Hive Tutorial

Hive is a data warehousing infrastructure based on Apache Hadoop. Hadoop provides massive scale out and fault tolerance capabilities for data storage and processing on commodity hardware.

In the order of granularity Hive data is organized into Databases, Tables, Partitions, Buckets (or Clusters)

Hive supports primitive and complex data types, as described below.

**Primitive Types**

Integers, Boolean type, Floating point numbers, Fixed point numbers, String types,

Date and time types, Binary types.

**Complex Types**

* Structs: the elements within the type can be accessed using the DOT (.) notation.
* Maps (key-value tuples): The elements are accessed using ['element name'] notation.
* Arrays (indexable lists): The elements in the array have to be in the same type. Elements can be accessed using the [n] notation where n is an index (zero-based) into the array

.

**Browsing Tables and Partitions**

SHOW TABLES;

To list existing tables in the warehouse; there are many of these, likely more than you want to browse.

SHOW TABLES 'page.\*';

To list tables with prefix 'page'. The pattern follows Java regular expression syntax (so the period is a wildcard).

SHOW PARTITIONS page\_view;

To list partitions of a table. If the table is not a partitioned table then an error is thrown.

DESCRIBE page\_view;

To list columns and column types of table.

DESCRIBE EXTENDED page\_view;

To list columns and all other properties of table. This prints lot of information and that too not in a pretty format. Usually used for debugging.

DESCRIBE EXTENDED page\_view PARTITION (ds='2008-08-08');

To list columns and all other properties of a partition. This also prints lot of information which is usually used for debugging.

**Altering Tables**

ALTER TABLE old\_table\_name RENAME TO new\_table\_name;

To rename the columns of an existing table. Be sure to use the same column types, and to include an entry for each preexisting column:

ALTER TABLE old\_table\_name REPLACE COLUMNS (col1 TYPE, ...);

To add columns to an existing table:

ALTER TABLE tab1 ADD COLUMNS (c1 INT COMMENT 'a new int column', c2 STRING DEFAULT 'def val');

**Dropping Tables and Partitions**

DROP TABLE pv\_users;

To dropping a partition. Alter the table to drop the partition.

ALTER TABLE pv\_users DROP PARTITION (ds='2008-08-08')

**Querying and Inserting Data**

Simple Query

*INSERT OVERWRITE TABLE user\_active*

*SELECT user.\**

*FROM user*

*WHERE user.active = 1;*

Partition Based Query

*INSERT OVERWRITE TABLE xyz\_com\_page\_views*

*SELECT page\_views.\**

*FROM page\_views*

*WHERE page\_views.date >= '2008-03-01' AND page\_views.date <= '2008-03-31' AND*

*page\_views.referrer\_url like '%xyz.com';*

Joins

*INSERT OVERWRITE TABLE pv\_users*

*SELECT pv.\*, u.gender, u.age*

*FROM user u JOIN page\_view pv ON (pv.userid = u.id)*

*WHERE pv.date = '2008-03-03';*

Aggregations

*INSERT OVERWRITE TABLE pv\_gender\_sum*

*SELECT pv\_users.gender, count (DISTINCT pv\_users.userid)*

*FROM pv\_users*

*GROUP BY pv\_users.gender;*

Multi Table/File Inserts

*FROM pv\_users*

*INSERT OVERWRITE TABLE pv\_gender\_sum*

*SELECT pv\_users.gender, count\_distinct(pv\_users.userid)*

*GROUP BY pv\_users.gender*

*INSERT OVERWRITE DIRECTORY '/user/data/tmp/pv\_age\_sum'*

*SELECT pv\_users.age, count\_distinct(pv\_users.userid)*

*GROUP BY pv\_users.age;*

Dynamic-Partition Insert

*FROM page\_view\_stg pvs*

*INSERT OVERWRITE TABLE page\_view PARTITION(dt='2008-06-08', country='US')*

*SELECT pvs.viewTime, pvs.userid, pvs.page\_url, pvs.referrer\_url, null, null, pvs.ip WHERE pvs.country = 'US'*

*INSERT OVERWRITE TABLE page\_view PARTITION(dt='2008-06-08', country='CA')*

*SELECT pvs.viewTime, pvs.userid, pvs.page\_url, pvs.referrer\_url, null, null, pvs.ip WHERE pvs.country = 'CA'*

*INSERT OVERWRITE TABLE page\_view PARTITION(dt='2008-06-08', country='UK')*

*SELECT pvs.viewTime, pvs.userid, pvs.page\_url, pvs.referrer\_url, null, null, pvs.ip WHERE pvs.country = 'UK';*

Inserting into Local Files

*INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv\_gender\_sum'*

*SELECT pv\_gender\_sum.\**

*FROM pv\_gender\_sum;*

Sampling

*INSERT OVERWRITE TABLE pv\_gender\_sum\_sample*

*SELECT pv\_gender\_sum.\**

*FROM pv\_gender\_sum TABLESAMPLE(BUCKET 3 OUT OF 32);*

Union All

*INSERT OVERWRITE TABLE actions\_users*

*SELECT u.id, actions.date*

*FROM (*

*SELECT av.uid AS uid*

*FROM action\_video av*

*WHERE av.date = '2008-06-03'*

*UNION ALL*

*SELECT ac.uid AS uid*

*FROM action\_comment ac*

*WHERE ac.date = '2008-06-03'*

*) actions JOIN users u ON(u.id = actions.uid);*

Array Operations

*CREATE TABLE array\_table (int\_array\_column ARRAY<INT>);*

Map (Associative Arrays) Operations

*INSERT OVERWRITE page\_views\_map*

*SELECT pv.userid, pv.properties['page type']*

*FROM page\_views pv;*

Custom Map/Reduce Scripts

*FROM (*

*FROM pv\_users*

*MAP pv\_users.userid, pv\_users.date*

*USING 'map\_script'*

*AS dt, uid*

*CLUSTER BY dt) map\_output*

*INSERT OVERWRITE TABLE pv\_users\_reduced*

*REDUCE map\_output.dt, map\_output.uid*

*USING 'reduce\_script'*

*AS date, count;*

Co-Groups

*FROM (*

*FROM (*

*FROM action\_video av*

*SELECT av.uid AS uid, av.id AS id, av.date AS date*

*UNION ALL*

*FROM action\_comment ac*

*SELECT ac.uid AS uid, ac.id AS id, ac.date AS date*

*) union\_actions*

*SELECT union\_actions.uid, union\_actions.id, union\_actions.date*

*CLUSTER BY union\_actions.uid) map*

*INSERT OVERWRITE TABLE actions\_reduced*

*SELECT TRANSFORM(map.uid, map.id, map.date) USING 'reduce\_script' AS (uid, id, reduced\_val);*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*