Hive & SQL

DATA LOAD

Load operations are currently pure copy/move operations that move datafiles into location corresponding to Hive tables.

```
create table if not exists saurav_tbl_csv
(
id string,
name string,
marks int
)
row format delimited
fields terminated by ','
STORED AS TEXTFILE;
LOAD DATA LOCAL INPATH '/tmp/saurav.csv' INTO TABLE saurav_tbl_csv;
```

filepath can refer to a file (in which case Hive will move the file into the table) or it can be a directory (in which case Hive will move all the files within that directory into the table).

If the keyword LOCAL is specified, then the load command will look for filepath in the local file system.

If a relative path is specified, it will be interpreted relative to the user's current working directory.

The load command will try to copy all the files addressed by filepath to the target filesystem.

DATA LOAD (CONTD.)

If the OVERWRITE keyword is used then the contents of the target table (or partition) will be deleted and replaced by the files referred to by filepath, otherwise the files referred by filepath will be added to the table.

Query results can be inserted into filesystem directories.

```
INSERT OVERWRITE [LOCAL] DIRECTORY directory1
[ROW FORMAT row_format] [STORED AS file_format] (Note: Only available starting with
    Hive 0.11.0)
  SELECT ... FROM ...
```

QUIZ

Q• Identify the correct options

1. The SQL-like queries in Hive are translated to MapReduce jobs.

2. We can not access hive tables without Metastore.

3. A table with less records will have lesser entries in Metastore as compared to tables with larger records.

4. EXPLAIN gives the Abstract Syntax Tree of execution and then executes the MR job.

ANSWER

Q• Identify the correct options

1. The SQL-like queries in Hive are translated to MapReduce jobs.
2. We can not access hive tables without Metastore.
3. A table with less records will have lesser entries in Metastore as compared to tables with larger records.
4. EXPLAIN gives the Abstract Syntax Tree of execution and then executes the MR job.

TABLE TYPES

When you drop an internal table, it drops the data, and it also drops the metadata.

When you drop an external table, it only drops the meta data.

On dropping the external table, the data does not get deleted from HDFS.

Thus it is evident that the external table is just a pointer on HDFS data.

TABLE TYPES (CONTD.)

Use EXTERNAL tables when:

The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.

Data needs to remain in the underlying location even after a DROP TABLE. This can applyif you are pointing multiple schemas (tables or views) at a single data set or if you areiterating through various possible schemas.

Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.

You are not creating table based on existing table (AS SELECT).

Use INTERNAL tables when:

The data is temporary.

You want Hive to completely manage the lifecycle of the table and data.

Demo: ExternalTable.ddl

COMPLEX DATA TYPES

Complex Types can be built up from primitive types and other composite types.

Data type of the fields in the collection are specified using an angled bracket notation.

Currently Hive supports four complex data types. They are:

ARRAY

ARRAY<data_type>

An Ordered sequences of similar type elements that are indexable using

zero-based integers.

It is similar to arrays in Java.

Example – array ('1', 'bala', 'praveen');

Second element is accessed with array[1].

COMPLEX DATA TYPES (CONTD.)

MAP

MAP<primitive_type, data_type>

Collection of key-value pairs.

Fields are accessed using array notation of keys (e.g., [‘key’]).

STRUCT

STRUCT<col_name : data_type [COMMENT col_comment], ...>

It is similar to STRUCT in C language.

It is a record type which encapsulates a set of named fields that can be any primitive data type.

Elements in STRUCT type are accessed using the DOT (.) notation.

Example – For a column c of type STRUCT {a INT; b INT} the a field is accessed by the expression c.a

COMPLEX DATA TYPES (CONTD.)

UNIONTYPE

UNIONTYPE<data_type, data_type, ...>

It is similar to Unions in C.

At any point of time, an Union Type can hold any one (exactly one) data type from its specified data types.

QUIZ

Q. Identify the correct options

1. We can not insert data in EXTERNAL tables.-
2. We can store integers in STRING array, in which data will be implicitly converted.-
3. We can not have a single Hive tables with different complex types.-
4. Struct can contain different data types referred by different names.-

ANSWER

Q.Identify the correct options

1. We can not insert data in EXTERNAL tables.
2. We can store integers in STRING array, in which data will be implicitly converted.
3. We can not have a single Hive tables with different complex types.
4. Struct can contain different data types referred by different names.

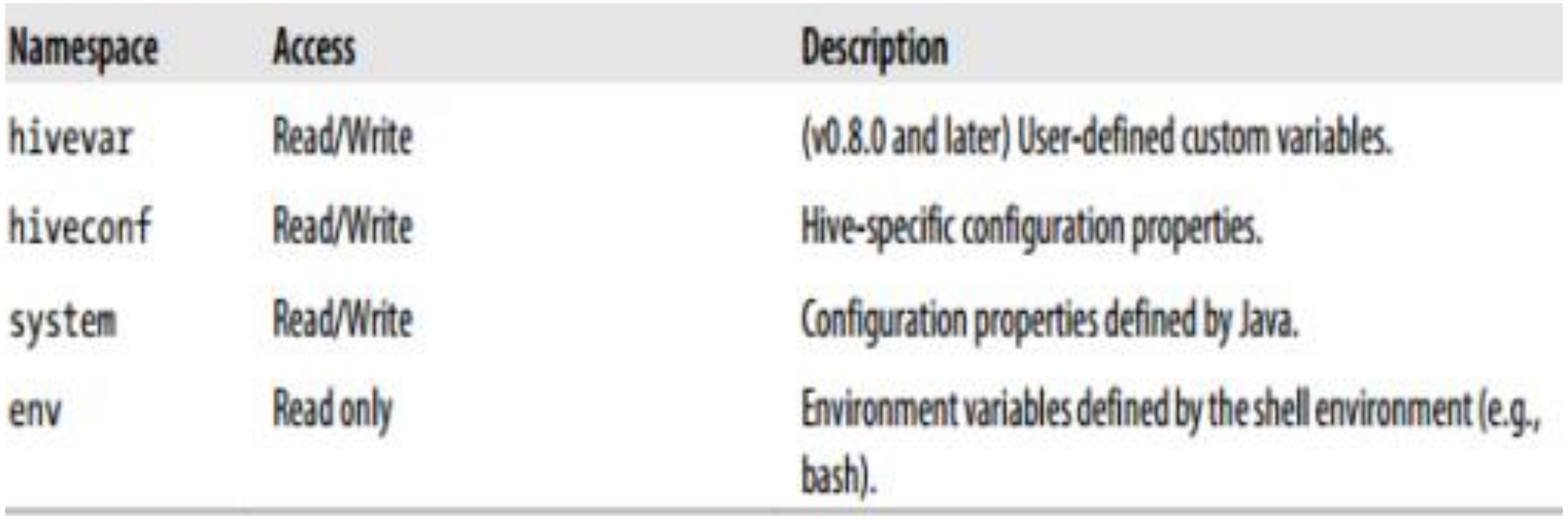
HIVE VARIABLES

You can define variables on the command line or inside script so that you can reference in Hive scripts to customize execution.

Inside the CLI, variables are displayed and changed using the SET command.

Hive’s variables are internally stored as Java Strings.

You can reference variables in queries; Hive replaces the reference with the variable’s value before sending the query to the query processor.



Namespace	Access	Description
hivevar	Read/Write	(v0.8.0 and later) User-defined custom variables.
hiveconf	Read/Write	Hive-specific configuration properties.
system	Read/Write	Configuration properties defined by Java.
env	Read only	Environment variables defined by the shell environment (e.g., bash).

HIVE VARIABLES AND EXECUTION CUSTOMIZATION

--hiveconf option is used for all properties that configure Hive behavior.

Unlike hivevar variables, you have to use the system: or env: prefix with system properties and environment variables.

-e option is used to execute query in a single shot.

Adding the –S, starts execution in silent mode.

Hive can execute one or more queries that were saved to a file using the -f file argument

In $HOME/.hiverc file, one can specify a file of commands for the CLI to run as it starts, before showing you the prompt. Hive automatically looks for a file named .hiverc in your HOME directory and runs the commands it contains, if any.

These files are convenient for commands that you run frequently, such as setting system or adding Java archives (JAR files) of custom Hive extensions to Hadoop’s distributed cache.

SORT BY VS ORDER BY VS CLUSTER BY VS DISTRIBUTE BY IN HIVE

Sort By:-

Hive uses sort by to sort the data based on the data type of the column to be used for sorting per reducer i.e. overall sorting of output is not maintained. e.g. if column is of numeric type the data will be sorted per reducer in numeric order.

ORDER BY:-

Very similar to ORDER BY of SQL. The overall sorting is maintained in case of order by and output is produced in single reducer. Hence, we need to use limit clause so that reducer is not overloaded.

DISTRIBUTE BY:-

Hive uses the columns in *Distribute By* to distribute the rows among reducers. All rows with the same *Distribute By* columns will go to the same reducer. However,*Distribute By* does not guarantee clustering or sorting properties on the distributed keys.

CLUSTER BY:-

Cluster By is a short-cut for both *Distribute By* and *Sort By*.

http://anujaneja.com/wp/2016/11/14/sort-by-vs-order-by-vs-cluster-by-vs-distribute-by-in-hive/

PARTITIONING

Hive organizes tables horizontally into partitions.

•It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, department etc.

•Using partition, it is easy to query a portion of the data.

•Partitioning can be done based on more than column which will impose multi-dimensional structure on directory storage.

•In Hive, partitioning is supported for both managed and external tables.

•The partition statement lets Hive alter the way it manages the underlying structures of the table’s data directory.

•In case of partitioned tables, subdirectories are created under the table’s data directory for each unique value of a partition column.

•When a partitioned table is queried with one or both partition columns in criteria or in the WHERE clause, what Hive effectively does is partition elimination by scanning only those data directories that are needed.

•If no partitioned columns are used, then all the directories are scanned (full table scan) and partitioning will not have any effect.

CLASSIFICATION OF PARTITIONING

- Static partitioning
- Dynamic Partitioning

When to use static partitioning

- Static partitioning needs to be applied when we know data (supposed to be inserted) belongs to which partition.

When to use dynamic partitioning

- In static partitioning, every partitioning needs to be backed with individual hive statement which is not feasible for large number of partitions as it will require writing of lot of hive statements.
- In that scenario dynamic partitioning is suggested as we can create as many number of partitions with single hive statement.

BUCKETING

Bucketing concept is based on (hashing function on the bucketed column) mod (by total number of buckets). The hash_function depends on the type of the bucketing column.

•Records with the same bucketed column will always be stored in the same bucket.

•We use CLUSTERED BY clause to divide the table into buckets.

•Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based.

•Bucketing can be done along with Partitioning on Hive tables and even without partitioning.

•Bucketed tables will create almost equally distributed data file parts, unless there is skew in data.

•Bucketing is enabled by setting hive.enforce.bucketing= true;

Advantages

•Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.

•As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables.

•Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient, since the join of each bucket becomes an efficient merge-sort.

BUCKETING VS PARTITIONING

•Partitioning helps in elimination of data, if used in WHERE clause, where as bucketing helps in organizing data in each partition into multiple files, so that the same set of data is always written in same bucket.

•Bucketing helps a lot in joining of columns.

•Hive Bucket is nothing but another technique of decomposing data or decreasing the data into more manageable parts or equal parts.

•Partitioning a table stores data in sub-directories categorized by table location, which allows Hive to exclude unnecessary data from queries without reading all the data every time a new query is made.

•Hive does support Dynamic Partitioning (DP) where column values are only known at EXECUTION TIME. To enable Dynamic Partitioning :

SET hive.exec.dynamic.partition =true;

•Another situation we want to protect against dynamic partition insert is that the user may accidentally specify all partitions to be dynamic partitions without specifying one static partition, while the original intention is to just overwrite the sub-partitions of one root partition.

SET hive.exec.dynamic.partition.mode =strict;

To enable bucketing:

SET hive.enforce.bucketing =true;