Hive in Depth

CISC-525

Phil Grim

HARRISBURG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Overview

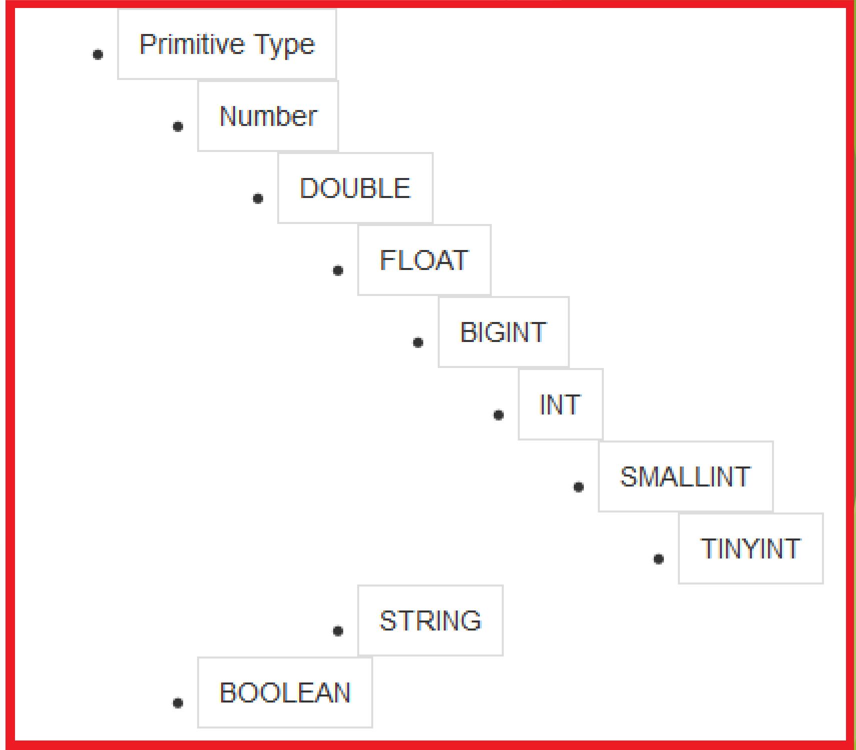
- Data Types
- Data Definition Language
- Data Manipulation Language
  - Loading Data
  - Querying Data
- User-Defined Functions
- MapReduce and HCatalog

Data Types in Hive

- Primitive and complex types
- Types associated with columns in tables
- Organized in a hierarchy

Primitive Data Types in Hive

- Integers
  - TINYINT
  - SMALLINT
  - INT
  - BIGINT
- Real Numbers
  - FLOAT
  - DOUBLE
  - DECIMAL



Primitive Data Types in Hive

- Date/Time
    - TIMESTAMP
    - DATE
- String
    - STRING
    - VARCHAR
- Other
    - BOOLEAN
    - BINARY

Complex Data Types in Hive

- ARRAY
- MAP
- STRUCT
- UNIONTYPE

Creating a Database

```
CREATE DATABASE [IF NOT EXISTS] database_name
  [COMMENT database_comment]
  [LOCATION hdfs_path]
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

- Removing:

```
DROP DATABASE [IF EXISTS] database_name [RESTRICT|CASCADE];
```

- Altering:

```
ALTER DATABASE database_name SET DBPROPERTIES (property_name=property_value, ...);
```

Creating a Table

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]
```

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  LIKE existing_table_or_view_name
  [LOCATION hdfs_path]
```

Creating a Table - Example

```
hive> CREATE TABLE counting (
    >   rownum INT COMMENT 'row number',
    >   cardinal INT COMMENT 'Arabic cardinal number',
    >   english_cardinal STRING COMMENT 'English language cardinal word',
    >   native_cardinal STRING COMMENT 'Native language cardinal word',
    >   english_ordinal STRING COMMENT 'English language ordinal word',
    >   native_ordinal STRING COMMENT 'Native language ordinal word')
    > PARTITIONED BY (language STRING COMMENT 'Native language')
    > ROW FORMAT DELIMITED
    >   FIELDS TERMINATED BY ','
    >   LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE;
OK
Time taken: 0.211 seconds
hive>
```

Removing a Table

```
DROP TABLE [IF EXISTS] table_name
```

- Remove table from database

```
TRUNCATE TABLE table_name [PARTITION partition_spec];
```

Altering a Table

- Rename a Table

```
ALTER TABLE table_name RENAME TO new_table_name
```

- Alter Table Properties

```
ALTER TABLE table_name SET TBLPROPERTIES table_properties
```

- Alter Columns in a Table

```
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name data_type [COMMENT
col_comment], ...)
```

```
ALTER TABLE table_name CHANGE [COLUMN] col_old_name col_new_name column_type
[COMMENT col_comment] [FIRST|AFTER column_name]
```

Creating an Index

```
CREATE INDEX index_name
ON TABLE base_table_name (col_name, ...)
AS index_type
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
```

- Alter Index

```
ALTER INDEX index ON table [PARTITION partition] REBUILD
```

- Drop Index

```
DROP INDEX [IF EXISTS] index_name ON table_name
```

Creating a View

- Create a View

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...) ]
[COMMENT view_comment]
[TBLPROPERTIES (property_name = property_value, ...)]
AS SELECT ..
```

- Alter A View

```
ALTER VIEW view_name AS SELECT…
```

- Delete A View

```
DROP VIEW [IF EXISTS] view_name
```

Internal vs. External Tables

- Internal Table
  - Stored in Hive’s warehouse directory
  - Physically deleted by DROP TABLE command
- External Table
  - Stored in HDFS in arbitrary location
  - Not physically deleted by DROP TABLE command
  - Generally used for importing and exporting data

Creating an External Table - Example

```
hive> CREATE EXTERNAL TABLE counting_ext (
    >   rownum INT COMMENT 'row number',
    >   cardinal INT COMMENT 'Arabic cardinal number',
    >   english_cardinal STRING COMMENT 'English language cardinal word',
    >   native_cardinal STRING COMMENT 'Native language cardinal word',
    >   english_ordinal STRING COMMENT 'English language ordinal word',
    >   native_ordinal STRING COMMENT 'Native language ordinal word',
    >   language STRING)
    > ROW FORMAT DELIMITED
    >   FIELDS TERMINATED BY ','
    >   LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE
    > LOCATION '/user/mapr/counting-ext';
OK
Time taken: 0.11 seconds
hive>
```

Partitioning

- Splits table into separate files using one or more columns as partition key.
- Improves query performance by limiting the number of rows that must be scanned.
- Can be defined at table creation time, or added via ALTER commands.
- Partitions stored as subdirectories of table directory.

Partitioning

- Creating a Partitioned Table

```
CREATE TABLE sequence_part_test (
  id int,
  str string
  grk string
  ord string)
PARTITIONED BY (dt STRING, country STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
STORED AS SEQUENCEFILE
```

Partitioning – Altering Partitions

- Add A Partition

```
ALTER TABLE sequence_part_test ADD PARTITION (dt='2014-05-29', country='USA')
  LOCATION '/warehouse/path/table/USA/part20140529'
```

- Rename A Partition

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION
partition_spec;
```

- Remove A Partition

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION
partition_spec,...
```

Clustering

- Group and sort by one or more columns

```
CREATE TABLE sequence_cluster_test (
  id int,
  str string
  grk string
  ord string)
PARTITIONED BY (dt STRING, country STRING)
CLUSTERED BY (str) SORTED BY (id) INTO 10 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
STORED AS SEQUENCEFILE
```

Serialization/Deserialization (SerDe)

- Hive stores data using serialization and deserialization
- Native file formats provide classes that perform these operations
- Custom formats can be provided to support non-native formats

```
CREATE EXTERNAL TABLE mapr_table_1(
  key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")
TBLPROPERTIES("hbase.table.name" = "/user/mapr/my_mapr_table");
```

Loading Data - Methods

- LOAD DATA
- INSERT [OVERWRITE | INTO]
- CREATE TABLE AS SELECT
- External Table
- MapReduce

LOAD DATA

- Simplest way to load data
- No transformation of data as it is loaded
- Table must exist

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

LOAD DATA - Example

```
$  hive -e 'LOAD DATA LOCAL foo.csv INTO TABLE counting PARTITION
(language=“French")‘

$  hive -e 'LOAD DATA LOCAL foo2.csv INTO TABLE counting PARTITION
(language="Greek")‘

$  hive -e 'LOAD DATA LOCAL foo3.csv.gz INTO TABLE counting PARTITION
(language=“Spanish")‘
```

INSERT [OVERWRITE | INTO]

- Loads data into a table based on a query
- Can use dynamic partitioning
- Can insert multiple tables in one pass
- Table must exist

```
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2
...) [IF NOT EXISTS]] select_statement1 FROM from_statement;


INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1 FROM from_statement;
```

INSERT INTO - Example

```
INSERT INTO TABLE counting PARTITION (language = 'Korean')
  SELECT rownum, cardinal, english_cardinal, native_cardinal,
         english_ordinal, native_ordinal
   FROM  counting_ext
   WHERE language='Korean';

INSERT INTO TABLE counting PARTITION (language = 'English')
  SELECT rownum, cardinal, english_cardinal, native_cardinal,
         english_ordinal, native_ordinal
   FROM  counting_ext
   WHERE language='English';
```

INSERT [OVERWRITE | INTO]

```
FROM from_statement
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2
...) [IF NOT EXISTS]] select_statement1
[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]]
select_statement2]
[INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2] ...;

FROM from_statement
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1
[INSERT INTO TABLE tablename2 [PARTITION ...] select_statement2]
[INSERT OVERWRITE TABLE tablename2 [PARTITION ... [IF NOT EXISTS]]
select_statement2] ...;
```

INSERT [OVERWRITE | INTO]

- Dynamic Partition Syntax

```
INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2]
...) select_statement FROM from_statement;


INSERT INTO TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...)
select_statement FROM from_statement;
```

CREATE TABLE AS SELECT

- Combines table create and data load in one step
- Does not work for external tables

```
CREATE TABLE new_key_value_store
   ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
   STORED AS RCFile
   AS
SELECT (key % 1024) new_key, concat(key, value) key_value_pair
FROM key_value_store
SORT BY new_key, key_value_pair;
```

External Table

- Attach an existing data file to Hive as a table
- Only creates metadata – user must format the file properly
- File is not deleted when table is dropped
- Usually used to provide a data source for an INSERT or CTAS

MapReduce

- Several ways to use MapReduce to get data into Hive
  - Write files that can be attached as partitions, external tables, or internal tables
  - Use HCatalog output format to insert directly into a Hive table from the job

```
protected void reduce(IntWritable key, Iterable<IntWritable> values, Context context)
  throws IOException, InterruptedException {
  int sum = 0;
  Iterator<IntWritable> iter = values.iterator();
  while (iter.hasNext()) {
    sum++;
    iter.next();
  }
  HCatRecord record = new DefaultHCatRecord(2);
  record.set(0, key.get());
  record.set(1, sum);
  context.write(null, record);
}
```

Simple Query

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[ORDER BY col_name col_order]
[GROUP BY col_list]
[CLUSTER BY col_list
  | [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT number]
```

```
SELECT COUNT(*)
FROM table_reference
[WHERE where_condition]
```

Query Examples

Counting Rows:

```
hive> SELECT count(*) FROM counting;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
Starting Job = job_201406241055_0001, Tracking URL = http://pbj-master:50030/jobdetails.jsp?jobid=job_201406241055_0001
Kill Command = /opt/mapr/hadoop/hadoop-0.20.2/bin/../bin/hadoop job  -kill job_201406241055_0001
Hadoop job information for Stage-1: number of mappers: 4; number of reducers: 1
2014-06-24 10:59:06,582 Stage-1 map = 0%,  reduce = 0%
2014-06-24 10:59:11,688 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 3.2 sec
2014-06-24 10:59:12,696 Stage-1 map = 75%,  reduce = 0%, Cumulative CPU 4.84 sec
2014-06-24 10:59:13,703 Stage-1 map = 75%,  reduce = 0%, Cumulative CPU 4.84 sec
2014-06-24 10:59:17,728 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.01 sec
MapReduce Total cumulative CPU time: 7 seconds 10 msec
Ended Job = job_201406241055_0001
MapReduce Jobs Launched:
Job 0: Map: 4  Reduce: 1   Cumulative CPU: 7.01 sec   MAPRFS Read: 0 MAPRFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 10 msec
OK
40
Time taken: 13.637 seconds, Fetched: 1 row(s)
hive>
```

Query Examples

Simple SELECT:

```
hive> SELECT * FROM counting;
OK
2        1        One      One      First    First    English
4        2        Two      Two      Second   Second   English
6        3        Three    Three    Third    Third    English
8        4        Four     Four     Fourth   Fourth   English
10        5        Five     Five     Fifth    Fifth    English
37       7        Seven    Siete    Seventh Septimo Spanish
38       8        Eight    Ocho     Eighth   Octavo   Spanish
39       9        Nine     Nueve    Ninth    Noveno   Spanish
40       10       Ten      Diez     Tenth    Decimo   Spanish
Time taken: 1.435 seconds, Fetched: 40 row(s)
hive>
```

Query Examples

Specifying some conditions:

```
hive> SELECT native_cardinal FROM counting WHERE language = 'Spanish' AND
cardinal > 5;
OK
Seis
Siete
Ocho
Nueve
Diez
Time taken: 6.226 seconds, Fetched: 5 row(s)
hive>
```

Query Examples

Specifying some more conditions:

```
hive> SELECT cardinal, english_ordinal, native_ordinal, language FROM
counting WHERE cardinal IN (3,7) CLUSTER BY language;
OK
3       Third   Third   English
7       Seventh Seventh English
3       Third   Tritos  Greek
7       Seventh Ebdomos Greek
3       Third   Sam     Korean
7       Seventh Chil    Korean
3       Third   Tercero Spanish
7       Seventh Septimo Spanish
Time taken: 12.8 seconds, Fetched: 8 row(s)
hive>
```

Query Examples

Getting unique results:

```
hive> SELECT DISTINCT language FROM counting ORDER BY language;
OK
English
Greek
Korean
Spanish
Time taken: 9.739 seconds, Fetched: 4 row(s)
hive>
```

Join

- Get results from multiple tables based on equality of columns.
- Equality joins, outer joins, and left semi-joins are supported
- Joins must occur before WHERE clauses
- Explicit JOIN notation required until Hive 0.13

```
join_table:
    table_reference JOIN table_factor [join_condition]    |
    table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN
        table_reference join_condition                    |
    table_reference LEFT SEMI JOIN table_reference join_condition
```

Join Examples

Introducing another table:

```
hive> SELECT a.team, b.english_ordinal FROM rankings a JOIN counting b ON
(a.rank = b.cardinal);
Charlie First
Echo    Second
India   Third
Bravo   Fourth
Hotel   Fifth
Juliet  Sixth
Alpha   Seventh
Foxtrot Eighth
Delta   Ninth
Golf    Tenth
Charlie First
Echo    Second
India   Third
Bravo   Fourth
…
```

Join Examples

Adding a condition:

```
hive> SELECT a.team, b.english_ordinal FROM rankings a JOIN counting b ON
(a.rank = b.cardinal) WHERE b.language = 'English';
Charlie First
Echo    Second
India   Third
Bravo   Fourth
Hotel   Fifth
Juliet  Sixth
Alpha   Seventh
Foxtrot Eighth
Delta   Ninth
Golf    Tenth
Time taken: 6.58 seconds, Fetched: 10 row(s)
hive>
```

Join Examples

Left Outer Join

```
hive> SELECT a.team, b.english_ordinal FROM rankings a LEFT OUTER JOIN
counting b ON (a.rank = b.cardinal) WHERE b.language = 'English‘ OR
b.language IS NULL;
Alpha   Seventh
Bravo   Fourth
Charlie First
Delta   Ninth
Echo    Second
Foxtrot Eighth
Golf    Tenth
Hotel   Fifth
India   Third
Juliet  Sixth
Spinal Tap      NULL
Time taken: 5.412 seconds, Fetched: 11 row(s)
hive>
```

Join Examples

Joining another table

```
SELECT b.cardinal Rank, b.native_ordinal Place, a.team Team, c.name Country
from rankings a JOIN counting b ON (a.rank = b.cardinal) JOIN countries c ON
(a.country_code = c.country_code) WHERE b.language = 'English' ORDER BY
b.cardinal;
1       First   Charlie Republic of Korea
2       Second  Echo    Mexico
3       Third   India   Spain
4       Fourth  Bravo   France
5       Fifth   Hotel   Great Britain
6       Sixth   Juliet  Columbia
7       Seventh Alpha   United States of America
8       Eighth  Foxtrot United States of America
9       Ninth   Delta   Canada
10      Tenth   Golf    Republic of Korea
Time taken: 11.834 seconds, Fetched: 10 row(s)
```

UNION ALL

- Hive supports the UNION ALL construct to combine the results of multiple select statements into one result set

```
select_statement UNION ALL select_statement UNION ALL select_statement

SELECT *
FROM (
  select_statement
  UNION ALL
  select_statement
) unionResult
```

EXPLAIN

- The EXPLAIN command shows the execution plan for a query
- EXPLAIN EXTENDED gives significantly more information
- EXPLAIN DEPENDENCY gives extra information about the inputs to the query.

```
EXPLAIN [EXTENDED|DEPENDENCY] query
```

EXPLAIN Example

```
EXPLAIN SELECT team, rank FROM rankings;
```

```
ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_TABREF (TOK_TABNAME rankings))) (TOK_INSERT
(TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR
(TOK_TABLE_OR_COL team)) (TOK_SELEXPR (TOK_TABLE_OR_COL rank)))))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 is a root stage
```

EXPLAIN Example

…continued

```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        rankings
          TableScan
            alias: rankings
            Select Operator
              expressions:
                    expr: team
                    type: string
                    expr: rank
                    type: int
              outputColumnNames: _col0, _col1
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.TextInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

  Stage: Stage-0
    Fetch Operator
      limit: -1
```

User-Defined Functions

- Provide functionality that can be used in conjunction with queries
- Three Types:
  - User Defined Function (UDF)
  - User Defined Aggregate Function (UDAF)
  - User Defined Table-generating Function (UDTF)
- All Hive functions are UDFs
- Custom UDFs can be created

```
SHOW FUNCTIONS;

DESCRIBE FUNCTION [EXTENDED] <function_name>;
```

User-Defined Functions: Relational

Operator	Operand Types	Description
A = B	Primitive	TRUE if expressions are equal, FALSE otherwise
A < B	Primitive	TRUE if A is less than B, FALSE otherwise
A <> B	Primitive	TRUE if expressions are not equal, FALSE otherwise
A != B	Primitive	Synonym for A <> B
A <= B	Primitive	TRUE if A is less than or equal to B, FALSE otherwise
A IS NULL	All types	TRUE if A has no value, FALSE otherwise
A IS NOT NULL	All types	TRUE if A has a value, FALSE otherwise
A LIKE B	Strings	NULL if A or B is NULL, TRUE if A matches the SQL regular expression B, FALSE otherwise.

User-Defined Functions: Arithmetic

Operator	Operand Types	Description
A + B	All Number	Adds the operands.
A - B	All Number	Subtracts the operands.
A * B	All Number	Multiplies the operands.
A / B	All Number	Divides A by B
A % B	All Number	Gives the remainder resulting from dividing A by B
A & B	All Number	Bitwise AND of A and B
A | B	All Number	Bitwise OR of A and B
A ^ B	All Number	Bitwise XOR of A and B
~A	All Number	Bitwise NOT of A

User-Defined Functions: Logical

Operator	Operand Types	Description
A AND B	Boolean	TRUE if both expressions are TRUE, NULL if either expression is NULL, FALSE otherwise
A && B	Boolean	Synonym for A AND B
A OR B	Boolean	TRUE if either expression is TRUE
A || B	Boolean	Synonym for A OR B
NOT A	Boolean	TRUE if A is FALSE
! A	Boolean	Synonym for NOT A
A IN (list)	Boolean	TRUE if A is equal to any value in the list
EXISTS query	Boolean	TRUE if the query returns at least one row

User-Defined Functions: Complex Type

Function	Operands	Description
map	(key1, val1, key2, val2…)	Creates a map
struct	(val1, val2, val3…)	Creates a structure
named_struct	(name1,val1,name2,val2…)	Creates a structure
array	(val1, val2, val3…)	Creates a array
create_union	(tag, val1, val2, val3…)	Creates a union
A[n]	A is an array, n is an int	Gets the nth value in the array
M[key]	M is a map<K,V>, key is K	Gets the value associated with key
S.x	S is a structure, x is an element name	Gets the value of the named element

User-Defined Functions: Collection

Return Type	Function Signature	Description
int	size(Map<K,V>)	Returns the number of elements in the map
int	size(Array<T>)	Returns the number of elements in the array
array<K>	map_keys(Map<K,V>)	Returns an unsorted array of the keys from the input map
array<V>	map_values(Map<K,V>)	Returns an unsorted array of the values from the input map
boolean	array_contains(Array<T>, value)	TRUE if the input array contains the value, FALSE otherwise
array<T>	sort_array(Array<T>)	Sorts the input array in ascending order by natural ordering of T

User-Defined Functions: Math Functions

Return Type	Function Signature	Description
DOUBLE	rand(INT seed)	Generates random number between 0 and 1
DOUBLE	ln(DOUBLE a)	Returns the natural log of a
DOUBLE	pow(DOUBLE a, DOUBLE p)	Returns a raised to the p power
DOUBLE	sqrt(DOUBLE a)	Returns the square root of a
DOUBLE	abs(DOUBLE a)	Returns the absolute value of a
DOUBLE	sin(DOUBLE a)	Returns the sine of a
DOUBLE	degrees(DOUBLE a)	Converts a radians to degrees
DOUBLE	e()	Returns the value of e
DOUBLE	pi()	Returns the value of pi

User Defined Aggregate Functions

Return Type	Function Signature	Description
BIGINT	count(*), count(expr)	Returns the total number of retrieved rows.
DOUBLE	sum(col)	Returns the sum of retrieved elements
DOUBLE	avg(col)	Returns the average of retrieved elements
DOUBLE	min(col)	Returns the minimum of retrieved elements
DOUBLE	max(col)	Returns the maximum of retrieved elements
DOUBLE	stddev_pop(col)	Returns the standard deviation
DOUBLE	corr(col1, col2)	Returns the coefficient of correlation
DOUBLE	percentile(col,p)	Returns the pth percentile of the column
array	collect_set(col)	Returns an array of distinct values

User-Defined Table Generating Functions

Return Type	Function Signature	Description
T	explode(array<T>)	Returns one row for each array element
K,V	explode(map<K,V>)	Returns one row for each map entry
n, T	pos_explode(array<T>)	Returns the position and array element for each row in the array
tuple	json_tuple(jsonStr, k1, k2, …)	Returns a tuple of values from a given JSON string
tuple	parse_url_tuple(url,p1,p2,p3…)	Extracts parts from a URL
rows	inline(array<STRUCT>)	Explodes an array of structs into a table

Creating Custom UDFs

- User Defined Functions are implemented as Java classes
- Hive API provides generic implementations of UDF, UDAF, and UDTF that can be extended for custom functionality
- Custom JAR files are added to Hive using the “add jar” command
- Custom function created using “CREATE FUNCTION” statement
- Custom function can then be called in any query
- Custom functions can be temporary or persistent as of Hive 0.13

Custom UDF Example

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class Lower extends UDF {
  public Text evaluate(final Text s) {
    if (s == null) { return null; }
    return new Text(s.toString().toLowerCase());
  }
}
```

```
hive> add jar my_jar.jar;
OK
hive> CREATE FUNCTION my_lower AS 'com.example.hive.udf.Lower';
OK
```

MapReduce and HCatalog

- HCatInputFormat allows access to HCatalog-managed tables.
- Input can be filtered by partition
- Columns included in input can be specified
- HCatalog API provides functions to inspect the table schema programmatically

```
protected void map(WritableComparable key, HCatRecord value, Context context)
  throws IOException, InterruptedException {
  // Just select and emit the cardinal number and its native word
  int cardinal = (String) value.get(1);
  String native = (Integer) value.get(3);
  context.write(new IntWritable(cardinal), new Text(native));
}
```

MapReduce and HCatalog

```
public int run(String[] args) throws Exception {
  Configuration conf = getConf();
  String dbName = “my_db”;
  String tableName = “counting”;

  Job job = new Job(conf, "GroupByAge");
  HCatInputFormat.setInput(job, InputJobInfo.create(dbName,
                               inputTableName, null));
  job.setInputFormatClass(HCatInputFormat.class);
  job.setMapperClass(Map.class);
  job.setMapOutputKeyClass(IntWritable.class);
  job.setMapOutputValueClass(Text.class);

  // Set up output format here.

  return (job.waitForCompletion(true) ? 0 : 1);
}
```