server to issue commands into the cluster.
Typical user: External app.

Hive
The Hive Environment

4. Web Interface

Cloudera offers HUE (http://gethue.com/)

Typical user: Analysts, Executives, Product Managers.



Hive
Java vs Hive

```
SELECT * FROM users
    JOIN user_types
        ON (users.user_type = user_types.id)
```

The java equivalent = 100+ lines

Also, Hive has ways to optimize code based on the data statistics.

Hive
Hive vs Database

Relational Databases

- Allow updates of single records
- Typical response time in milliseconds
- Transaction Support
  (updates in multiple tables)
- Typical large table = Terabytes

HIVE

- Allow updates of files (not single records)
- Typical response time in minutes
- No Transaction Support
- Typical large hive table = 100’s Terabytes

Hive
Hive References

Web

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

https://hive.apache.org/

Books

Programming Hive: Data Warehouse and Query Language for Hadoop 1st Edition
by Edward Capriolo (Author), Dean Wampler (Author), Jason Rutherglen (Author)

Hive
Primitive numeric types

Type	Size	Approximate Range
TINYINT	1 byte	-128 to 127
SMALLINT	2 bytes	-32k to 32k
INT	4 bytes	-2B to 2B
BIGINT	8 bytes	$-9x10^{18}$ to $9x10^{18}$
FLOAT	4 bytes	single precision floating point number
DOUBLE	8 bytes	double precision floating point number
DECIMAL/NUMERIC	Up to 38 digits	

Hive
Primitive Date/Time types

- TIMESTAMP

Represent time formatted using “YYYY-MM-DD HH:MM:SS.fffffffff”.

- DATE

Support values formatted like ‘2018-01-01’ and have range from ‘0000-01-01’ to ‘9999-12-31’.

- INTERVAL

Used to represent intervals. Values can be combined like below: INTERVAL '1' DAY+ INTERVAL '2' HOUR + INTERVAL '3' MINUTE + INTERVAL '4' SECOND + INTERVAL '5' NANO

Hive
Hive string types

- STRING

String can represent strings with a maximum size of 2 Gigabytes. No space is wasted.

- VARCHAR(n)

Varchar types are created with a length specifier (between 1 and 65535). Values larger than 64K are truncated.

- CHAR(n)

Char types are similar to Varchar but they are fixed-length meaning that values shorter than the specified length value are padded to the n size.

Hive
Hive other types

- BOOLEAN

Can represent a true/false value.

- BINARY

Used to represent a sequence of bytes in field. Ex: encrypted data, etc.

Hive
Hive complex types

- ARRAY<data_type>

Sequence of elements of a particular type.

- MAP<primitive_type, data_type>

A dictionary where the key is a primitive type and can hold a value.

- STRUCT<col_name : data_type, ...>

Represent record like entities.

- UNIONTYPE<data_type, data_type, ...>

Represent one of the types listed.

Ex: UNIONTYPE<INT, DOUBLE>

Hive
Hive Tables

Hive tables are represented using "metadata" about their schema.

Hive combines the table schema with the HDFS data and creates the "illusion" that they are the same.

Hive uses the concept of "Schema on Read". It means that the schema representation is applied as the data is read. No additional enforcement is done.

Hive
Hive Tables

MANAGED HIVE TABLE: It's the default type of table in Hive. This type of table "owns" the HDFS data in it. Deleting an internal table causes the HDFS data to be deleted.

EXTERNAL HIVE TABLE: Just point the metadata representation to HDFS locations. Table does not control the data. Deleting an external table does not change anything on HDFS.

Hive SQL
Data Definition Language

- CREATE TABLE

Creates a table in Hive database.

- ALTER TABLE

Makes modifications on a existing table.

- DROP TABLE

Removes a table from the Hive database.

Hive
CREATE TABLE - Simple

```
CREATE TABLE user (
  user_id         BIGINT,
  name            STRING,
  email           STRING,
  user_type       INT
);
```

Hive
CREATE TABLE - Custom Storage

```
CREATE TABLE user_visit(
    user_id BIGINT,
    date_visited STRING,
    server_time BIGINT,
    cookies STRING COMMENT ’Cookies for the user'
 )
 COMMENT 'This is the user_visits table'
 PARTITIONED BY(date_visited STRING)
 CLUSTERED BY(user_id) SORTED BY(server_time)
   INTO 48 BUCKETS
 ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001'
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
 STORED AS SEQUENCEFILE;
```

Hive
CREATE TABLE – Complex Types

```
CREATE TABLE user (
  name        STRING,
  id          BIGINT,
  fulltime    BOOLEAN,
  role        VARCHAR(64),
  salary      DECIMAL(8,2),
  phones      ARRAY<INT>,
  deductions MAP<CHAR, FLOAT>,
  address     STRUCT<street:STRING, city:STRING,
                     state:STRING, zip:INT>,
  others      UNIONTYPE<FLOAT,BOOLEAN,STRING>,
  password    BINARY)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\001'
  COLLECTION ITEMS TERMINATED BY '\002'
  MAP KEYS TERMINATED BY '\003'
  LINES TERMINATED BY '\n';
```

Hive
CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE user (
  name        STRING,
  id          BIGINT,
  fulltime    BOOLEAN,
  role        STRING,
  salary      INT
)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',’
LOCATION 'data/logs/users';
```

Hive
COPY TABLE SCHEMA

```
CREATE TABLE user_tb_copy
    LIKE user;
```

Creates an empty table using the schema from another table.

Hive
CREATE TABLE – From Select

```
CREATE TABLE jfk_1_month AS
  SELECT year, month, flight_num, origin, dest
  FROM flight_info
  WHERE
   (Month = 1 AND Year = 2017) AND
   (Origin='JFK');
```

Selects the data from the “flight_info” table and insert the selected data into the the new table.

Hive
ALTER TABLE

```
ALTER TABLE my_table
    CHANGE aeg age INT;
```

There are many other operations that can be done using alter table.

Most of them are related with ‘PARTITIONS’ that will be covered later in this course.

Hive
DROP TABLE

```
DROP TABLE IF EXISTS my_old_table;
```

Drop table operation depends on the type of table used in the command.

For external hive tables, the drop table command just remove the metadata and does not touch HDFS.

For managed Hive tables, the drop table command removes the table metadata and also deletes the HDFS data used by the table.

Hive
DML - Data Manipulation Language

Allows Hive users to:

- **Loading data** files into into Hive tables
- **Inserting Data from queries** using Hive into other Hive tables.
- **Exporting data** files from the cluster using Hive Queries.

Hive
The SELECT Statement

```
SELECT [DISTINCT]
  field_name,
  expression as name, ...
FROM table
[WHERE where_condition]
[GROUP BY field_list]
[ORDER BY field_list]
[LIMIT number];
```

Hive
The SELECT Statement

```
SELECT
  user_id,
  name,
  (yearly_salary/12) as salary
FROM employees
WHERE state = ‘CA’
--GROUP BY fld This line is a comment!
ORDER BY user_id
LIMIT 100;
```

Hive
The SELECT Statement

```
SELECT
  user_id,
  name,
  (yearly_salary/12) as salary
FROM employees
WHERE state = ‘CA’
-- GROUP BY fld This line is a comment!
ORDER BY user_id
LIMIT 100;
```

Hive
The JOIN Statement

```
SELECT a.*, b.salary FROM a
  JOIN b
    ON (a.id = b.id);
```

-The Boolean operation after the “ON” Statement CANNOT be based on inequality.

ex: **ON (a.id <> b.id)**
THIS IS NOT ALLOWED!

Hive
INNER JOIN = JOIN

```
SELECT a.id, a.name, b.salary FROM a
  INNER JOIN b
    ON (a.id = b.id);

SELECT a.id, a.name, b.salary FROM a
  JOIN b
    ON (a.id = b.id);

-Returns all records from “a” that
matched records from “b”
```

Hive
LEFT OUTER JOIN

```
SELECT a.id, a.name, b.salary FROM a
  LEFT OUTER JOIN b
    ON (a.id = b.id);
```

-Returns all records from “a” that matched records from “b”

- Returns the others records with no match with the b part empty.

Hive
RIGHT OUTER JOIN

```
SELECT * FROM a
  RIGHT OUTER JOIN b
    ON (a.id = b.id);
```

- Returns all records from “b” matched with records from “a”

- Returns the others records with no match with the “a” part empty.

Hive
FULL OUTER JOIN

```
SELECT * FROM inv
  FULL OUTER JOIN tb_acc
    ON (inv.acc_id = tb_acc.id);
```

-Returns all records from “b” matched with records from “a”

-Returns records from “b” with no match in “a” with the “a” part empty.

-Returns records from “a” with no match in “b” with the “b” part empty.

Hive
LEFT SEMI JOIN

Sub queries like

```
SELECT a.key, a.value
 FROM a
 WHERE
  a.key in (SELECT b.key FROM B);
```

ARE NOT SUPPORTED!

Hive
LEFT SEMI JOIN

The query can be rewritten as:

```
SELECT a.key, a.val
FROM a
  LEFT SEMI JOIN b
    ON (a.key = b.key)
```

Hive
MAP JOIN Strategy

If we are joining a large table with one or more smaller tables:

We can process each part of the large table in parallel if we can load the small table(s) in memory.

Hive
MAP JOIN Strategy

```
SELECT /*+ MAPJOIN(b) */
  a.key, a.value
FROM a
  JOIN b
    ON a.key = b.key;
```

The /*+ MAPJOIN(b) */ acts as a hint to HIVE to load table

Hive
LATERAL VIEW

Lets imagine that we have a table called user_friends like:

user_name	friend_id_list
”John"	[11, 12, 31]
”Jane”	[23, 24, 55]

Hive
LATERAL VIEW

```
SELECT
  user_name,
  friend_id
FROM user_friends
LATERAL VIEW
  explode(friend_id_list) AS friend_id;
```

Hive
LATERAL VIEW

Will produce the following results:

user_name	friend_id
”john"	11
"john"	12
"john"	31
"Jane"	23
"Jane"	24
"Jane"	55

Hive
UNION

```
SELECT q1.user_id. q1.email, q1.desc
    FROM (
        SELECT user_id, email,
          "standard member" as desc
          FROM  users
          WHERE user_type < 3
        UNION ALL
        SELECT user_id, email,
          "gold member" as desc
          FROM  users
          WHERE user_type = 3
     ) q1;
```

Hive
DESCRIBE <table>

```
hive> describe USERS;
Ok
user_id                 string
name                    string
…
DT_DATE                 string
```

See Also: DESCRIBE EXTENDED <table>;

Hive
SHOW DATABASES

```
hive> SHOW DATABASES;
 Ok
 default
```

List the databases on Hive. The standard installation creates a Database called ‘default’ with no tables.

Hive
SHOW TABLES

```
hive> SHOW TABLES;
Ok
nasa_raw
ratings
```

Lists the tables defined in the current database.

Hive
SHOW CREATE TABLE <table>

```
hive> SHOW CREATE TABLE USERS;
Ok
CREATE EXTERNAL TABLE ‘USERS’ (
user_id                 string
name                    string
…)
ROW FORMAT SERDE
…
TextOutputFormat'LOCATION
'hdfs://quickstart.cloudera:8020/data/data_nasa’
TBLPROPERTIES ('COLUMN_STATS_ACCURATE'='false',
'numFiles'='0',   'numRows'=‘-1’…)
```

Hive
BUCKETING

It’s a technique that is used to separate data results into separate groups (buckets)

Bucketing separates data based on a particular data field.

This field is input for a function that returns a number between 1 and Number of buckets.

Hive
BUCKETING

Buckets are the atomic part of a table during reads in a source folder

If no bucketing is defined ALL DATA is read from source folder

Hive
BUCKETING

Let’s say we bucket on user_id number

Bucketing allows programs to deterministically access subsets of the original data.

Use case example:

We have 300 million of users but want to build models using slices of 6 million users without reading all 300 million users.

Hive
BUCKETING

```
CREATE TABLE user (
  user_id  BIGINT COMMENT ‘user_id fld’,
  cookies  STRING COMMENT ‘user cookies’,
  email    STRING,
  user_type INT COMMENT ‘1=STD, 2=SVL, 3=GLD”
)
CLUSTERED BY (user_id) INTO 32 BUCKETS;
```

Hive
SELECTING A BUCKET

```
SELECT *
FROM users
TABLESAMPLE(BUCKET 4 OUT OF 32
            ON user_id) q1;
```

Hive
BUCKETING vs SAMPLING

- Bucketing

Guarantees that data with the same id will go into the same bucket.

This means that if we have id number 1 stored in bucket 7 all records with id 1 will be available.

- Sampling

Generates a random number and based on that defines if we are including this record or not.

New set of records every-time we run the program

Hive
INSERT OVERWRITE

```
INSERT OVERWRITE TABLE page_view_us
  SELECT view_time, user_id, page_url
  FROM page_view
  WHERE country = 'US';
```

Inserts data from a select into a new table.

Hive
HIVE FUNCTIONS

Hive provides a library of many functions that you can use from the standard install.

We are going to be covering many functions here. For more details “google hive functions”.

Also, Hive allows developers to create they own UDF’s (User defined functions) in java.

We are covering UDF’s later in this class.

Hive
HIVE FUNCTIONS

ROUND

DOUBLE round(DOUBLE a)
Returns the rounded BIGINT value of a.

DOUBLE round(DOUBLE a, INT d)
Returns a rounded to d decimal places.

Hive
HIVE FUNCTIONS

FLOOR & CEIL

BIGINT floor(DOUBLE a)
Returns the maximum BIGINT value that is equal tc or less than a.

BIGINT ceil(DOUBLE a), ceiling(DOUBLE a)
Returns the minimum BIGINT value that is equal tc or greater than a.

Hive
HIVE FUNCTIONS

RAND

DOUBLE rand(), rand(INT seed)

Returns a random number (that changes from row tc row) that is distributed uniformly from 0 to 1.

Specifying the seed will make sure the generated random number sequence is deterministic.

Hive
HIVE FUNCTIONS

EXP & LN

DOUBLE exp(DOUBLE a), exp(DECIMAL a)
Returns e^a where e is the base of the natural logarithm.

DOUBLE ln(DOUBLE a), ln(DECIMAL a)

Returns the natural logarithm of the argument a.

Hive
HIVE FUNCTIONS

LOG10 & LOG2

DOUBLE log10(DOUBLE a), log10(DECIMAL a)
Returns the base-10 logarithm of the argument a.

DOUBLE log2(DOUBLE a), log2(DECIMAL a)
Returns the base-2 logarithm of the argument a.

Hive
HIVE FUNCTIONS

LOG, POW and SQRT

DOUBLE log(DOUBLE b, DOUBLE a)
DOUBLE log(DECIMAL b, DECIMAL a)
Returns the b-base logarithm of the argument a.

DOUBLE pow(DOUBLE a, DOUBLE p),
DOUBLE power(DOUBLE a, DOUBLE p)
Returns a^p.

DOUBLE sqrt(DOUBLE a),
DOUBLE sqrt(DECIMAL a)
Returns the square root of a.

Hive
HIVE FUNCTIONS

BIN and HEX

STRING bin(BIGINT a)
Returns the number in binary format.

STRING hex(BIGINT a) hex(STRING a) hex(BINARY a)
If the argument is an INT or binary, hex returns the number as a STRING in hexadecimal format. Otherwise if the number is a STRING, it converts each character into its hexadecimal representation and returns the resulting STRING.

Hive
HIVE FUNCTIONS

UNHEX

BINARY unhex(STRING a)
Inverse of hex. Interprets each pair of characters as a hexadecimal number and converts to the byte representation of the number.

Hive
HIVE FUNCTIONS

CONV and ABS

STRING conv(BIGINT num, INT from_base, INT to_base),
STRING conv(STRING num, INT from_base, INT to_base)
Converts a number from a given base to another

DOUBLE abs(DOUBLE a)
Returns the absolute value.

INT pmod(INT a, INT b),
DOUBLE pmod(DOUBLE a, DOUBLE b)
Returns the positive value of a mod b.

Hive
HIVE FUNCTIONS

SIN and ASIN

DOUBLE sin(DOUBLE a), sin(DECIMAL a)
Returns the sine of a (a is in radians).

DOUBLE asin(DOUBLE a), asin(DECIMAL a)
Returns the arc sin of a if -1<=a<=1 or NULL otherwise.

Hive
HIVE FUNCTIONS

TAN, ATAN and DEGREES

DOUBLE tan(DOUBLE a), tan(DECIMAL a)
Returns the tangent of a (a is in radians).

DOUBLE atan(DOUBLE a), atan(DECIMAL a)
Returns the arctangent of a.

DOUBLE degrees(DOUBLE a), degrees(DECIMAL a)
Converts value of a from radians to degrees.

Hive
HIVE FUNCTIONS

RADIANS, POSITIVE and NEGATIVE

DOUBLE radians(DOUBLE a), radians(DOUBLE a)
Converts value of a from degrees to radians.

INT or DOUBLE positive(INT a), positive(DOUBLE a)
Returns a.

INT or DOUBLE negative(INT a), negative(DOUBLE a)
Returns -a.

Hive
HIVE FUNCTIONS

SIGN, E and PI

DOUBLE or INT sign(DOUBLE a), sign(DECIMAL a)
Returns the sign of a as '1.0' (if a is positive) or '-1.0' (if a is negative), '0.0' otherwise.

DOUBLE e()
Returns the value of e.

DOUBLE pi()
Returns the value of pi.

Hive
HIVE FUNCTIONS

FACTORIAL and CBRT

BIGINT factorial(INT a)
Returns the factorial of a Valid a is [0..20].

DOUBLE cbrt(DOUBLE a)
Returns the cube root of a double value.

Hive
HIVE FUNCTIONS

shiftleft and shiftright

shiftleft(BIGINT a, INT b)
Bitwise left shift. Shifts a b positions to the left.
Returns int for tinyint, smallint and int a. Returns bigint for bigint a.

shiftright(BIGINT a, INT b)
Bitwise right shift. Shifts a b positions to the right.Returns int for tinyint, smallint and int a. Returns bigint for bigint a.

Hive
HIVE FUNCTIONS

Shiftrightunsigned

Shiftrightunsigned(BIGINT a, INT b)
Bitwise unsigned right shift. Shifts a b positions to the right.

Returns int for tinyint, smallint and int a. Returns bigint for bigint a.

Hive
HIVE FUNCTIONS

GREASTEST and LEAST

T greatest(T v1, T v2, ...)
Returns the greatest value of the list of values. Fixed to return NULL when one or more arguments are NULL, and strict type restriction relaxed, consistent with ">" operator.

T least(T v1, T v2, ...)
Returns the least value of the list of values. Fixed to return NULL when one or more arguments are NULL, and strict type restriction relaxed, consistent with "<" operator.

Hive
HIVE FUNCTIONS

WITH_BUCKET

INT with_bucket(NUMERIC expr, NUMERIC min_val,
NUMERIC max_val, INT num_buckets)

Returns an integer between 0 and num_buckets+1 by mapping expr into the ith equally sized bucket.

Buckets are made by dividing [min_value, max_value] into equally sized regions.

If expr < min_value, return 1, if expr > max_value return num_buckets+1.

Hive
HIVE FUNCTIONS

REGEXP_EXTRACT

String regexp_extract(string subject,
string pattern,
int index)

Returns the string extracted using the pattern.
For example,
regexp_extract('foothebar', 'foo(.*?)(bar)', 1)
returns 'the.'

Note that some care is necessary in using predefined character classes: using '\s' as the second argument will match the letter s; '\\s' is necessary to match whitespace, etc. The 'index' parameter is the Java regex Matcher group() method index.

Hive
HIVE FUNCTIONS

ARRAY_CONTAINS

array_contains(Array<T>, value)

Returns TRUE if the array contains value.

Hive
Table Partitions

Table partition allows us to create a set of folders as a source for table data.

The most common use is to partition by date.

During SELECTS we can define what ‘dates’ we care and HIVE ignores the folders that do not match our selection.

Hive
Table Partitions

Hive allows multiple levels of partitions:

Examples:

a. partition by date, hour
b. partition by date, hour, country
c. partition by product, date
d. partition by country, date, zip_code

Hive
Table without partitions

```
CREATE TABLE nasa_daily_v1 (
  address  STRING,
  html_page  STRING COMMENT “the page”,
  http_code    STRING,
  size INT,
  dt_data STRING
);
```

Hive
Table with partitions

```
CREATE TABLE nasa_daily_v2 (
  address  STRING,
  html_page  STRING COMMENT “the page”,
  http_code    STRING,
  size INT
)
PARTITIONED BY(dt_date STRING);
```

Notice that the date is not inside of table def.

Hive
Load data from temp table – no partitions

```
INSERT OVERWRITE TABLE nasa_daily_v1
SELECT
   address,
   html_page,
   http_error_code,
  size,
  “1995-07-01” as dt_date
FROM nasa_temp;
```

Hive

Load data from temp table to partitioned table.

```
INSERT OVERWRITE TABLE nasa_daily_v2
PARTITION(dt_date="1995-07-01")
SELECT
   address, html_page, http_code, size
FROM nasa_temp;
```

Notice that the date field disappear from the select.

Dynamic partitions allows you insert data in multiple partitions in a single step.

Hive
Dynamic Partitions

```
INSERT OVERWRITE TABLE
nasa_daily_v2
PARTITION(the_date)
SELECT
   ip_address,
   html_page,
   http_code,
   size,
   the_date
FROM nasa_temp_with_dt;
```

Hive
Look at table partitions

SHOW PARTITIONS nasa_daily_v2;

Step 1. To see the partitions of a table use:

SHOW PARTITIONS nasa_daily_v2;

Step 2. Lets look at the v2 table in hdfs

hdfs dfs -ls /data/data_nasa

Where did the table go?

Review
Unix Here files

```
#!/bin/bash
echo "INFO: This is my script”
RUN_DATE=20130611
hive << EOF
  SELECT ip, url
  FROM links
  WHERE data_date = ${RUN_DATE}
;
EOF
```

Review
Unix Here with bash parameters

```
#!/bin/bash
echo "INFO: This is my first script”
RUN_DATE=$1
hive << EOF
  SELECT ip, url
  FROM links
  WHERE data_date = ${RUN_DATE}
;
EOF
```

Now we call the script passing the date we want to process

Review
Unix Here with saving to file.

```
#!/bin/bash
echo "INFO: This is my first script”
RUN_DATE=$1
hive << EOF > /tmp/my_data.txt
  SELECT ip, url
  FROM links
  WHERE data_date = ${RUN_DATE}
;
EOF
```

Now we call the script passing the date we want to process