**Hive Data Definition Language**

* [Hive Data Definition Language](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-HiveDataDefinitionLanguage)
  + [Create/Drop Database](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDropDatabase)
    - [Create Database](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateDatabase)
    - [Drop Database](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropDatabase)
  + [Create/Drop Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDropTable)
    - [Create Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable)
      * [Inserting Data Into Bucketed Tables](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-InsertingDataIntoBucketedTables)
    - [Drop Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropTable)
  + [Alter Table/Partition Statements](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTable%2FPartitionStatements)
    - [Add Partitions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AddPartitions)
    - [Drop Partitions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropPartitions)
    - [Rename Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-RenameTable)
    - [Change Column Name/Type/Position/Comment](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ChangeColumnName%2FType%2FPosition%2FComment)
    - [Add/Replace Columns](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Add%2FReplaceColumns)
    - [Alter Table Properties](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTableProperties)
    - [Add Serde Properties](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AddSerdeProperties)
    - [Alter Table/Partition File Format](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTable%2FPartitionFileFormat)
    - [Alter Table Storage Properties](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTableStorageProperties)
    - [Alter Table/Partition Location](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTable%2FPartitionLocation)
    - [Alter Table Touch](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTableTouch)
    - [Alter Table (Un)Archive](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterTable%28Un%29Archive)
  + [Create/Drop View](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDropView)
    - [Create View](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateView)
    - [Drop View](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropView)
    - [Alter View Properties](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-AlterViewProperties)
  + [Create/Drop Function](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDropFunction)
    - [Create Function](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateFunction)
    - [Drop Function](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropFunction)
  + [Create/Drop Index](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDropIndex)
    - [Create Index](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateIndex)
    - [Drop Index](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropIndex)
  + [Show/Describe Statements](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Show%2FDescribeStatements)
    - [Show Databases](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowDatabases)
    - [Show Tables](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowTables)
    - [Show Partitions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowPartitions)
    - [Show Table/Partitions Extended](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowTable%2FPartitionsExtended)
    - [Show Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowFunctions)
    - [Show Indexes](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-ShowIndexes)
    - [Describe Database](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DescribeDatabase)
    - [Describe Table/Column](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DescribeTable%2FColumn)

**Create/Drop Database**

**Create Database**

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name

[COMMENT database\_comment]

[LOCATION hdfs\_path]

[WITH DBPROPERTIES (property\_name=property\_value, ...)];

The use of SCHEMA and DATABASE are interchangeable – they mean the same thing.

**Drop Database**

DROP (DATABASE|SCHEMA) [IF EXISTS] database\_name [RESTRICT|CASCADE];

The use of SCHEMA and DATABASE are interchangeable – they mean the same thing.

**Create/Drop Table**

**Create Table**

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.]table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]

[CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]

[

[ROW FORMAT row\_format] [STORED AS file\_format]

| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)] (Note: only available starting with 0.6.0)

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (property\_name=property\_value, ...)] (Note: only available starting with 0.6.0)

[AS select\_statement] (Note: this feature is only available starting with 0.5.0.)

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.]table\_name

LIKE existing\_table\_or\_view\_name

[LOCATION hdfs\_path]

data\_type

: primitive\_type

| array\_type

| map\_type

| struct\_type

primitive\_type

: TINYINT

| SMALLINT

| INT

| BIGINT

| BOOLEAN

| FLOAT

| DOUBLE

| STRING

array\_type

: ARRAY < data\_type >

map\_type

: MAP < primitive\_type, data\_type >

struct\_type

: STRUCT < col\_name : data\_type [COMMENT col\_comment], ...>

row\_format

: DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]

[MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]

| SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

file\_format:

: SEQUENCEFILE

| TEXTFILE

| RCFILE (Note: only available starting with 0.6.0)

| INPUTFORMAT input\_format\_classname OUTPUTFORMAT output\_format\_classname

CREATE TABLE creates a table with the given name. An error is thrown if a table or view with the same name already exists. You can use IF NOT EXISTS to skip the error.

The EXTERNAL keyword lets you create a table and provide a LOCATION so that Hive does not use a default location for this table. This comes in handy if you already have data generated. When dropping an EXTERNAL table, data in the table is NOT deleted from the file system.

The LIKE form of CREATE TABLE allows you to copy an existing table definition exactly (without copying its data). Before 0.8.0, CREATE TABLE LIKE view\_name would make a copy of the view. From 0.8.0, CREATE TABLE LIKE view\_name creates a table by adopting the schema of view\_name (fields and partition columns) using defaults for serde and file formats.

You can create tables with custom SerDe or using native SerDe. A native SerDe is used if ROW FORMAT is not specified or ROW FORMAT DELIMITED is specified. You can use the DELIMITED clause to read delimited files. Use the SERDE clause to create a table with custom SerDe. Refer to SerDe section of the User Guide for more information on SerDe.

You must specify a list of a columns for tables that use a native SerDe. Refer to the Types part of the User Guide for the allowable column types. A list of columns for tables that use a custom SerDe may be specified but Hive will query the SerDe to determine the actual list of columns for this table.

Use STORED AS TEXTFILE if the data needs to be stored as plain text files. Use STORED AS SEQUENCEFILE if the data needs to be compressed. Please read more about [CompressedStorage](https://cwiki.apache.org/confluence/display/Hive/CompressedStorage) if you are planning to keep data compressed in your Hive tables. Use INPUTFORMAT and OUTPUTFORMAT to specify the name of a corresponding InputFormat and OutputFormat class as a string literal, e.g. 'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'.

Use STORED BY to create a non-native table, for example in HBase. See [StorageHandlers](https://cwiki.apache.org/confluence/display/Hive/StorageHandlers) for more information on this option.

Partitioned tables can be created using the PARTITIONED BY clause. A table can have one or more partition columns and a separate data directory is created for each distinct value combination in the partition columns. Further, tables or partitions can be bucketed using CLUSTERED BY columns, and data can be sorted within that bucket via SORT BY columns. This can improve performance on certain kinds of queries.

If, when creating a partitioned table, you get this error: "FAILED: Error in semantic analysis: Column repeated in partitioning columns," it means you are trying to include the partitioned column in the data of the table itself. You probably really do have the column defined. However, the partition you create makes a pseudocolumn on which you can query, so you must rename your table column to something else (that users should not query on!).

Here is an example. Suppose your original table was this:

id int,

date date,

name varchar

Now you want to partition on date. Your Hive definition would be this:

create table table\_name (

id int,

dtDontQuery string,

name string

)

partitioned by (date string)

Now your users will still query on "where date = '...'" but the 2nd column will be the original values.

Table names and column names are case insensitive but SerDe and property names are case sensitive. Table and column comments are string literals (single-quoted). The TBLPROPERTIES clause allows you to tag the table definition with your own metadata key/value pairs.

Tables can also be created and populated by the results of a query in one create-table-as-select (CTAS) statement. The table created by CTAS is atomic, meaning that the table is not seen by other users until all the query results are populated. So other users will either see the table with the complete results of the query or will not see the table at all.

There are two parts in CTAS, the SELECT part can be any [SELECT statement](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select) supported by HiveQL. The CREATE part of the CTAS takes the resulting schema from the SELECT part and creates the target table with other table properties such as the SerDe and storage format. The only restrictions in CTAS is that the target table cannot be a partitioned table (nor can it be an external table).

Examples:

Here's an example statement to create a table:

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

STORED AS SEQUENCEFILE;

The statement above creates the page\_view table with viewTime, userid, page\_url, referrer\_url, and ip columns (including comments). The table is also partitioned and data is stored in sequence files. The data format in the files is assumed to be field-delimited by ctrl-A and row-delimited by newline.

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

STORED AS SEQUENCEFILE;

The above statement lets you create the same table as the previous table.

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

STORED AS SEQUENCEFILE;

In the example above, the page\_view table is bucketed (clustered by) userid and within each bucket the data is sorted in increasing order of viewTime. Such an organization allows the user to do efficient sampling on the clustered column - in this case userid. The sorting property allows internal operators to take advantage of the better-known data structure while evaluating queries, also increasing efficiency. MAP KEYS and COLLECTION ITEMS keywords can be used if any of the columns are lists or maps.

In all the examples until now the data is stored in <hive.metastore.warehouse.dir>/page\_view. Specify a value for the key hive.metastore.warehouse.dir in Hive config file hive-site.xml.

CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User',

country STRING COMMENT 'country of origination')

COMMENT 'This is the staging page view table'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

STORED AS TEXTFILE

LOCATION '<hdfs\_location>';

You can use the above statement to create a page\_view table which points to any hdfs location for its storage. But you still have to make sure that the data is delimited as specified in the query above.

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;

The above CTAS statement creates the target table new\_key\_value\_store with the schema (new\_key DOUBLE, key\_value\_pair STRING) derived from the results of the SELECT statement. If the SELECT statement does not specify column aliases, the column names will be automatically assigned to \_col0, \_col1, and \_col2 etc. In addition, the new target table is created using a specific SerDe and a storage format independent of the source tables in the SELECT statement.

CREATE TABLE empty\_key\_value\_store

LIKE key\_value\_store;

In contrast, the statement above creates a new empty\_key\_value\_store table whose definition exactly matches the existing key\_value\_store in all particulars other than table name. The new table contains no rows.

**Inserting Data Into Bucketed Tables**

The CLUSTERED BY and SORTED BY creation commands do not affect how data is inserted into a table – only how it is read. This means that users must be careful to insert data correctly by specifying the number of reducers to be equal to the number of buckets, and using CLUSTER BY and SORT BY commands in their query.

There is also an example of [creating and populating bucketed tables](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL+BucketedTables).

**Drop Table**

DROP TABLE [IF EXISTS] table\_name

DROP TABLE removes metadata and data for this table. The data is actually moved to the .Trash/Current directory if Trash is configured. The metadata is completely lost.

When dropping an EXTERNAL table, data in the table will NOT be deleted from the file system.

When dropping a table referenced by views, no warning is given (the views are left dangling as invalid and must be dropped or recreated by the user).

See the next section on ALTER TABLE for how to drop partitions.

Otherwise, the table information is removed from the metastore and the raw data is removed as if by 'hadoop dfs -rm'. In many cases, this results in the table data being moved into the user's .Trash folder in their home directory; users who mistakenly DROP TABLEs mistakenly may thus be able to recover their lost data by re-creating a table with the same schema, re-creating any necessary partitions, and then moving the data back into place manually using Hadoop. This solution is subject to change over time or across installations as it relies on the underlying implementation; users are strongly encouraged not to drop tables capriciously.

In Hive 0.70 or later, DROP returns an error if the table doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

**Alter Table/Partition Statements**

Alter table statements enable you to change the structure of an existing table. You can add columns/partitions, change serde, add table and serde properties, or rename the table itself. Similarly, alter table partition statements allow you change the properties of a specific partition in the named table.

**Add Partitions**

ALTER TABLE table\_name ADD [IF NOT EXISTS] PARTITION partition\_spec [LOCATION 'location1'] partition\_spec [LOCATION 'location2'] ...

partition\_spec:

: (partition\_col = partition\_col\_value, partition\_col = partiton\_col\_value, ...)

You can use ALTER TABLE ADD PARTITION to add partitions to a table. Partition values should be quoted only if they are strings. The location must be a directory inside of which data files reside.

Note that it is proper syntax to have multiple partition\_spec in a single ALTER TABLE, but if you do this in version 0.7, your partitioning scheme will fail. That is, every query specifying a partition will always use only the first partition. Instead, you should use the following form if you want to add many partitions:

ALTER TABLE table\_name ADD PARTITION (partCol = 'value1') location 'loc1';

ALTER TABLE table\_name ADD PARTITION (partCol = 'value2') location 'loc2';

...

ALTER TABLE table\_name ADD PARTITION (partCol = 'valueN') location 'locN';

Specifically, the following example (which was the default example before) will FAIL silently and without error, and all queries will go only to dt='2008-08-08' partition, no matter which partition you specify.

ALTER TABLE page\_view ADD PARTITION (dt='2008-08-08', country='us') location '/path/to/us/part080808' PARTITION (dt='2008-08-09', country='us') location '/path/to/us/part080809';

An error is thrown if the partition\_spec for the table already exists. You can use IF NOT EXISTS to skip the error.

**Drop Partitions**

ALTER TABLE table\_name DROP [IF EXISTS] partition\_spec, partition\_spec,...

You can use ALTER TABLE DROP PARTITION to drop a partition for a table. This removes the data and metadata for this partition.

In Hive 0.70 or later, DROP returns an error if the partition doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

ALTER TABLE page\_view DROP PARTITION (dt='2008-08-08', country='us');

**Rename Table**

ALTER TABLE table\_name RENAME TO new\_table\_name

This statement lets you change the name of a table to a different name.

As of version 0.6, a rename on a managed table moves its HDFS location as well. (Older Hive versions just renamed the table in the metastore without moving the HDFS location.)

**Change Column Name/Type/Position/Comment**

ALTER TABLE table\_name CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type [COMMENT col\_comment] [FIRST|AFTER column\_name]

This command will allow users to change a column's name, data type, comment, or position, or an arbitrary combination of them.

Example: CREATE TABLE test\_change (a int, b int, c int);

"ALTER TABLE test\_change CHANGE a a1 INT;" will change column a's name to a1.

"ALTER TABLE test\_change CHANGE a a1 STRING AFTER b;" will change column a's name to a1, a's data type to string, and put it after column b. The new table's structure is: b int, a1 string, c int.

"ALTER TABLE test\_change CHANGE b b1 INT FIRST;" will change column b's name to b1, and put it as the first column. The new table's structure is: b1 int, a string, c int.

NOTE: The column change command will only modify Hive's metadata, and will NOT touch data. Users should make sure the actual data layout conforms with the metadata definition.

**Add/Replace Columns**

ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

ADD COLUMNS lets you add new columns to the end of the existing columns but before the partition columns.

REPLACE COLUMNS removes all existing columns and adds the new set of columns. This can be done only for tables with native serde (DynamicSerDe or MetadataTypedColumnsetSerDe). Refer to SerDe section of User Guide for more information. REPLACE COLUMNS can also be used to drop columns. For example:

"ALTER TABLE test\_change REPLACE COLUMNS (a int, b int);" will remove column `c' from test\_change's schema. Note that this does not delete underlying data, it just changes the schema.

**Alter Table Properties**

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties

table\_properties:

: (property\_name = property\_value, property\_name = property\_value, ... )

You can use this statement to add your own metadata to the tables. Currently last\_modified\_user, last\_modified\_time properties are automatically added and managed by Hive. Users can add their own properties to this list. You can do DESCRIBE EXTENDED TABLE to get this information.

**Add Serde Properties**

ALTER TABLE table\_name SET SERDE serde\_class\_name [WITH SERDEPROPERTIES serde\_properties]

ALTER TABLE table\_name SET SERDEPROPERTIES serde\_properties

serde\_properties:

: (property\_name = property\_value, property\_name = property\_value, ... )

This statement enables you to add user defined metadata to table SerDe object. The serde properties are passed to the table's SerDe when it is being initialized by Hive to serialize and deserialize data. So users can store any information required for their custom serde here. Refer to SerDe section of Users Guide for more information.

Example, note that both property\_name and property\_value must be quoted:

ALTER TABLE table\_name SET SERDEPROPERTIES ('field.delim' = ',');

**Alter Table/Partition File Format**

ALTER TABLE table\_name [PARTITION partitionSpec] SET FILEFORMAT file\_format

This statement changes the table's (or partition's) file format. For available file\_format options, see the section above on CREATE TABLE.

**Alter Table Storage Properties**

ALTER TABLE table\_name CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name, ...)] INTO num\_buckets BUCKETS

These statements change the table's physical storage properties.

NOTE: These commands will only modify Hive's metadata, and will NOT reorganize or reformat existing data. Users should make sure the actual data layout conforms with the metadata definition.

**Alter Table/Partition Location**

ALTER TABLE table\_name [PARTITION partitionSpec] SET LOCATION "new location"

**Alter Table Touch**

ALTER TABLE table\_name TOUCH [PARTITION partitionSpec];

TOUCH reads the metadata, and writes it back. This has the effect of causing the pre/post execute hooks to fire. An example use case is if you have a hook that logs all the tables/partitions that were modified, along with an external script that alters the files on HDFS directly. Since the script modifies files outside of hive, the modification wouldn't be logged by the hook. The external script could call TOUCH to fire the hook and mark the said table or partition as modified.

Also, it may be useful later if we incorporate reliable last modified times. Then touch would update that time as well.

Note that TOUCH doesn't create a table or partition if it doesn't already exist. (See [Create Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create.2BAC8DropTable))

**Alter Table (Un)Archive**

ALTER TABLE table\_name ARCHIVE PARTITION partition\_spec;

ALTER TABLE table\_name UNARCHIVE PARTITION partition\_spec;

Archiving is a feature to moves a partition's files into a Hadoop Archive (HAR). Note that only the file count will be reduced; HAR does not provide any compression. See [LanguageManual Archiving](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Archiving) for more information

**Create/Drop View**

*Note:* View support is only available starting in Hive 0.6.

**Create View**

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...) ]

[COMMENT view\_comment]

[TBLPROPERTIES (property\_name = property\_value, ...)]

AS SELECT ...

CREATE VIEW creates a view with the given name. An error is thrown if a table or view with the same name already exists. You can use IF NOT EXISTS to skip the error.

If no column names are supplied, the names of the view's columns will be derived automatically from the defining SELECT expression. (If the SELECT contains unaliased scalar expressions such as x+y, the resulting view column names will be generated in the form \_C0, \_C1, etc.) When renaming columns, column comments can also optionally be supplied. (Comments are not automatically inherited from underlying columns.)

A CREATE VIEW statement will fail if the view's defining SELECT expression is invalid.

Note that a view is a purely logical object with no associated storage. (No support for materialized views is currently available in Hive.) When a query references a view, the view's definition is evaluated in order to produce a set of rows for further processing by the query. (This is a conceptual description; in fact, as part of query optimization, Hive may combine the view's definition with the query's, e.g. pushing filters from the query down into the view.)

A view's schema is frozen at the time the view is created; subsequent changes to underlying tables (e.g. adding a column) will not be reflected in the view's schema. If an underlying table is dropped or changed in an incompatible fashion, subsequent attempts to query the invalid view will fail.

Views are read-only and may not be used as the target of LOAD/INSERT/ALTER.

A view may contain ORDER BY and LIMIT clauses. If a referencing query also contains these clauses, the query-level clauses are evaluated **after** the view clauses (and after any other operations in the query). For example, if a view specifies LIMIT 5, and a referencing query is executed as (select \* from v LIMIT 10), then at most 5 rows will be returned.

Example of view creation:

CREATE VIEW onion\_referrers(url COMMENT 'URL of Referring page')

COMMENT 'Referrers to The Onion website'

AS

SELECT DISTINCT referrer\_url

FROM page\_view

WHERE page\_url='http://www.theonion.com';

**Drop View**

DROP VIEW [IF EXISTS] view\_name

DROP VIEW removes metadata for the specified view. (It is illegal to use DROP TABLE on a view.)

When dropping a view referenced by other views, no warning is given (the dependent views are left dangling as invalid and must be dropped or recreated by the user).

In Hive 0.70 or later, DROP returns an error if the view doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

Example:

DROP VIEW onion\_referrers;

**Alter View Properties**

ALTER VIEW view\_name SET TBLPROPERTIES table\_properties

table\_properties:

: (property\_name = property\_value, property\_name = property\_value, ...)

As with ALTER TABLE, you can use this statement to add your own metadata to a view.

**Create/Drop Function**

**Create Function**

CREATE TEMPORARY FUNCTION function\_name AS class\_name

This statement lets you create a function that is implemented by the class\_name. You can use this function in Hive queries as long as the session lasts. You can use any class that is in the class path of Hive. You can add jars to class path by executing 'ADD FILES' statements. Please refer to the CLI section in the User Guide for more information on how to add/delete files from the Hive classpath. Using this, you can register User Defined Functions (UDF's).

**Drop Function**

You can unregister a UDF as follows:

DROP TEMPORARY FUNCTION [IF EXISTS] function\_name

In Hive 0.70 or later, DROP returns an error if the function doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

**Create/Drop Index**

Not available until 0.7 release

**Create 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]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

[COMMENT "index comment"]

CREATE INDEX creates an index on a table using the given list of columns as keys. See <http://wiki.apache.org/hadoop/Hive/IndexDev#CREATE_INDEX>

**Drop Index**

DROP INDEX [IF EXISTS] index\_name ON table\_name

DROP INDEX drops the index, as well as deleting the index table.

In Hive 0.70 or later, DROP returns an error if the index doesn't exist, unless IF EXISTS is specified or the configuration variable hive.exec.drop.ignorenonexistent is set to true.

**Show/Describe Statements**

These statements provide a way to query the Hive metastore for existing data and metadata accessible to this Hive system.

**Show Databases**

SHOW (DATABASES|SCHEMAS) [LIKE identifier\_with\_wildcards];

SHOW DATABASES lists all of the databases defined in the metastore. The optional LIKE clause allows the list of databases to be filtered using a regular expression. The regular expression may only contain '**' for any character[s] or '|' for a choice. Examples are 'employees', 'emp**', "emp\*|\*ees', all of which will match the database named 'employees'.

**Show Tables**

SHOW TABLES identifier\_with\_wildcards

SHOW TABLES lists all the base tables and views with names matching the given regular expression. Regular expression can contain only '**' for any character[s] or '|' for a choice. Examples are 'page\_view', 'page\_v**', '**view|page**', all which will match 'page\_view' table. Matching tables are listed in alphabetical order. It is not an error if there are no matching tables found in metastore.

**Show Partitions**

SHOW PARTITIONS table\_name

SHOW PARTITIONS lists all the existing partitions for a given base table. Partitions are listed in alphabetical order.

It is also possible to specify parts of a partition specification to filter the resulting list. Note: This feature is only available starting in version 0.6.

SHOW PARTITIONS table\_name PARTITION(ds='2010-03-03');

SHOW PARTITIONS table\_name PARTITION(hr='12');

SHOW PARTITIONS table\_name PARTITION(ds='2010-03-03', hr='12');

**Show Table/Partitions Extended**

SHOW TABLE EXTENDED [IN|FROM database\_name] LIKE identifier\_with\_wildcards [PARTITION(partition\_desc)]

SHOW TABLE EXTENDED will list information for all tables matching the given regular expression. Users can not use regular expression for table name if a partition specification is present. This command's output includes basic table information and file system information like totalNumberFiles, totalFileSize, maxFileSize, minFileSize,lastAccessTime, and lastUpdateTime. If partition is present, it will output the given partition's file system information instead of table's file system information.

**Show Functions**

SHOW FUNCTIONS "a.\*"

SHOW FUNCTIONS lists all the user defined and builtin functions matching the regular expression. To get all functions use ".\*"

**Show Indexes**

SHOW [FORMATTED] (INDEX|INDEXES) ON table\_with\_index [(FROM|IN) db\_name]

SHOW INDEXES shows all of the indexes on a certain column, as well as information about them: index name, table name, names of the columns used as keys, index table name, index type, and comment. If the FORMATTED keyword is used, then column titles are printed for each column. Not available until 0.7 release.

**Describe Database**

DESCRIBE DATABASE db\_name

DESCRIBE DATABASE will show the name of the database, its comment (if one has been set), and its root location on the filesystem. Not available until the 0.7 release.

**Describe Table/Column**

DESCRIBE [EXTENDED|FORMATTED] table\_name[DOT col\_name ( [DOT field\_name] | [DOT '$elem$'] | [DOT '$key$'] | [DOT '$value$'] )\* ]

DESCRIBE TABLE shows the list of columns including partition columns for the given table. If the EXTENDED keyword is specified then it will show all the metadata for the table in Thrift serialized form. This is generally only useful for debugging and not for general use. If the FORMATTED keyword is specified, then it will show the metadata in a tabular format.

If a table has complex column then you can examine the attributes of this column by specifying table\_name.complex\_col\_name (and '$elem$' for array element, '$key$' for map key, and '$value$' for map value). You can specify this recursively to explore the complex column type.

For a view, DESCRIBE TABLE EXTENDED can be used to retrieve the view's definition. Two relevant attributes are provided: both the original view definition as specified by the user, and an expanded definition used internally by Hive.

DESCRIBE [EXTENDED] table\_name partition\_spec

This statement lists metadata for a given partition. The output is similar to that of DESCRIBE TABLE. Presently, the column information associated with a particular partition is not used while preparing plans.

Example:

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

# [LanguageManual DML](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Franklin Hu](https://cwiki.apache.org/confluence/display/~franklinhu) on Jul 18, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362036&originalId=27821985))

= Hive Data Manipulation Language =

* [Loading files into tables](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Loadingfilesintotables)
  + [Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Syntax)
  + [Synopsis](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Synopsis)
  + [Notes](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Notes)
* [Inserting data into Hive Tables from queries](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-InsertingdataintoHiveTablesfromqueries)
  + [Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Syntax)
  + [Synopsis](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Synopsis)
  + [Notes](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Notes)
* [Writing data into filesystem from queries](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Writingdataintofilesystemfromqueries)
  + [Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Syntax)
  + [Synopsis](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Synopsis)
  + [Notes](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Notes)

There are two primary ways of modifying data in Hive:

### Loading files into tables

Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

##### Syntax

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

##### Synopsis

Load operations are current pure copy/move operations that move datafiles into locations corresponding to Hive tables.

* filepath can be a
  + relative path, eg: project/data1
  + absolute path, eg: /user/hive/project/data1
  + a full URI with scheme and (optionally) an authority, eg: hdfs://namenode:9000/user/hive/project/data1
* The target being loaded to can be a table or a partition. If the table is partitioned, then one must specify a specific partition of the table by specifying values for all of the partitioning columns.
* 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). In either case filepath addresses a set of files.
* 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 current directory of the user. User can specify a full URI for local files as well - for example: [file:///user/hive/project/data1](file:///\\user\hive\project\data1)
  + the load command will try to copy all the files addressed by filepath to the target filesystem. The target file system is inferred by looking at the location attribute of the table. The copied data files will then be moved to the table.
* If the keyword LOCAL is not specified, then Hive will either use the full URI of filepath if one is specified. Otherwise the following rules are applied:
  + If scheme or authority are not specified, Hive will use the scheme and authority from hadoop configuration variable fs.default.name that specifies the Namenode URI.
  + If the path is not absolute - then Hive will interpret it relative to /user/<username>
  + Hive will move the files addressed by filepath into the table (or partition)
* if the OVERWRITE keyword is used then the contents of the target table (or partition) will be deleted and replaced with the files referred to by filepath. Otherwise the files referred by filepath will be added to the table.
  + Note that if the target table (or partition) already has a file whose name collides with any of the filenames contained in filepath - then the existing file will be replaced with the new file.

##### Notes

* filepath cannot contain subdirectories.
* If we are not using the keyword LOCAL - filepath must refer to files within the same filesystem as the table (or partition's) location.
* Hive does some minimal checks to make sure that the files being loaded match the target table. Currently it checks that if the table is stored in sequencefile format - that the files being loaded are also sequencefiles and vice versa.
* Please read [CompressedStorage](https://cwiki.apache.org/confluence/display/Hive/CompressedStorage) if your datafile is compressed

### Inserting data into Hive Tables from queries

Query Results can be inserted into tables by using the insert clause

##### Syntax

Standard syntax:

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement;

INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement;

Hive extension (multiple inserts):

FROM from\_statement

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1

[INSERT OVERWRITE TABLE tablename2 [PARTITION ...] 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 ...] select\_statement2] ...;

Hive extension (dynamic partition inserts):

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;

##### Synopsis

* INSERT OVERWRITE will overwrite any existing data in the table or partition
* INSERT INTO will append to the table or partition keeping the existing data in tact. (Note: INSERT INTO syntax is only available starting in version 0.8)
* Inserts can be done to a table or a partition. If the table is partitioned, then one must specify a specific partition of the table by specifying values for all of the partitioning columns.
* Multiple insert clauses (also known as Multi Table Insert) can be specified in the same query
* The output of each of the select statements is written to the chosen table (or partition). Currently the OVERWRITE keyword is mandatory and implies that the contents of the chosen table or partition are replaced with the output of corresponding select statement.
* The output format and serialization class is determined by the table's metadata (as specified via DDL commands on the table)
* In the dynamic partition inserts, users can give partial partition specification, which means you just specify the list of partition column names in the PARTITION clause. The column values are optional. If a partition column value is given, we call this static partition, otherwise dynamic partition. Each dynamic partition column has a corresponding input column from the select statement. This means that the dynamic partition creation is determined by the value of the input column.

##### Notes

* Multi Table Inserts minimize the number of data scans required. Hive can insert data into multiple tables by scanning the input data just once (and applying different query operators) to the input data.

### Writing data into filesystem from queries

Query results can be inserted into filesystem directories by using a slight variation of the syntax above:

##### Syntax

Standard syntax:

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 SELECT ... FROM ...

Hive extension (multiple inserts):

FROM from\_statement

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1

[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2] ...

##### Synopsis

* directory can be full URI. If scheme or authority are not specified, Hive will use the scheme and authority from hadoop configuration variable fs.default.name that specifies the Namenode URI.
* if LOCAL keyword is used - then Hive will write data to the directory on the local file system.
* Data written to the filesystem is serialized as text with columns separated by ^A and rows separated by newlines. If any of the columns are not of primitive type - then those columns are serialized to JSON format.

##### Notes

* Insert statements to directories, local directories and tables (or partitions) can all be used together within the same query.
* Inserts to HDFS filesystem directories is the best way to extract large amounts of data from Hive. Hive can write to HDFS directories in parallel from within a map-reduce job.

**Hive Authorization**

* [Hive Authorization](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-HiveAuthorization)
  + [Disclaimer](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Disclaimer)
  + [Prerequisites](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Prerequisites)
  + [Users, Groups, and Roles](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Users%2CGroups%2CandRoles)
  + [Creating/Dropping/Using Roles](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Creating%2FDropping%2FUsingRoles)
    - [Create/Drop Role](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Create%2FDropRole)
    - [Grant/Revoke Roles](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Grant%2FRevokeRoles)
    - [Viewing Granted Roles](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-ViewingGrantedRoles)
  + [Privileges](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Privileges)
    - [Grant/Revoke Privileges](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-Grant%2FRevokePrivileges)
    - [Viewing Granted Privileges](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-ViewingGrantedPrivileges)
  + [Hive Operations and Required Privileges](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Authorization#LanguageManualAuthorization-HiveOperationsandRequiredPrivileges)

**Disclaimer**

Hive authorization is not completely secure. In its current form, the authorization scheme is intended primarily to prevent good users from accidentally doing bad things, but makes no promises about preventing malicious users from doing malicious things.

**Prerequisites**

In order to use Hive authorization, there are two parameters that should be set in hive-site.xml

<property>

<name>hive.security.authorization.enabled</name>

<value>true</value>

<description>enable or disable the hive client authorization</description>

</property>

<property>

<name>hive.security.authorization.createtable.owner.grants</name>

<value>ALL</value>

<description>the privileges automatically granted to the owner whenever a table gets created.

An example like "select,drop" will grant select and drop privilege to the owner of the table</description>

</property>

Note that, by default, the hive.security.authorization.createtable.owner.grants are set to null, which would result in the creator of a table having no access to the table.

**Users, Groups, and Roles**

At the core of Hive's authorization system are users, groups, and roles. Roles allow administrators to give a name to a set of grants which can be easily reused. A role may be assigned to users, groups, and other roles. For example, consider a system with the following users and groups:

* <User>: <Groups>
* user\_all\_dbs: group\_db1, group\_db2
* user\_db1: group\_db1
* user\_db2: group\_db2

If we wanted to restrict each user to a specific set of databases, we could use roles to build the authorization mechanism. The administrator would create two roles, called role\_db1 and role\_db2. The role\_db1 role would provide privileges just for the first database, and the role\_db2 role would provide privileges just for the second database. The administrator could then grant the role\_db1 role to group\_db1, or explicitly for the users in the group, and do the same for role\_db2 with the users of the second database. In order to allow users who need to see all databases to get their appropriate privileges, a third role could be created called role\_all\_dbs, which would be granted role\_db1 and role\_db2. When user\_all\_dbs is granted the role\_all\_dbs role, the user implicitly is granted all the privileges of role\_db1 and role\_db2.

Hive roles must be created manually before being used, unlike users and groups. Users and groups are managed by the hive.security.authenticator.manager. When a user connects to a Metastore Server and issues a query, the Metastore will determine the username of the connecting user, and the groups associated with that ushive.security.authorization.ername. That information is then used to determine if the user should have access to the metadata being requested, by comparing the required privileges of the Hive operation to the user privileges using the following rules:

* User privileges (Has the privilege been granted to the user)
* Group privileges (Does the user belong to any groups that the privilege has been granted to)
* Role privileges (Does the user or any of the groups that the user belongs to have a role that grants the privilege)

By default, the Metastore uses the HadoopDefaultAuthenticator for determing user -> group mappings, which determines authorization by using the Unix usernames and groups on the machine where the Metastore is running. To make this more clear, consider a scenario where a user foo is a member of group bar on the machine running the Hive CLI, and connects to a Metastore running on a separate server that also has a user named foo, but on the Metastore Server, foo is a member of group baz. When an operation is executed, the Metastore will determine foo to be in the group baz.

Taking this a step further, it is also possible for the groups that a user belongs to on the Metastore Server may differ from the groups that the same user belongs to, as determined by HDFS. This could be the case if Hive or HDFS are configured to use non-default user -> group mappers, or the Metastore and the Namenode both use the defaults, but the processes are running on different machines, and the user -> group mappings are not the same on each machine.

It is important to realize that Hive Metastore only controls authorization for metadata, and the underlying data is controlled by HDFS, so if permissions and privileges between the two systems are not in sync, users may have access to metadata, but not the physical data. If the user -> group mappings across the Metastore and Namenode are not in sync, as in the scenarios above, a user may have the privileges required to access a table according to the Metastore, but may not have permission to access the underlying files according to the Namenode. This could also happen due to administrator intervention, if permissions on the files were changed by hand, but Metastore grants had not been updated.

**Creating/Dropping/Using Roles**

**Create/Drop Role**

CREATE ROLE role\_name

DROP ROLE role\_name

**Grant/Revoke Roles**

GRANT ROLE role\_name [, role\_name] ...

TO principal\_specification [, principal\_specification] ...

REVOKE ROLE role\_name [, role\_name] ...

FROM principal\_specification [, principal\_specification] ...

principal\_specification

: USER user

| GROUP group

| ROLE role

**Viewing Granted Roles**

SHOW ROLE GRANT principal\_specification

**Privileges**

The following privileges are supported in Hive:

* ALL - Gives users all privileges
* ALTER - Allows users to modify the metadata of an object
* UPDATE - Allows users to modify the physical data of an object
* CREATE - Allows users to create objects. For a database, this means users can create tables, and for a table, this means users can create partitions
* DROP - Allows users to drop objects
* INDEX - Allows users to create indexes on an object (Note: this is not currently implemented)
* LOCK - Allows users to lock or unlock tables when concurrency is enabled
* SELECT - Allows users to access data for objects
* SHOW\_DATABASE - Allows users to view available databases

**Grant/Revoke Privileges**

GRANT

priv\_type [(column\_list)]

[, priv\_type [(column\_list)]] ...

[ON object\_type]

TO principal\_specification [, principal\_specification] ...

[WITH GRANT OPTION]

REVOKE

priv\_type [(column\_list)]

[, priv\_type [(column\_list)]] ...

[ON object\_type priv\_level]

FROM principal\_specification [, principal\_specification] ...

REVOKE ALL PRIVILEGES, GRANT OPTION

FROM user [, user] ...

object\_type:

TABLE

| DATABASE

priv\_level:

db\_name

| tbl\_name

**Viewing Granted Privileges**

SHOW GRANT principal\_specification

[ON object\_type priv\_level [(column\_list)]]

**Hive Operations and Required Privileges**

As of the release of Hive 0.7, only these operations require permissions, according to org.apache.hadoop.hive.ql.plan.HiveOperation:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Operation | ALTER | UPDATE | CREATE | DROP | INDEX | LOCK | SELECT | SHOW\_DATABASE |
| LOAD |  | X |  |  |  |  |  |  |
| EXPORT |  |  |  |  |  |  | X |  |
| IMPORT | X | X |  |  |  |  |  |  |
| CREATE TABLE |  |  | X |  |  |  |  |  |
| CREATE TABLE AS SELECT |  |  | X |  |  |  | X |  |
| DROP TABLE |  |  |  | X |  |  |  |  |
| SELECT |  |  |  |  |  |  | X |  |
| ALTER TABLE ADD COLUMN | X |  |  |  |  |  |  |  |
| ALTER TABLE REPLACE COLUMN | X |  |  |  |  |  |  |  |
| ALTER TABLE RENAME | X |  |  |  |  |  |  |  |
| ALTER TABLE ADD PARTITION |  |  | X |  |  |  |  |  |
| ALTER TABLE DROP PARTITION |  |  |  | X |  |  |  |  |
| ALTER TABLE ARCHIVE |  | X |  |  |  |  |  |  |
| ALTER TABLE UNARCHIVE |  | X |  |  |  |  |  |  |
| ALTER TABLE SET PROPERTIES | X |  |  |  |  |  |  |  |
| ALTER TABLE SET SERDE | X |  |  |  |  |  |  |  |
| ALTER TABLE SET SERDE | X |  |  |  |  |  |  |  |
| ALTER TABLE SET SERDEPROPERTIES | X |  |  |  |  |  |  |  |
| ALTER TABLE CLUSTER BY | X |  |  |  |  |  |  |  |
| ALTER TABLE PROTECT MODE | X |  |  |  |  |  |  |  |
| ALTER PARTITION PROTECT MODE | X |  |  |  |  |  |  |  |
| ALTER TABLE SET FILEFORMAT | X |  |  |  |  |  |  |  |
| ALTER PARTITION SET FILEFORMAT | X |  |  |  |  |  |  |  |
| ALTER TABLE SET LOCATION |  | X |  |  |  |  |  |  |
| ALTER PARTITION SET LOCATION |  | X |  |  |  |  |  |  |
| ALTER TABLE CONCATENATE |  | X |  |  |  |  |  |  |
| ALTER PARTITION CONCATENATE |  | X |  |  |  |  |  |  |
| SHOW DATABASES |  |  |  |  |  |  |  | X |
| LOCK TABLE |  |  |  |  |  | X |  |  |
| UNLOCK TABLE |  |  |  |  |  | X |  |  |

**Hive Cli**

* [Hive Cli](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli#LanguageManualCli-HiveCli)
  + [Hive Command line Options](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli#LanguageManualCli-HiveCommandlineOptions)
  + [Hive interactive Shell Command](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli#LanguageManualCli-HiveinteractiveShellCommand)
    - [Logging](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli#LanguageManualCli-Logging)
    - [Hive Resources](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli#LanguageManualCli-HiveResources)

$HIVE\_HOME/bin/hive is a shell utility which can be used to run hive queries in either interactive or batch mode.

**Hive Command line Options**

Usage:

Usage: hive [-hiveconf x=y]\* [<-i filename>]\* [<-f filename>|<-e query-string>] [-S]

-i <filename> Initialization Sql from file (executed automatically and silently before any other commands)

-e 'quoted query string' Sql from command line

-f <filename> Sql from file

-S Silent mode in interactive shell where only data is emitted

-v Verbose mode (echo executed SQL to the console)

-p <port> connect to Hive Server on port number

-hiveconf x=y Use this to set hive/hadoop configuration variables.

-e and -f cannot be specified together. In the absence of these options, interactive shell is started.

However, -i can be used with any other options. Multiple instances of -i can be used to execute multiple init scripts.

To see this usage help, run hive -h

* Example of running Query from command line
* $HIVE\_HOME/bin/hive -e 'select a.col from tab1 a'

* Example of setting hive configuration variables
* $HIVE\_HOME/bin/hive -e 'select a.col from tab1 a' -hiveconf hive.exec.scratchdir=/home/my/hive\_scratch -hiveconf mapred.reduce.tasks=32

* Example of dumping data out from a query into a file using silent mode
* HIVE\_HOME/bin/hive -S -e 'select a.col from tab1 a' > a.txt

* Example of running a script non-interactively
* HIVE\_HOME/bin/hive -f /home/my/hive-script.sql

* Example of running an initialization script before entering interactive mode
* HIVE\_HOME/bin/hive -i /home/my/hive-init.sql

**Hive interactive Shell Command**

When $HIVE\_HOME/bin/hive is run without either -e/-f option it enters interactive shell mode.

Use ";" (semicolon) to terminate commands. Comments in scripts can be specified using the "--" prefix.

|  |  |
| --- | --- |
| \*Command \* | **Description** |
| quit | Use quit or exit to come out of interactive shell. |
| set <key>=<value> | Use this to set value of particular configuration variable. One thing to note here is that if you misspell the variable name, cli will not show an error. |
| set | This will print list of configuration variables that overridden by user or hive. |
| set -v | This will give all possible hadoop/hive configuration variables. |
| add FILE <value> <value>\* | Adds a file to the list of resources. |
| list FILE | list all the resources already added |
| list FILE <value>\* | Check given resources are already added or not. |
| ! <cmd> | execute a shell command from hive shell |
| dfs <dfs command> | execute dfs command command from hive shell |
| <query string> | executes hive query and prints results to stdout |

Sample Usage:

hive> set mapred.reduce.tasks=32;

hive> set;

hive> select a.\* from tab1;

hive> !ls;

hive> dfs -ls;

**Logging**

Hive uses log4j for logging. These logs are not emitted to the standard output by default but are instead captured to a log file specified by Hive's log4j properties file. By default Hive will use hive-log4j.default in the conf/ directory of the hive installation which writes out logs to/tmp/<userid>/hive.log and uses the WARN level.

It is often desirable to emit the logs to the standard output and/or change the logging level for debugging purposes. These can be done from the command line as follows:

$HIVE\_HOME/bin/hive -hiveconf hive.root.logger=INFO,console

hive.root.logger specifies the logging level as well as the log destination. Specifying console as the target sends the logs to the standard error (instead of the log file).

**Hive Resources**

Hive can manage the addition of resources to a session where those resources need to be made available at query execution time. Any locally accessible file can be added to the session. Once a file is added to a session, hive query can refer to this file by its name (in map/reduce/transform clauses) and this file is available locally at execution time on the entire hadoop cluster. Hive uses Hadoop's Distributed Cache to distribute the added files to all the machines in the cluster at query execution time.

Usage:

ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]\*

LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

* FILE resources are just added to the distributed cache. Typically, this might be something like a transform script to be executed.
* JAR resources are also added to the Java classpath. This is required in order to reference objects they contain such as UDF's.
* ARCHIVE resources are automatically unarchived as part of distributing them.

Example:

hive> add FILE /tmp/tt.py;

hive> list FILES;

/tmp/tt.py

hive> from networks a MAP a.networkid USING 'python tt.py' as nn where a.ds = '2009-01-04' limit 10;

It is not neccessary to add files to the session if the files used in a transform script are already available on all machines in the hadoop cluster using the same path name. For example:

* ... MAP a.networkid USING 'wc -l' ...: here wc is an executable available on all machines
* ... MAP a.networkid USING '/home/nfsserv1/hadoopscripts/tt.py' ...: here tt.py may be accessible via a nfs mount point that's configured identically on all the cluster nodes.

[**LanguageManual Select**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362043&originalId=27823704))
* [Select Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-SelectSyntax)
  + [WHERE Clause](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-WHEREClause)
  + [ALL and DISTINCT Clauses](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-ALLandDISTINCTClauses)
  + [Partition Based Queries](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-PartitionBasedQueries)
  + [HAVING Clause](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-HAVINGClause)
  + [LIMIT Clause](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-LIMITClause)
  + [REGEX Column Specification](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select#LanguageManualSelect-REGEXColumnSpecification)

**Select Syntax**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list]

[CLUSTER BY col\_list

| [DISTRIBUTE BY col\_list] [SORT BY col\_list]

]

[LIMIT number]

* A SELECT statement can be part of a [union](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Union) query or a [subquery](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries) of another query.
* table\_reference indicates the input to the query. It can be a regular table, [a view](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateView), a [join construct](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins) or a [subquery](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries).
* Simple query. For example, the following query retrieves all columns and all rows from table t1.

SELECT \* FROM t1

**WHERE Clause**

The where condition is a [boolean] [expression]. For example, the following query returns only those sales records which have an amount greater than 10 from the US region. Hive does not support IN, EXISTS or subqueries in the WHERE clause.

SELECT \* FROM sales WHERE amount > 10 AND region = "US"

**ALL and DISTINCT Clauses**

The ALL and DISTINCT options specify whether duplicate rows should be returned. If none of these options are given, the default is ALL (all matching rows are returned). DISTINCT specifies removal of duplicate rows from the result set.

hive> SELECT col1, col2 FROM t1

1 3

1 3

1 4

2 5

hive> SELECT DISTINCT col1, col2 FROM t1

1 3

1 4

2 5

hive> SELECT DISTINCT col1 FROM t1

1

2

**Partition Based Queries**

In general, a SELECT query scans the entire table (other than for [sampling](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling)). If a table created using the [PARTITIONED BY](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable) clause, a query can do **partition pruning** and scan only a fraction of the table relevant to the partitions specified by the query. Hive currently does partition pruning if the partition predicates are specified in the WHERE clause or the ON clause in a JOIN. For example, if table page\_views is partitioned on column date, the following query retrieves rows for just days between 2008-03-01 and 2008-03-31.

SELECT page\_views.\*

FROM page\_views

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

If a table page\_views is joined with another table dim\_users, you can specify a range of partitions in the ON clause as follows:

SELECT page\_views.\*

FROM page\_views JOIN dim\_users

ON (page\_views.user\_id = dim\_users.id AND page\_views.date >= '2008-03-01' AND page\_views.date <= '2008-03-31')

* See also [Group By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy)
* See also [Sort By / Cluster By / Distribute By / Order By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy)

**HAVING Clause**

Hive added support for the HAVING clause in version 0.7.0. In older versions of Hive it is possible to achieve the same effect by using a subquery, e.g:

SELECT col1 FROM t1 GROUP BY col1 HAVING SUM(col2) > 10

can also be expressed as

SELECT col1 FROM (SELECT col1, SUM(col2) AS col2sum FROM t1 GROUP BY col1) t2 WHERE t2.col2sum > 10

**LIMIT Clause**

Limit indicates the number of rows to be returned. The rows returned are chosen at random. The following query returns 5 rows from t1 at random.

SELECT \* FROM t1 LIMIT 5

* Top k queries. The following query returns the top 5 sales records wrt amount.
* SET mapred.reduce.tasks = 1
* SELECT \* FROM sales SORT BY amount DESC LIMIT 5

**REGEX Column Specification**

A SELECT statement can take regex-based column specification.

* We use java regex syntax. Try <http://www.fileformat.info/tool/regex.htm> for testing purposes.
* The following query select all columns except ds and hr.

SELECT {{(ds|hr)?+.+}} FROM sales

[**LanguageManual GroupBy**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362038&originalId=27820692))
* [Group By Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy#LanguageManualGroupBy-GroupBySyntax)
  + [Simple Examples](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy#LanguageManualGroupBy-SimpleExamples)
* [Advanced Features](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy#LanguageManualGroupBy-AdvancedFeatures)
  + [Multi-Group-By Inserts](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy#LanguageManualGroupBy-MultiGroupByInserts)
  + [Map-side Aggregation for Group By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+GroupBy#LanguageManualGroupBy-MapsideAggregationforGroupBy)

**Group By Syntax**

groupByClause: GROUP BY groupByExpression (, groupByExpression)\*

groupByExpression: expression

groupByQuery: SELECT expression (, expression)\* FROM src groupByClause?

**Simple Examples**

In order to count the number of rows in a table:

SELECT COUNT(\*) FROM table2;

Note that for versions of Hive which don't include [HIVE-287](https://issues.apache.org/jira/browse/HIVE-287), you'll need to use COUNT(1) in place of COUNThttps://cwiki.apache.org/confluence/images/icons/emoticons/star_yellow.gif.

In order to count the number of distinct users by gender one could write the following query:

INSERT OVERWRITE TABLE pv\_gender\_sum

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

FROM pv\_users

GROUP BY pv\_users.gender;

Multiple aggregations can be done at the same time, however, no two aggregations can have different DISTINCT columns .e.g while the following is possible

INSERT OVERWRITE TABLE pv\_gender\_agg

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

FROM pv\_users

GROUP BY pv\_users.gender;

Note that for versions of Hive which don't include [HIVE-287](https://issues.apache.org/jira/browse/HIVE-287), you'll need to use COUNT(1) in place of COUNThttps://cwiki.apache.org/confluence/images/icons/emoticons/star_yellow.gif.

However, the following query is not allowed. We don't allow multiple DISTINCT expressions in the same query.

INSERT OVERWRITE TABLE pv\_gender\_agg

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

FROM pv\_users

GROUP BY pv\_users.gender;

**Advanced Features**

**Multi-Group-By Inserts**

The output of the aggregations or simple selects can be further sent into multiple tables or even to hadoop dfs files (which can then be manipulated using hdfs utilitites). e.g. if along with the gender breakdown, one needed to find the breakdown of unique page views by age, one could accomplish that with the following query:

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/facebook/tmp/pv\_age\_sum'

SELECT pv\_users.age, count(DISTINCT pv\_users.userid)

GROUP BY pv\_users.age;

**Map-side Aggregation for Group By**

*hive.map.aggr* controls how we do aggregations. The default is false. If it is set to true, Hive will do the first-level aggregation directly in the map task.  
This usually provides better efficiency, but may require more memory to run successfully.

set hive.map.aggr=true;

SELECT COUNT(\*) FROM table2;

Note that for versions of Hive which don't include [HIVE-287](https://issues.apache.org/jira/browse/HIVE-287), you'll need to use COUNT(1) in place of COUNThttps://cwiki.apache.org/confluence/images/icons/emoticons/star_yellow.gif.

[**LanguageManual SortBy**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362045&originalId=27823705))
* [show comment](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy)
* [Syntax of Order By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-SyntaxofOrderBy)
* [Syntax of Sort By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-SyntaxofSortBy)
  + [Difference between Sort By and Order By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-DifferencebetweenSortByandOrderBy)
  + [Setting Types for Sort By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-SettingTypesforSortBy)
* [Syntax of Cluster By and Distribute By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-SyntaxofClusterByandDistributeBy)

**Syntax of Order By**

The *ORDER BY* syntax in Hive QL is similar to the syntax of *ORDER BY* in SQL language.

colOrder: ( ASC | DESC )

orderBy: ORDER BY colName colOrder? (',' colName colOrder?)\*

query: SELECT expression (',' expression)\* FROM src orderBy

There are some limitations in the "order by" clause. In the strict mode (i.e., hive.mapred.mode=strict), the order by clause has to be followed by a "limit" clause. The limit clause is not necessary if you set hive.mapred.mode to nonstrict. The reason is that in order to impose total order of all results, there has to be one reducer to sort the final output. If the number of rows in the output is too large, the single reducer could take a very long time to finish.

**Syntax of Sort By**

The *SORT BY* syntax is similar to the syntax of *ORDER BY* in SQL language.

colOrder: ( ASC | DESC )

sortBy: SORT BY colName colOrder? (',' colName colOrder?)\*

query: SELECT expression (',' expression)\* FROM src sortBy

Hive uses the columns in *SORT BY* to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order.

**Difference between Sort By and Order By**

Hive supports *SORT BY* which sorts the data per reducer. The difference between "order by" and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, "sort by" may give partially ordered final results.

Basically, the data in each reducer will be sorted according to the order that the user specified. The following example shows

SELECT key, value FROM src SORT BY key ASC, value DESC

The query had 2 reducers, and the output of each is:

0 5

0 3

3 6

9 1

0 4

0 3

1 1

2 5

**Setting Types for Sort By**

After a transform, variable types are generally considered to be strings, meaning that numeric data will be sorted lexicographically. To overcome this, a second SELECT statement with casts can be used before using SORT BY.

FROM (FROM (FROM src

SELECT TRANSFORM(value)

USING 'mapper'

AS value, count) mapped

SELECT cast(value as double) AS value, cast(count as int) AS count

SORT BY value, count) sorted

SELECT TRANSFORM(value, count)

USING 'reducer'

AS whatever

**Syntax of Cluster By and Distribute By**

*Cluster By* and *Distribute By* are used mainly with the [Transform/Map-Reduce Scripts](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform). But, it is sometimes useful in SELECT statements if there is a need to partition and sort the output of a query for subsequent queries.

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

For example, we are *Distributing By x* on the following 5 rows to 2 reducer:

x1

x2

x4

x3

x1

Reducer 1 got

x1

x2

x1

Reducer 2 got

x4

x3

Note that all rows with the same key x1 is guaranteed to be distributed to the same reducer (reducer 1 in this case), but they are not guaranteed to be clustered in adjacent positions.

In contrast, if we use *Cluster By x*, the two reducers will further sort rows on x:

Reducer 1 got

x1

x1

x2

Reducer 2 got

x3

x4

Instead of specifying *Cluster By*, the user can specify *Distribute By* and *Sort By*, so the partition columns and sort columns can be different. The usual case is that the partition columns are a prefix of sort columns, but that is not required.

SELECT col1, col2 FROM t1 CLUSTER BY col1

SELECT col1, col2 FROM t1 DISTRIBUTE BY col1

SELECT col1, col2 FROM t1 DISTRIBUTE BY col1 SORT BY col1 ASC, col2 DESC

FROM (

FROM pv\_users

MAP ( pv\_users.userid, pv\_users.date )

USING 'map\_script'

AS c1, c2, c3

DISTRIBUTE BY c2

SORT BY c2, c1) map\_output

INSERT OVERWRITE TABLE pv\_users\_reduced

REDUCE ( map\_output.c1, map\_output.c2, map\_output.c3 )

USING 'reduce\_script'

AS date, count;

[**LanguageManual Transform**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362047&originalId=27823714))
* [show comment](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform)
* [Transform/Map-Reduce Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform#LanguageManualTransform-Transform%2FMapReduceSyntax)
* [Schema-less Map-reduce Scripts](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform#LanguageManualTransform-SchemalessMapreduceScripts)
* [Typing the output of TRANSFORM](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform#LanguageManualTransform-TypingtheoutputofTRANSFORM)

**Transform/Map-Reduce Syntax**

Users can also plug in their own custom mappers and reducers in the data stream by using features natively supported in the Hive 2.0 language. e.g. in order to run a custom mapper script - map\_script - and a custom reducer script - reduce\_script - the user can issue the following command which uses the TRANSFORM clause to embed the mapper and the reducer scripts.

By default, columns will be transformed to *STRING* and delimited by TAB before feeding to the user script; similarly, all NULL values will be converted to the literal string **\N** in order to differentiate NULL values from empty strings. The standard output of the user script will be treated as TAB-separated *STRING* columns, any cell containing only **\N** will be re-interpreted as a NULL, and then the resulting STRING column will be cast to the data type specified in the table declaration in the usual way. User scripts can output debug information to standard error which will be shown on the task detail page on hadoop. These defaults can be overridden with *ROW FORMAT ...*.

**NOTE:** It is your responsibility to sanitize any STRING columns prior to transformation. If your STRING column contains tabs, an identity transformer will not give you back what you started with! To help with this, see [REGEXP\_REPLACE](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-StringFunctions) and replace the tabs with some other character on their way into the TRANSFORM() call.

Formally, *MAP ...* and *REDUCE ...* are syntactic transformations of *SELECT TRANSFORM ( ... )*. In other words, they serve as comments or notes to the reader of the query. BEWARE: Use of these keywords may be **dangerous** as (e.g.) typing "REDUCE" does not force a reduce phase to occur and typing "MAP" does not force a new map phase!

Please also see [Sort By / Cluster By / Distribute By](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy) and Larry Ogrodnek's [blog post](http://dev.bizo.com/2009/10/hive-map-reduce-in-java.html).

clusterBy: CLUSTER BY colName (',' colName)\*

distributeBy: DISTRIBUTE BY colName (',' colName)\*

sortBy: SORT BY colName (ASC | DESC)? (',' colName (ASC | DESC)?)\*

rowFormat

: ROW FORMAT

(DELIMITED [FIELDS TERMINATED BY char]

[COLLECTION ITEMS TERMINATED BY char]

[MAP KEYS TERMINATED BY char]

[ESCAPED BY char]

[LINES SEPARATED BY char]

|

SERDE serde\_name [WITH SERDEPROPERTIES

property\_name=property\_value,

property\_name=property\_value, ...])

outRowFormat : rowFormat

inRowFormat : rowFormat

outRecordReader : RECORDREADER className

query:

FROM (

FROM src

MAP expression (',' expression)\*

(inRowFormat)?

USING 'my\_map\_script'

( AS colName (',' colName)\* )?

(outRowFormat)? (outRecordReader)?

( clusterBy? | distributeBy? sortBy? ) src\_alias

)

REDUCE expression (',' expression)\*

(inRowFormat)?

USING 'my\_reduce\_script'

( AS colName (',' colName)\* )?

(outRowFormat)? (outRecordReader)?

FROM (

FROM src

SELECT TRANSFORM '(' expression (',' expression)\* ')'

(inRowFormat)?

USING 'my\_map\_script'

( AS colName (',' colName)\* )?

(outRowFormat)? (outRecordReader)?

( clusterBy? | distributeBy? sortBy? ) src\_alias

)

SELECT TRANSFORM '(' expression (',' expression)\* ')'

(inRowFormat)?

USING 'my\_reduce\_script'

( AS colName (',' colName)\* )?

(outRowFormat)? (outRecordReader)?

Example #1:

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;

FROM (

FROM pv\_users

SELECT TRANSFORM(pv\_users.userid, pv\_users.date)

USING 'map\_script'

AS dt, uid

CLUSTER BY dt) map\_output

INSERT OVERWRITE TABLE pv\_users\_reduced

SELECT TRANSFORM(map\_output.dt, map\_output.uid)

USING 'reduce\_script'

AS date, count;

Example #2

FROM (

FROM src

SELECT TRANSFORM(src.key, src.value) ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.TypedBytesSerDe'

USING '/bin/cat'

AS (tkey, tvalue) ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.TypedBytesSerDe'

RECORDREADER 'org.apache.hadoop.hive.ql.exec.TypedBytesRecordReader'

) tmap

INSERT OVERWRITE TABLE dest1 SELECT tkey, tvalue

**Schema-less Map-reduce Scripts**

If there is no *AS* clause after *USING my\_script*, Hive assumes that the output of the script contains 2 parts: key which is before the first tab, and value which is the rest after the first tab. Note that this is different from specifying *AS key, value* because in that case, value will only contain the portion between the first tab and the second tab if there are multiple tabs.

Note that we can directly do *CLUSTER BY key* without specifying the output schema of the scripts.

FROM (

FROM pv\_users

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

USING 'map\_script'

CLUSTER BY key) map\_output

INSERT OVERWRITE TABLE pv\_users\_reduced

REDUCE map\_output.key, map\_output.value

USING 'reduce\_script'

AS date, count;

**Typing the output of TRANSFORM**

The output fields from a script are typed as strings by default; for example in

SELECT TRANSFORM(stuff)

USING 'script'

AS thing1, thing2

They can be immediately casted with the syntax:

SELECT TRANSFORM(stuff)

USING 'script'

AS (thing1 INT, thing2 INT)

**Hive User Defined Functions**

* [Hive User Defined Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-HiveUserDefinedFunctions)
  + [Built-in Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinOperators)
    - [Relational Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-RelationalOperators)
    - [Arithmetic Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ArithmeticOperators)
    - [Logical Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-LogicalOperators)
    - [Complex Type Constructors](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ComplexTypeConstructors)
    - [Operators on Complex Types](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-OperatorsonComplexTypes)
  + [Built-in Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinFunctions)
    - [Mathematical Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-MathematicalFunctions)
    - [Collection Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-CollectionFunctions)
    - [Type Conversion Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-TypeConversionFunctions)
    - [Date Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-DateFunctions)
    - [Conditional Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ConditionalFunctions)
    - [String Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-StringFunctions)
    - [Misc. Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Misc.Functions)
      * [xpath](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-xpath)
      * [get\_json\_object](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-getjsonobject)
  + [Built-in Aggregate Functions (UDAF)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinAggregateFunctions%28UDAF%29)
  + [Built-in Table-Generating Functions (UDTF)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinTableGeneratingFunctions%28UDTF%29)
    - [explode](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-explode)
    - [json\_tuple](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-jsontuple)
  + [GROUPing and SORTing on f(column)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-GROUPingandSORTingonf%28column%29)

In the CLI, use the commands below to show the latest documentation:

SHOW FUNCTIONS;

DESCRIBE FUNCTION <function\_name>;

**Built-in Operators**

**Relational Operators**

The following operators compare the passed operands and generate a TRUE or FALSE value depending on whether the comparison between the operands holds.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A = B | All primitive types | TRUE if expression A is equal to expression B otherwise FALSE |
| A == B | None! | Fails because of invalid syntax. SQL uses =, not == |
| A <> B | All primitive types | NULL if A or B is NULL, TRUE if expression A is NOT equal to expression B otherwise FALSE |
| A < B | All primitive types | NULL if A or B is NULL, TRUE if expression A is less than expression B otherwise FALSE |
| A <= B | All primitive types | NULL if A or B is NULL, TRUE if expression A is less than or equal to expression B otherwise FALSE |
| A > B | All primitive types | NULL if A or B is NULL, TRUE if expression A is greater than expression B otherwise FALSE |
| A >= B | All primitive types | NULL if A or B is NULL, TRUE if expression A is greater than or equal to expression B otherwise FALSE |
| 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 | NULL if A or B is NULL, TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. The comparison is done character by character. The \_ character in B matches any character in A(similar to . in posix regular expressions) while the % character in B matches an arbitrary number of characters in A(similar to .\* in posix regular expressions) e.g. 'foobar' like 'foo' evaluates to FALSE where as 'foobar' like 'foo\_ \_ \_' evaluates to TRUE and so does 'foobar' like 'foo%' |
| A RLIKE B | strings | NULL if A or B is NULL, TRUE if string A matches the Java regular expression B(See Java regular expressions syntax), otherwise FALSE e.g. 'foobar' rlike 'foo' evaluates to FALSE where as 'foobar' rlike '^f.\*r$' evaluates to TRUE |
| A REGEXP B | strings | Same as RLIKE |

**Arithmetic Operators**

The following operators support various common arithmetic operations on the operands. All return number types; if any of the operands are NULL, then the result is also NULL.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A + B | All number types | Gives the result of adding A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. e.g. since every integer is a float, therefore float is a containing type of integer so the + operator on a float and an int will result in a float. |
| A - B | All number types | Gives the result of subtracting B from A. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A \* B | All number types | Gives the result of multiplying A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. Note that if the multiplication causing overflow, you will have to cast one of the operators to a type higher in the type hierarchy. |
| A / B | All number types | Gives the result of dividing B from A. The result is a double type. |
| A % B | All number types | Gives the reminder resulting from dividing A by B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A & B | All number types | Gives the result of bitwise AND of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A | B | All number types | Gives the result of bitwise OR of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A ^ B | All number types | Gives the result of bitwise XOR of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| ~A | All number types | Gives the result of bitwise NOT of A. The type of the result is the same as the type of A. |

**Logical Operators**

The following operators provide support for creating logical expressions. All of them return boolean TRUE, FALSE, or NULL depending upon the boolean values of the operands. NULL behaves as an "unknown" flag, so if the result depends on the state of an unknown, the result itself is unknown.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE. NULL if A or B is NULL |
| A && B | boolean | Same as A AND B |
| A OR B | boolean | TRUE if either A or B or both are TRUE; FALSE OR NULL is NULL; otherwise FALSE |
| A || B | boolean | Same as A OR B |
| NOT A | boolean | TRUE if A is FALSE or NULL if A is NULL. Otherwise FALSE. |
| ! A | boolean | Same as NOT A |

**Complex Type Constructors**

The following functions construct instances of complex types.

|  |  |  |
| --- | --- | --- |
| **Constructor Function** | **Operands** | **Description** |
| map | (key1, value1, key2, value2, ...) | Creates a map with the given key/value pairs |
| struct | (val1, val2, val3, ...) | Creates a struct with the given field values. Struct field names will be col1, col2, ... |
| array | (val1, val2, ...) | Creates an array with the given elements |

**Operators on Complex Types**

The following operators provide mechanisms to access elements in 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 e.g. if A is an array comprising of ['foo', 'bar'] then A[0] returns 'foo' and A[1] returns 'bar' |
| M[key] | M is a Map<K, V> and key has type K | Returns the value corresponding to the key in the map e.g. if M is a map comprising of {'f' -> 'foo', 'b' -> 'bar', 'all' -> 'foobar'} then M['all'] returns 'foobar' |
| S.x | S is a struct | Returns the x field of S. e.g for struct foobar {int foo, int bar} foobar.foo returns the integer stored in the foo field of the struct. |

**Built-in Functions**

**Mathematical Functions**

The following built-in mathematical functions are supported in hive; most return NULL when the argument(s) are NULL:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| BIGINT | round(double a) | Returns the rounded BIGINT value of the double |
| DOUBLE | round(double a, int d) | Returns the double rounded to d decimal places |
| BIGINT | floor(double a) | Returns the maximum BIGINT value that is equal or less than the double |
| BIGINT | ceil(double a), ceiling(double a) | Returns the minimum BIGINT value that is equal or greater than the double |
| double | rand(), rand(int seed) | Returns a random number (that changes from row to row) that is distributed uniformly from 0 to 1. Specifiying the seed will make sure the generated random number sequence is deterministic. |
| double | exp(double a) | Returns e^a where e is the base of the natural logarithm |
| double | ln(double a) | Returns the natural logarithm of the argument |
| double | log10(double a) | Returns the base-10 logarithm of the argument |
| double | log2(double a) | Returns the base-2 logarithm of the argument |
| double | log(double base, double a) | Return the base "base" logarithm of the argument |
| double | pow(double a, double p) power(double a, double p) | Return a^p |
| double | sqrt(double a) | Returns the square root of a |
| string | bin(BIGINT a) | Returns the number in binary format (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_bin>]) |
| string | hex(BIGINT a) hex(string a) | If the argument is an int, hex returns the number as a string in hex format. Otherwise if the number is a string, it converts each character into its hex representation and returns the resulting string. (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_hex>]) |
| string | unhex(string a) | Inverse of hex. Interprets each pair of characters as a hexidecimal number and converts to the character represented by the number. |
| string | conv(BIGINT num, int from\_base, int to\_base) | Converts a number from a given base to another (see [<http://dev.mysql.com/doc/refman/5.0/en/mathematical-functions.html#function_conv>]) |
| double | abs(double a) | Returns the absolute value |
| int double | pmod(int a, int b) pmod(double a, double b) | Returns the positive value of a mod b |
| double | sin(double a) | Returns the sine of a (a is in radians) |
| double | asin(double a) | Returns the arc sin of x if -1<=a<=1 or null otherwise |
| double | cos(double a) | Returns the cosine of a (a is in radians) |
| double | acos(double a) | Returns the arc cosine of x if -1<=a<=1 or null otherwise |
| int double | positive(int a) positive(double a) | Returns a |
| int double | negative(int a) negative(double a) | Returns -a |

**Collection Functions**

The following built-in collection functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| int | size(Map<K.V>) | Returns the number of elements in the map type |
| int | size(Array<T>) | Returns the number of elements in the array type |

**Type Conversion Functions**

The following type conversion functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| **Expected "=" to follow "type"** | cast(expr as <type>) | Converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) will convert the string '1' to it integral representation. A null is returned if the conversion does not succeed. |

**Date Functions**

The following built-in date functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| string | from\_unixtime(bigint unixtime[, string format]) | Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00" |
| bigint | unix\_timestamp() | Gets current time stamp using the default time zone. |
| bigint | unix\_timestamp(string date) | Converts time string in format yyyy-MM-dd HH:mm:ss to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20 11:30:01') = 1237573801 |
| bigint | unix\_timestamp(string date, string pattern) | Convert time string with given pattern (see [<http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>]) to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20', 'yyyy-MM-dd') = 1237532400 |
| string | to\_date(string timestamp) | Returns the date part of a timestamp string: to\_date("1970-01-01 00:00:00") = "1970-01-01" |
| int | year(string date) | Returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970 |
| int | month(string date) | Returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11 |
| int | day(string date) dayofmonth(date) | Return the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1 |
| int | hour(string date) | Returns the hour of the timestamp: hour('2009-07-30 12:58:59') = 12, hour('12:58:59') = 12 |
| int | minute(string date) | Returns the minute of the timestamp |
| int | second(string date) | Returns the second of the timestamp |
| int | weekofyear(string date) | Return the week number of a timestamp string: weekofyear("1970-11-01 00:00:00") = 44, weekofyear("1970-11-01") = 44 |
| int | datediff(string enddate, string startdate) | Return the number of days from startdate to enddate: datediff('2009-03-01', '2009-02-27') = 2 |
| int | date\_add(string startdate, int days) | Add a number of days to startdate: date\_add('2008-12-31', 1) = '2009-01-01' |
| int | date\_sub(string startdate, int days) | Subtract a number of days to startdate: date\_sub('2008-12-31', 1) = '2008-12-30' |

**Conditional Functions**

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| T | if(boolean testCondition, T valueTrue, T valueFalseOrNull) | Return valueTrue when testCondition is true, returns valueFalseOrNull otherwise |
| T | COALESCE(T v1, T v2, ...) | Return the first v that is not NULL, or NULL if all v's are NULL |
| T | CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END | When a = b, returns c; when a = d, return e; else return f |
| T | CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END | When a = true, returns b; when c = true, return d; else return e |

**String Functions**

The following are built-in String functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| int | length(string A) | Returns the length of the string |
| string | reverse(string A) | Returns the reversed string |
| string | concat(string A, string B...) | Returns the string resulting from concatenating the strings passed in as parameters in order. e.g. concat('foo', 'bar') results in 'foobar'. Note that this function can take any number of input strings. |
| string | concat\_ws(string SEP, string A, string B...) | Like concat() above, but with custom separator SEP. |
| string | substr(string A, int start) substring(string A, int start) | Returns the substring of A starting from start position till the end of string A e.g. substr('foobar', 4) results in 'bar' (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | substr(string A, int start, int len) substring(string A, int start, int len) | Returns the substring of A starting from start position with length len e.g. substr('foobar', 4, 1) results in 'b' (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | upper(string A) ucase(string A) | Returns the string resulting from converting all characters of A to upper case e.g. upper('fOoBaR') results in 'FOOBAR' |
| string | lower(string A) lcase(string A) | Returns the string resulting from converting all characters of B to lower case e.g. lower('fOoBaR') results in 'foobar' |
| string | trim(string A) | Returns the string resulting from trimming spaces from both ends of A e.g. trim(' foobar ') results in 'foobar' |
| string | ltrim(string A) | Returns the string resulting from trimming spaces from the beginning(left hand side) of A e.g. ltrim(' foobar ') results in 'foobar ' |
| string | rtrim(string A) | Returns the string resulting from trimming spaces from the end(right hand side) of A e.g. rtrim(' foobar ') results in ' foobar' |
| string | regexp\_replace(string A, string B, string C) | Returns the string resulting from replacing all substrings in B that match the Java regular expression syntax(See Java regular expressions syntax) with C e.g. regexp\_replace("foobar", "oo|ar", "") returns 'fb.' 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. |
| string | regexp\_extract(string subject, string pattern, int index) | Returns the string extracted using the pattern. e.g. regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) returns 'bar.' 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. See docs/api/java/util/regex/Matcher.html for more information on the 'index' or Java regex group() method. |
| string | parse\_url(string urlString, string partToExtract [, string keyToExtract]) | Returns the specified part from the URL. Valid values for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO. e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') returns 'facebook.com'. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') returns 'v1'. |
| string | get\_json\_object(string json\_string, string path) | Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid. **NOTE: The json path can only have the characters [0-9a-z\_], i.e., no upper-case or special characters. Also, the keys \*cannot** start with numbers.\* This is due to restrictions on Hive column names. |
| string | space(int n) | Return a string of n spaces |
| string | repeat(string str, int n) | Repeat str n times |
| int | ascii(string str) | Returns the numeric value of the first character of str |
| string | lpad(string str, int len, string pad) | Returns str, left-padded with pad to a length of len |
| string | rpad(string str, int len, string pad) | Returns str, right-padded with pad to a length of len |
| array | split(string str, string pat) | Split str around pat (pat is a regular expression) |
| int | find\_in\_set(string str, string strList) | Returns the first occurance of str in strList where strList is a comma-delimited string. Returns null if either argument is null. Returns 0 if the first argument contains any commas. e.g. find\_in\_set('ab', 'abc,b,ab,c,def') returns 3 |
| array<array<string>> | sentences(string str, string lang, string locale) | Tokenizes a string of natural language text into words and sentences, where each sentence is broken at the appropriate sentence boundary and returned as an array of words. The 'lang' and 'locale' are optional arguments. e.g. sentences('Hello there! How are you?') returns ( ("Hello", "there"), ("How", "are", "you") ) |
| array<struct<string,double>> | ngrams(array<array<string>>, int N, int K, int pf) | Returns the top-k N-grams from a set of tokenized sentences, such as those returned by the sentences() UDAF. See [StatisticsAndDataMining](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining) for more information. |
| array<struct<string,double>> | context\_ngrams(array<array<string>>, array<string>, int K, int pf) | Returns the top-k contextual N-grams from a set of tokenized sentences, given a string of "context". See [StatisticsAndDataMining](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining) for more information. |

**Misc. Functions**

**xpath**

The following functions are described in [LanguageManual XPathUDF](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+XPathUDF):

* xpath, xpath\_short, xpath\_int, xpath\_long, xpath\_float, xpath\_double, xpath\_number, xpath\_string

**get\_json\_object**

A limited version of JSONPath is supported:

* $ : Root object
* . : Child operator
* [] : Subscript operator for array
* \* : Wildcard for []

Syntax not supported that's worth noticing:

* : Zero length string as key
* .. : Recursive descent
* @ : Current object/element
* () : Script expression
* ?() : Filter (script) expression.
* [,] : Union operator
* [start:end.step] : array slice operator

Example: src\_json table is a single column (json), single row table:

+----+

json

+----+

{"store":

{"fruit":\[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],

"bicycle":{"price":19.95,"color":"red"}

},

"email":"amy@only\_for\_json\_udf\_test.net",

"owner":"amy"

}

+----+

The fields of the json object can be extracted using these queries:

hive> SELECT get\_json\_object(src\_json.json, '$.owner') FROM src\_json;

amy

hive> SELECT get\_json\_object(src\_json.json, '$.store.fruit\[0]') FROM src\_json;

{"weight":8,"type":"apple"}

hive> SELECT get\_json\_object(src\_json.json, '$.non\_exist\_key') FROM src\_json;

NULL

**Built-in Aggregate Functions (UDAF)**

The following are built-in aggregate functions are supported in Hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| bigint | count(\*), count(expr), count(DISTINCT expr[, expr\_.]) | count(\*) - Returns the total number of retrieved rows, including rows containing NULL values; count(expr) - Returns the number of rows for which the supplied expression is non-NULL; count(DISTINCT expr[, expr]) - Returns the number of rows for which the supplied expression(s) are unique and non-NULL. |
| double | sum(col), sum(DISTINCT col) | Returns the sum of the elements in the group or the sum of the distinct values of the column in the group |
| double | avg(col), avg(DISTINCT col) | Returns the average of the elements in the group or the average of the distinct values of the column in the group |
| double | min(col) | Returns the minimum of the column in the group |
| double | max(col) | Returns the maximum value of the column in the group |
| double | var\_pop(col) | Returns the variance of a numeric column in the group |
| double | var\_samp(col) | Returns the unbiased sample variance of a numeric column in the group |
| double | stddev\_pop(col) | Returns the standard deviation of a numeric column in the group |
| double | stddev\_samp(col) | Returns the unbiased sample standard deviation of a numeric column in the group |
| double | covar\_pop(col1, col2) | Returns the population covariance of a pair of numeric columns in the group |
| double | covar\_samp(col1, col2) | Returns the sample covariance of a pair of a numeric columns in the group |
| double | corr(col1, col2) | Returns the Pearson coefficient of correlation of a pair of a numeric columns in the group |
| double | percentile(col, p) | Returns the exact p^th^ percentile of an integer column in the group (does not work with floating point types). p must be between 0 and 1. |
| array<double> | percentile(col, array(p~1,, [, p,,2,,]...)) | Returns the exact percentiles p,,1,,, p,,2,,, ... of an integer column in the group (does not work with floating point types). p,,i~ must be between 0 and 1. |
| double | percentile\_approx(col, p [, B]) | Returns an approximate p^th^ percentile of a numeric column (including floating point types) in the group. The B parameter controls approximation accuracy at the cost of memory. Higher values yield better approximations, and the default is 10,000. When the number of distinct values in col is smaller than B, this gives an exact percentile value. |
| array<double> | percentile\_approx(col, array(p~1,, [, p,,2\_]...) [, B]) | Same as above, but accepts and returns an array of percentile values instead of a single one. |
| array<struct {'x','y'}> | histogram\_numeric(col, b) | Computes a histogram of a numeric column in the group using b non-uniformly spaced bins. The output is an array of size b of double-valued (x,y) coordinates that represent the bin centers and heights |
| array | collect\_set(col) | Returns a set of objects with duplicate elements eliminated |

**Built-in Table-Generating Functions (UDTF)**

Normal user-defined functions, such as concat(), take in a single input row and output a single output row. In contrast, table-generating functions transform a single input row to multiple output rows.

**explode**

explode() takes in an array as an input and outputs the elements of the array as separate rows. UDTF's can be used in the SELECT expression list and as a part of LATERAL VIEW.

An example use of explode() in the SELECT expression list is as follows:

Consider a table named myTable that has a single column (myCol) and two rows:

|  |
| --- |
| **Array<int> myCol** |
| [1,2,3] |
| [4,5,6] |

Then running the query:

SELECT explode(myCol) AS myNewCol FROM myTable;

Will produce:

|  |
| --- |
| **(int) myNewCol** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

Using the syntax "SELECT udtf(col) AS colAlias..." has a few limitations:

* No other expressions are allowed in SELECT
  + SELECT pageid, explode(adid\_list) AS myCol... is not supported
* UDTF's can't be nested
  + SELECT explode(explode(adid\_list)) AS myCol... is not supported
* GROUP BY / CLUSTER BY / DISTRIBUTE BY / SORT BY is not supported
  + SELECT explode(adid\_list) AS myCol ... GROUP BY myCol is not supported

Please see [LanguageManual LateralView](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView) for an alternative syntax that does not have these limitations.

The following are built-in table-generating functions are supported in Hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| Array Type | explode(array<*TYPE*> a) | For each element in a, explode() generates a row containing that element |

**json\_tuple**

A new json\_tuple() UDTF is introduced in hive 0.7. It takes a set of names (keys) and return a tuple of values in one function.  
If you are using get\_json\_object() and want to replace it with json\_tuple, the only changes is that your query will be using json\_tuple() in lateral view rather than multiple get\_json\_object() in the select clause.

For example,

select a.timestamp, get\_json\_object(a.appevents, '$.eventid'), get\_json\_object(a.appenvets, '$.eventname') from log a;

should be changed to

select a.timestamp, b.\*

from log a lateral view json\_tuple(a.appevent, 'eventid', 'eventname') b as f1, f2;

**GROUPing and SORTing on f(column)**

A typical OLAP pattern is that you have a timestamp column and you want to group by daily or other less granular date windows than by second. So you might want to select concat(year(dt),month(dt)) and then group on that concat(). But if you attempt to GROUP BY or SORT BY a column on which you've applied a function and alias, like this:

select f(col) as fc, count(\*) from table\_name group by fc

You will get an error:

FAILED: Error in semantic analysis: line 1:69 Invalid Table Alias or Column Reference fc

Because you are not able to GROUP BY or SORT BY a column alias on which a function has been applied. There are two workarounds. First, you can reformulate this query with subqueries, which is somewhat complicated:

select sq.fc,col1,col2,...,colN,count(\*) from

(select f(col) as fc,col1,col2,...,colN from table\_name) sq

group by sq.fc,col1,col2,...,colN

Or you can make sure not to use a column alias, which is simpler:

select f(col) as fc, count(\*) from table\_name group by f(col)

Contact Tim Ellis (tellis) at RiotGames dot com if you would like to discuss this in further detail.

[**LanguageManual XPathUDF**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+XPathUDF)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362051&originalId=27820705))

Documentation for Built-In User-Defined Functions Related To XPath

**UDFs**

**xpath, xpath\_short, xpath\_int, xpath\_long, xpath\_float, xpath\_double, xpath\_number, xpath\_string**

* Functions for parsing XML data using XPath expressions.
* Since version: 0.6.0

**Overview**

The *xpath* family of UDFs are wrappers around the Java XPath library javax.xml.xpath provided by the JDK. The library is based on the XPath 1.0 specification. Please refer to <http://java.sun.com/javase/6/docs/api/javax/xml/xpath/package-summary.html> for detailed information on the Java XPath library.

All functions follow the form: xpath\_\*(xml\_string, xpath\_expression\_string). The XPath expression string is compiled and cached. It is reused if the expression in the next input row matches the previous. Otherwise, it is recompiled. So, the xml string is always parsed for every input row, but the xpath expression is precompiled and reused for the vast majority of use cases.

Backward axes are supported. For example:

> select xpath ('<a><b id="1"><c/></b><b id="2"><c/></b></a>','/descendant::c/ancestor::b/@id') from t1 limit 1 ;

[1","2]

Each function returns a specific Hive type given the XPath expression:

* xpath returns a Hive array of strings.
* xpath\_string returns a string.
* xpath\_boolean returns a boolean.
* xpath\_short returns a short integer.
* xpath\_int returns an integer.
* xpath\_long returns a long integer.
* xpath\_float returns a floating point number.
* xpath\_double,xpath\_number returns a double-precision floating point number (xpath\_number is an alias for xpath\_double).

The UDFs are schema agnostic - no XML validation is performed. However, malformed xml (e.g., <a><b>1</b></aa>) will result in a runtime exception being thrown.

Following are specifics on each xpath UDF variant.

**xpath**

The xpath() function always returns a hive array of strings. If the expression results in a non-text value (e.g., another xml node) the function will return an empty array. There are 2 primary uses for this function: to get a list of node text values or to get a list of attribute values.

Examples:

Non-matching XPath expression:

> select xpath('<a><b>b1</b><b>b2</b></a>','a/\*') from src limit 1 ;

[]

Get a list of node text values:

> select xpath('<a><b>b1</b><b>b2</b></a>','a/\*/text()') from src limit 1 ;

[b1","b2]

Get a list of values for attribute 'id':

> select xpath('<a><b id="foo">b1</b><b id="bar">b2</b></a>','//@id') from src limit 1 ;

[foo","bar]

Get a list of node texts for nodes where the 'class' attribute equals 'bb':

> SELECT xpath ('<a><b class="bb">b1</b><b>b2</b><b>b3</b><c class="bb">c1</c><c>c2</c></a>', 'a/\*[@class="bb"]/text()') FROM src LIMIT 1 ;

[b1","c1]

**xpath\_string**

The xpath\_string() function returns the text of the first matching node.

Get the text for node 'a/b':

> SELECT xpath\_string ('<a><b>bb</b><c>cc</c></a>', 'a/b') FROM src LIMIT 1 ;

bb

Get the text for node 'a'. Because 'a' has children nodes with text, the result is a composite of text from the children.

> SELECT xpath\_string ('<a><b>bb</b><c>cc</c></a>', 'a') FROM src LIMIT 1 ;

bbcc

Non-matching expression returns an empty string:

> SELECT xpath\_string ('<a><b>bb</b><c>cc</c></a>', 'a/d') FROM src LIMIT 1 ;

Gets the text of the first node that matches '//b':

> SELECT xpath\_string ('<a><b>b1</b><b>b2</b></a>', '//b') FROM src LIMIT 1 ;

b1

Gets the second matching node:

> SELECT xpath\_string ('<a><b>b1</b><b>b2</b></a>', 'a/b[2]') FROM src LIMIT 1 ;

b2

Gets the text from the first node that has an attribute 'id' with value 'b\_2':

> SELECT xpath\_string ('<a><b>b1</b><b id="b\_2">b2</b></a>', 'a/b[@id="b\_2"]') FROM src LIMIT 1 ;

b2

**xpath\_boolean**

Returns true if the XPath expression evaluates to true, or if a matching node is found.

Match found:

> SELECT xpath\_boolean ('<a><b>b</b></a>', 'a/b') FROM src LIMIT 1 ;

true

No match found:

> SELECT xpath\_boolean ('<a><b>b</b></a>', 'a/c') FROM src LIMIT 1 ;

false

Match found:

> SELECT xpath\_boolean ('<a><b>b</b></a>', 'a/b = "b"') FROM src LIMIT 1 ;

true

No match found:

> SELECT xpath\_boolean ('<a><b>10</b></a>', 'a/b < 10') FROM src LIMIT 1 ;

false

**xpath\_short, xpath\_int, xpath\_long**

These functions return an integer numeric value, or the value zero if no match is found, or a match is found but the value is non-numeric.  
Mathematical operations are supported. In cases where the value overflows the return type, then the maximum value for the type is returned.

No match:

> SELECT xpath\_int ('<a>b</a>', 'a = 10') FROM src LIMIT 1 ;

0

Non-numeric match:

> SELECT xpath\_int ('<a>this is not a number</a>', 'a') FROM src LIMIT 1 ;

0

> SELECT xpath\_int ('<a>this 2 is not a number</a>', 'a') FROM src LIMIT 1 ;

0

Adding values:

> SELECT xpath\_int ('<a><b class="odd">1</b><b class="even">2</b><b class="odd">4</b><c>8</c></a>', 'sum(a/\*)') FROM src LIMIT 1 ;

15

> SELECT xpath\_int ('<a><b class="odd">1</b><b class="even">2</b><b class="odd">4</b><c>8</c></a>', 'sum(a/b)') FROM src LIMIT 1 ;

7

> SELECT xpath\_int ('<a><b class="odd">1</b><b class="even">2</b><b class="odd">4</b><c>8</c></a>', 'sum(a/b[@class="odd"])') FROM src LIMIT 1 ;

5

Overflow:

> SELECT xpath\_int ('<a><b>2000000000</b><c>40000000000</c></a>', 'a/b \* a/c') FROM src LIMIT 1 ;

2147483647

**xpath\_float, xpath\_double, xpath\_number**

Similar to xpath\_short, xpath\_int and xpath\_long but with floating point semantics. Non-matches result in zero. However,  
non-numeric matches result in NaN. Note that xpath\_number() is an alias for xpath\_double().

No match:

> SELECT xpath\_double ('<a>b</a>', 'a = 10') FROM src LIMIT 1 ;

0.0

Non-numeric match:

> SELECT xpath\_double ('<a>this is not a number</a>', 'a') FROM src LIMIT 1 ;

NaN

A very large number:

SELECT xpath\_double ('<a><b>2000000000</b><c>40000000000</c></a>', 'a/b \* a/c') FROM src LIMIT 1 ;

8.0E19

[**LanguageManual Joins**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ran Ari-Gur](https://cwiki.apache.org/confluence/display/~ruakh) on Jul 18, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362039&originalId=27821981))
* [show comment](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins)
* [Join Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins#LanguageManualJoins-JoinSyntax)

**Join Syntax**

Hive supports the following syntax for joining tables:

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

table\_reference:

table\_factor

| join\_table

table\_factor:

tbl\_name [alias]

| table\_subquery alias

| ( table\_references )

join\_condition:

ON equality\_expression ( AND equality\_expression )\*

equality\_expression:

expression = expression

Only equality joins, outer joins, and left semi joins are supported in Hive. Hive does not support join conditions that are not equality conditions as it is very difficult to express such conditions as a map/reduce job. Also, more than two tables can be joined in Hive.

Some salient points to consider when writing join queries are as follows:

* Only equality joins are allowed e.g.
* SELECT a.\* FROM a JOIN b ON (a.id = b.id)
* SELECT a.\* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)

are both valid joins, however

SELECT a.\* FROM a JOIN b ON (a.id <> b.id)

is NOT allowed

* More than 2 tables can be joined in the same query e.g.
* SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

is a valid join.

* Hive converts joins over multiple tables into a single map/reduce job if for every table the same column is used in the join clauses e.g.
* SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

is converted into a single map/reduce job as only key1 column for b is involved in the join. On the other hand

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

is converted into two map/reduce jobs because key1 column from b is used in the first join condition and key2 column from b is used in the second one. The first map/reduce job joins a with b and the results are then joined with c in the second map/reduce job.

* In every map/reduce stage of the join, the last table in the sequence is streamed through the reducers where as the others are buffered. Therefore, it helps to reduce the memory needed in the reducer for buffering the rows for a particular value of the join key by organizing the tables such that the largest tables appear last in the sequence. e.g. in
* SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

all the three tables are joined in a single map/reduce job and the values for a particular value of the key for tables a and b are buffered in the memory in the reducers. Then for each row retrieved from c, the join is computed with the buffered rows. Similarly for

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

there are two map/reduce jobs involved in computing the join. The first of these joins a with b and buffers the values of a while streaming the values of b in the reducers. The second of one of these jobs buffers the results of the first join while streaming the values of c through the reducers.

* In every map/reduce stage of the join, the table to be streamed can be specified via a hint. e.g. in
* SELECT /\*+ STREAMTABLE(a) \*/ a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

all the three tables are joined in a single map/reduce job and the values for a particular value of the key for tables b and c are buffered in the memory in the reducers. Then for each row retrieved from a, the join is computed with the buffered rows.

* LEFT, RIGHT, and FULL OUTER joins exist in order to provide more control over ON clauses for which there is no match. For example, this query:
* SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

will return a row for every row in a. This output row will be a.val,b.val when there is a b.key that equals a.key, and the output row will be a.val,NULL when there is no corresponding b.key. Rows from b which have no corresponding a.key will be dropped. The syntax "FROM a LEFT OUTER JOIN b" must be written on one line in order to understand how it works--a is to the LEFT of b in this query, and so all rows from a are kept; a RIGHT OUTER JOIN will keep all rows from b, and a FULL OUTER JOIN will keep all rows from a and all rows from b. OUTER JOIN semantics should conform to standard SQL specs.

* Joins occur BEFORE WHERE CLAUSES. So, if you want to restrict the OUTPUT of a join, a requirement should be in the WHERE clause, otherwise it should be in the JOIN clause. A big point of confusion for this issue is partitioned tables:
* SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)
* WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'

will join a on b, producing a list of a.val and b.val. The WHERE clause, however, can also reference other columns of a and b that are in the output of the join, and then filter them out. However, whenever a row from the JOIN has found a key for a and no key for b, all of the columns of b will be NULL, **including the ds column**. This is to say, you will filter out all rows of join output for which there was no valid b.key, and thus you have outsmarted your LEFT OUTER requirement. In other words, the LEFT OUTER part of the join is irrelevant if you reference any column of b in the WHERE clause. Instead, when OUTER JOINing, use this syntax:

SELECT a.val, b.val FROM a LEFT OUTER JOIN b

ON (a.key=b.key AND b.ds='2009-07-07' AND a.ds='2009-07-07')

..the result is that the output of the join is pre-filtered, and you won't get post-filtering trouble for rows that have a valid a.key but no matching b.key. The same logic applies to RIGHT and FULL joins.

* Joins are NOT commutative! Joins are left-associative regardless of whether they are LEFT or RIGHT joins.
* SELECT a.val1, a.val2, b.val, c.val
* FROM a
* JOIN b ON (a.key = b.key)
* LEFT OUTER JOIN c ON (a.key = c.key)

...first joins a on b, throwing away everything in a or b that does not have a corresponding key in the other table. The reduced table is then joined on c. This provides unintuitive results if there is a key that exists in both a and c but not b: The whole row (including a.val1, a.val2, and a.key) is dropped in the "a JOIN b" step because it is not in b. The result does not have a.key in it, so when it is LEFT OUTER JOINed with c, c.val does not make it in because there is no c.key that matches an a.key (because that row from a was removed). Similarly, if this were a RIGHT OUTER JOIN (instead of LEFT), we would end up with an even weirder effect: NULL, NULL, NULL, c.val, because even though we specified a.key=c.key as the join key, we dropped all rows of a that did not match the first JOIN.  
To achieve the more intuitive effect, we should instead do FROM c LEFT OUTER JOIN a ON (c.key = a.key) LEFT OUTER JOIN b ON (c.key = b.key).

* LEFT SEMI JOIN implements the correlated IN/EXISTS subquery semantics in an efficient way. Since Hive currently does not support IN/EXISTS subqueries, you can rewrite your queries using LEFT SEMI JOIN. The restrictions of using LEFT SEMI JOIN is that the right-hand-side table should only be referenced in the join condition (ON-clause), but not in WHERE- or SELECT-clauses etc.
* SELECT a.key, a.value
* FROM a
* WHERE a.key in
* (SELECT b.key
* FROM B);

can be rewritten to:

SELECT a.key, a.val

FROM a LEFT SEMI JOIN b on (a.key = b.key)

* If all but one of the tables being joined are small, the join can be performed as a map only job. The query
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value
* FROM a join b on a.key = b.key

does not need a reducer. For every mapper of A, B is read completely. The restriction is that **a FULL/RIGHT OUTER JOIN b** cannot be performed

* If the tables being joined are bucketized, and the buckets are a multiple of each other, the buckets can be joined with each other. If table A has 8 buckets are table B has 4 buckets, the following join
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value
* FROM a join b on a.key = b.key

can be done on the mapper only. Instead of fetching B completely for each mapper of A, only the required buckets are fetched. For the query above, the mapper processing bucket 1 for A will only fetch bucket 1 of B. It is not the default behavior, and is governed by the following parameter

set hive.optimize.bucketmapjoin = true

* If the tables being joined are sorted and bucketized, and the number of buckets are same, a sort-merge join can be performed. The corresponding buckets are joined with each other at the mapper. If both A and B have 4 buckets,
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value
* FROM A a join B b on a.key = b.key

can be done on the mapper only. The mapper for the bucket for A will traverse the corresponding bucket for B. This is not the default behavior, and the following parameters need to be set:

set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;

set hive.optimize.bucketmapjoin = true;

set hive.optimize.bucketmapjoin.sortedmerge = true;

[**LanguageManual LateralView**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362040&originalId=27823711))
* [show comment](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView)
* [Lateral View Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView#LanguageManualLateralView-LateralViewSyntax)
* [Description](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView#LanguageManualLateralView-Description)
* [Important note](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView#LanguageManualLateralView-Importantnote)
* [Example](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView#LanguageManualLateralView-Example)
* [Multiple Lateral Views](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView#LanguageManualLateralView-MultipleLateralViews)

**Lateral View Syntax**

lateralView: LATERAL VIEW udtf(expression) tableAlias AS columnAlias (',' columnAlias)\*

fromClause: FROM baseTable (lateralView)\*

**Description**

Lateral view is used in conjunction with user-defined table generating functions such as explode(). As mentioned in [Built-in Table-Generating Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinTableGeneratingFunctions%28UDTF%29), a UDTF generates one or more output rows for each input row. A lateral view first applies the UDTF to each row of base table and then joins resulting output rows to the input rows to form a virtual table having the supplied table alias.

**Important note**

Currently, lateral view does not support the predicate push-down optimization. If you use a WHERE clause, your query may not compile. Look for the fix to come out at <https://issues.apache.org/jira/browse/HIVE-1056>

Until then, try adding set hive.optimize.ppd=false; before your query.

**Example**

Consider the following base table named pageAds. It has two columns: pageid (name of the page) and adid\_list (an array of ads appearing on the page):

|  |  |
| --- | --- |
| string pageid | Array<int> adid\_list |
| "front\_page" | [1, 2, 3] |
| "contact\_page" | [3, 4, 5] |

and the user would like to count the total number of times an ad appears across all pages.

A lateral view with [explode()](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-explode) can be used to convert adid\_list into separate rows using the query:

SELECT pageid, adid

FROM pageAds LATERAL VIEW explode(adid\_list) adTable AS adid;

The resulting output will be

|  |  |
| --- | --- |
| string pageid | int adid |
| "front\_page" | 1 |
| "front\_page" | 2 |
| "front\_page" | 3 |
| "contact\_page" | 3 |
| "contact\_page" | 4 |
| "contact\_page" | 5 |

Then in order to count the number of times a particular ad appears, count/group by can be used:

SELECT adid, count(1)

FROM pageAds LATERAL VIEW explode(adid\_list) adTable AS adid

GROUP BY adid;

|  |  |
| --- | --- |
| int adid | count(1) |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |

**Multiple Lateral Views**

A FROM clause can have multiple LATERAL VIEW clauses. Subsequent LATERAL VIEWS can reference columns from any of the tables appearing to the left of the LATERAL VIEW.

For example, the following could be a valid query:

SELECT \* FROM exampleTable

LATERAL VIEW explode(col1) myTable1 AS myCol1

LATERAL VIEW explode(myCol1) myTable2 AS myCol2;

LATERAL VIEW clauses are applied in the order that they appear. For example with the following base table:

|  |  |
| --- | --- |
| Array<int> col1 | Array<string> col2 |
| [1, 2] | [a", "b", "c"] |
| [3, 4] | [d", "e", "f"] |

The query:

SELECT myCol1, col2 FROM baseTable

LATERAL VIEW explode(col1) myTable1 AS myCol1;

Will produce:

|  |  |
| --- | --- |
| int mycol1 | Array<string> col2 |
| 1 | [a", "b", "c"] |
| 2 | [a", "b", "c"] |
| 3 | [d", "e", "f"] |
| 4 | [d", "e", "f"] |

A query that adds an additional LATERAL VIEW:

SELECT myCol1, myCol2 FROM baseTable

LATERAL VIEW explode(col1) myTable1 AS myCol1

LATERAL VIEW explode(col2) myTable2 AS myCol2;

Will produce:

|  |  |
| --- | --- |
| int myCol1 | string myCol2 |
| 1 | "a" |
| 1 | "b" |
| 1 | "c" |
| 2 | "a" |
| 2 | "b" |
| 2 | "c" |
| 3 | "d" |
| 3 | "e" |
| 3 | "f" |
| 4 | "d" |
| 4 | "e" |
| 4 | "f" |

[**LanguageManual Union**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Union)

**Union Syntax**

select\_statement UNION ALL select\_statement UNION ALL select\_statement ...

UNION is used to combine the result from multiple SELECT statements into a single result set. We currently only support UNION ALL (bag union) i.e. duplicates are not eliminated. The number and names of columns returned by each select\_statement has to be the same. Otherwise, a schema error is thrown.

If some additional processing has to be done on the result of the UNION, the entire statement expression can be embedded in a FROM clause like below:

SELECT \*

FROM (

select\_statement

UNION ALL

select\_statement

) unionResult

For example, if we suppose there are two different tables that track which user has published a video and which user has published a comment, the following query joins the results of a union all with the user table to create a single annotated stream for all the video publishing and comment publishing events:

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)

[**LanguageManual SubQueries**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362044&originalId=27820700))
* [Subquery Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SubQueries#LanguageManualSubQueries-SubquerySyntax)

**Subquery Syntax**

SELECT ... FROM (subquery) name ...

Hive supports subqueries only in the FROM clause. The subquery has to be given a name because every table in a FROM clause must have a name. Columns in the subquery select list must have unique names. The columns in the subquery select list are available in the outer query just like columns of a table. The subquery can also be a query expression with UNION. Hive supports arbitrary levels of sub-queries.

Example with simple subquery:

SELECT col

FROM (

SELECT a+b AS col

FROM t1

) t2

Example with subquery containing a UNION ALL:

SELECT t3.col

FROM (

SELECT a+b AS col

FROM t1

UNION ALL

SELECT c+d AS col

FROM t2

) t3

[**LanguageManual Sampling**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362042&originalId=27820697))
* [Sampling Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling#LanguageManualSampling-SamplingSyntax)
  + [Sampling Bucketized Table](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling#LanguageManualSampling-SamplingBucketizedTable)
  + [Block Sampling](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling#LanguageManualSampling-BlockSampling)

**Sampling Syntax**

**Sampling Bucketized Table**

table\_sample: TABLESAMPLE (BUCKET x OUT OF y [ON colname])

The TABLESAMPLE clause allows the users to write queries for samples of the data instead of the whole table. The TABLESAMPLE clause can be added to any table in the FROM clause. The buckets are numbered starting from 1. **colname** indicates the column on which to sample each row in the table. colname can be one of the non-partition columns in the table or **rand()** indicating sampling on the entire row instead of an individual column. The rows of the table are 'bucketed' on the colname randomly into y buckets numbered 1 through y. Rows which belong to bucket x are returned.

In the following example the 3rd bucket out of the 32 buckets of the table source. 's' is the table alias.

SELECT \*

FROM source TABLESAMPLE(BUCKET 3 OUT OF 32 ON rand()) s;

**Input pruning**: Typically, TABLESAMPLE will scan the entire table and fetch the sample. But, that is not very efficient. Instead, the table can be created with a CLUSTERED BY clause which indicates the set of columns on which the table is hash-partitioned/clustered on. If the columns specified in the TABLESAMPLE clause match the columns in the CLUSTERED BY clause, TABLESAMPLE scans only the required hash-partitions of the table.

Example:

So in the above example, if table 'source' was created with 'CLUSTERED BY id INTO 32 BUCKETS'

TABLESAMPLE(BUCKET 3 OUT OF 16 ON id)

would pick out the 3rd and 19th clusters as each bucket would be composed of (32/16)=2 clusters.

On the other hand the tablesample clause

TABLESAMPLE(BUCKET 3 OUT OF 64 ON id)

would pick out half of the 3rd cluster as each bucket would be composed of (32/64)=1/2 of a cluster.

**Block Sampling**

It is a feature that is still on trunk and is not yet in any release version.

block\_sample: TABLESAMPLE (n PERCENT)

This will allow Hive to pick up at least n% data size (notice it doesn't necessarily mean number of rows) as inputs. Only CombineHiveInputFormat is supported and some special compression formats are not handled. If we fail to sample it, the input of MapReduce job will be the whole table/partition. We do it in HDFS block level so that the sampling granularity is block size. For example, if block size is 256MB, even if n% of input size is only 100MB, you get 256MB of data.

In the following example the input size 0.1% or more will be used for the query.

SELECT \*

FROM source TABLESAMPLE(0.1 PERCENT) s;

Sometimes you want to sample the same data with different blocks, you can change this seed number:

set hive.sample.seednumber=<INTEGER>;

[**LanguageManual Explain**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Explain)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362037&originalId=27820691))
* [EXPLAIN Syntax](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Explain#LanguageManualExplain-EXPLAINSyntax)

**EXPLAIN Syntax**

Hive provides an EXPLAIN command that shows the execution plan for a query. The syntax for this statement is as follows:

EXPLAIN [EXTENDED] query

The use of EXTENDED in the EXPLAIN statement produces extra information about the operators in the plan. This is typically physical information like file names.

A Hive query gets converted into a sequence (it is more an Directed Acyclic Graph) of stages. These stages may be map/reduce stages or they may even be stages that do metastore or file system operations like move and rename. The explain output comprises of three parts:

* The Abstract Syntax Tree for the query
* The dependencies between the different stages of the plan
* The description of each of the stages

The description of the stages itself shows a sequence of operators with the metadata associated with the operators. The metadata may comprise of things like filter expressions for the FilterOperator or the select expressions for the SelectOperator or the output file names for the FileSinkOperator.

As an example, consider the following EXPLAIN query:

EXPLAIN

FROM src INSERT OVERWRITE TABLE dest\_g1 SELECT src.key, sum(substr(src.value,4)) GROUP BY src.key;

The output of this statement contains the following parts:

* The Abstract Syntax Tree
* ABSTRACT SYNTAX TREE:
* (TOK\_QUERY (TOK\_FROM (TOK\_TABREF src)) (TOK\_INSERT (TOK\_DESTINATION (TOK\_TAB dest\_g1)) (TOK\_SELECT (TOK\_SELEXPR (TOK\_COLREF src key)) (TOK\_SELEXPR (TOK\_FUNCTION sum (TOK\_FUNCTION substr (TOK\_COLREF src value) 4)))) (TOK\_GROUPBY (TOK\_COLREF src key))))
* The Dependency Graph
* STAGE DEPENDENCIES:
* Stage-1 is a root stage
* Stage-2 depends on stages: Stage-1
* Stage-0 depends on stages: Stage-2

This shows that Stage-1 is the root stage, Stage-2 is executed after Stage-1 is done and Stage-0 is executed after Stage-2 is done.

* The plans of each Stage
* STAGE PLANS:
* Stage: Stage-1
* Map Reduce
* Alias -> Map Operator Tree:
* src
* Reduce Output Operator
* key expressions:
* expr: key
* type: string
* sort order: +
* Map-reduce partition columns:
* expr: rand()
* type: double
* tag: -1
* value expressions:
* expr: substr(value, 4)
* type: string
* Reduce Operator Tree:
* Group By Operator
* aggregations:
* expr: sum(UDFToDouble(VALUE.0))
* keys:
* expr: KEY.0
* type: string
* mode: partial1
* File Output Operator
* compressed: false
* table:
* input format: org.apache.hadoop.mapred.SequenceFileInputFormat
* output format: org.apache.hadoop.mapred.SequenceFileOutputFormat
* name: binary\_table
* Stage: Stage-2
* Map Reduce
* Alias -> Map Operator Tree:
* /tmp/hive-zshao/67494501/106593589.10001
* Reduce Output Operator
* key expressions:
* expr: 0
* type: string
* sort order: +
* Map-reduce partition columns:
* expr: 0
* type: string
* tag: -1
* value expressions:
* expr: 1
* type: double
* Reduce Operator Tree:
* Group By Operator
* aggregations:
* expr: sum(VALUE.0)
* keys:
* expr: KEY.0
* type: string
* mode: final
* Select Operator
* expressions:
* expr: 0
* type: string
* expr: 1
* type: double
* Select Operator
* expressions:
* expr: UDFToInteger(0)
* type: int
* expr: 1
* type: double
* File Output Operator
* compressed: false
* table:
* input format: org.apache.hadoop.mapred.TextInputFormat
* output format: org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat
* serde: org.apache.hadoop.hive.serde2.dynamic\_type.DynamicSerDe
* name: dest\_g1
* Stage: Stage-0
* Move Operator
* tables:
* replace: true
* table:
* input format: org.apache.hadoop.mapred.TextInputFormat
* output format: org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat
* serde: org.apache.hadoop.hive.serde2.dynamic\_type.DynamicSerDe
* name: dest\_g1

In this example there are 2 map/reduce stages (Stage-1 and Stage-2) and 1 File System related stage (Stage-0). Stage-0 basically moves the results from a temporary directory to the directory corresponding to the table dest\_g1.

A map/reduce stage itself comprises of 2 parts:

* A mapping from table alias to Map Operator Tree - This mapping tells the mappers which operator tree to call in order to process the rows from a particular table or result of a previous map/reduce stage. In Stage-1 in the above example, the rows from src table are processed by the operator tree rooted at a Reduce Output Operator. Similarly, in Stage-2 the rows of the results of Stage-1 are processed by another operator tree rooted at another Reduce Output Operator. Each of these Reduce Output Operators partitions the data to the reducers according to the criteria shown in the metadata.
* A Reduce Operator Tree - This is the operator tree which processes all the rows on the reducer of the map/reduce job. In Stage-1 for example, the Reducer Operator Tree is carrying out a partial aggregation where as the Reducer Operator Tree in Stage-2 computes the final aggregation from the partial aggregates computed in Stage-1

[**LanguageManual VirtualColumns**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+VirtualColumns)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362048&originalId=27820704))
* [Virtual Columns](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+VirtualColumns#LanguageManualVirtualColumns-VirtualColumns)
  + [Simple Examples](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+VirtualColumns#LanguageManualVirtualColumns-SimpleExamples)

**Virtual Columns**

Right now hive can only support 2 simple virtual columns:

One is INPUT+FILE+NAME, which is the input file's name for a mapper task.

the other is BLOCK+OFFSET\_\_INSIDE+FILE, which is the current global file position.

For block compressed file, it is the current block's file offset, which is the current block's first byte's file offset.

**Simple Examples**

select INPUT+FILE\_*NAME, key, BLOCKOFFSET*\_INSIDE+FILE from src;

select key, count(INPUT+FILE+NAME) from src group by key order by key;

select \* from src where BLOCK+OFFSET\_\_INSIDE+FILE > 12000 order by key;

[**LanguageManual Locks**](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Locks)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362041&originalId=27820696))
* [Locks](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Locks#LanguageManualLocks-Locks)
  + [Simple Examples](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Locks#LanguageManualLocks-SimpleExamples)

**Locks**

Right now hive can support the following locks:

* SHARED
* EXCLUSIVE

One is INPUT+FILE+NAME, which is the input file's name for a mapper task.

the other is BLOCK+OFFSET\_\_INSIDE+FILE, which is the current global file position.

For block compressed file, it is the current block's file offset, which is the current block's first byte's file offset.

**Simple Examples**

select INPUT+FILE\_*NAME, key, BLOCKOFFSET*\_INSIDE+FILE from src;

select key, count(INPUT+FILE+NAME) from src group by key order by key;

select \* from src where BLOCK+OFFSET\_\_INSIDE+FILE > 12000 order by key;

[**HivePlugins**](https://cwiki.apache.org/confluence/display/Hive/HivePlugins)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362098&originalId=27823730))

**Creating Custom UDFs**

First, you need to create a new class that extends UDF, with one or more methods named evaluate.

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());

}

}

(Note that there's already a built-in function for this, it's just an easy example).

After compiling your code to a jar, you need to add this to the hive classpath. See the section below on deploying jars.

Once hive is started up with your jars in the classpath, the final step is to register your function:

create temporary function my\_lower as 'com.example.hive.udf.Lower';

Now you can start using it:

hive> select my\_lower(title), sum(freq) from titles group by my\_lower(title);

...

Ended Job = job\_200906231019\_0006

OK

cmo 13.0

vp 7.0

For a more involved example, see [this page](https://cwiki.apache.org/confluence/display/Hive/GenericUDAFCaseStudy).

**Deploying jars for User Defined Functions and User Defined SerDes**

In order to start using your UDF, you first need to add the code to the classpath:

hive> add jar my\_jar.jar;

Added my\_jar.jar to class path

By default, it will look in the current directory. You can also specify a full path:

hive> add jar /tmp/my\_jar.jar;

Added /tmp/my\_jar.jar to class path

Your jar will then be on the classpath for all jobs initiated from that session. To see which jars have been added to the classpath you can use:

hive> list jars;

my\_jar.jar

**Hive User Defined Functions**

* [Hive User Defined Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-HiveUserDefinedFunctions)
  + [Built-in Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinOperators)
    - [Relational Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-RelationalOperators)
    - [Arithmetic Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ArithmeticOperators)
    - [Logical Operators](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-LogicalOperators)
    - [Complex Type Constructors](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ComplexTypeConstructors)
    - [Operators on Complex Types](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-OperatorsonComplexTypes)
  + [Built-in Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinFunctions)
    - [Mathematical Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-MathematicalFunctions)
    - [Collection Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-CollectionFunctions)
    - [Type Conversion Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-TypeConversionFunctions)
    - [Date Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-DateFunctions)
    - [Conditional Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-ConditionalFunctions)
    - [String Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-StringFunctions)
    - [Misc. Functions](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Misc.Functions)
      * [xpath](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-xpath)
      * [get\_json\_object](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-getjsonobject)
  + [Built-in Aggregate Functions (UDAF)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinAggregateFunctions%28UDAF%29)
  + [Built-in Table-Generating Functions (UDTF)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-BuiltinTableGeneratingFunctions%28UDTF%29)
    - [explode](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-explode)
    - [json\_tuple](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-jsontuple)
  + [GROUPing and SORTing on f(column)](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-GROUPingandSORTingonf%28column%29)

In the CLI, use the commands below to show the latest documentation:

SHOW FUNCTIONS;

DESCRIBE FUNCTION <function\_name>;

**Built-in Operators**

**Relational Operators**

The following operators compare the passed operands and generate a TRUE or FALSE value depending on whether the comparison between the operands holds.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A = B | All primitive types | TRUE if expression A is equal to expression B otherwise FALSE |
| A == B | None! | Fails because of invalid syntax. SQL uses =, not == |
| A <> B | All primitive types | NULL if A or B is NULL, TRUE if expression A is NOT equal to expression B otherwise FALSE |
| A < B | All primitive types | NULL if A or B is NULL, TRUE if expression A is less than expression B otherwise FALSE |
| A <= B | All primitive types | NULL if A or B is NULL, TRUE if expression A is less than or equal to expression B otherwise FALSE |
| A > B | All primitive types | NULL if A or B is NULL, TRUE if expression A is greater than expression B otherwise FALSE |
| A >= B | All primitive types | NULL if A or B is NULL, TRUE if expression A is greater than or equal to expression B otherwise FALSE |
| 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 | NULL if A or B is NULL, TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. The comparison is done character by character. The \_ character in B matches any character in A(similar to . in posix regular expressions) while the % character in B matches an arbitrary number of characters in A(similar to .\* in posix regular expressions) e.g. 'foobar' like 'foo' evaluates to FALSE where as 'foobar' like 'foo\_ \_ \_' evaluates to TRUE and so does 'foobar' like 'foo%' |
| A RLIKE B | strings | NULL if A or B is NULL, TRUE if string A matches the Java regular expression B(See Java regular expressions syntax), otherwise FALSE e.g. 'foobar' rlike 'foo' evaluates to FALSE where as 'foobar' rlike '^f.\*r$' evaluates to TRUE |
| A REGEXP B | strings | Same as RLIKE |

**Arithmetic Operators**

The following operators support various common arithmetic operations on the operands. All return number types; if any of the operands are NULL, then the result is also NULL.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A + B | All number types | Gives the result of adding A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. e.g. since every integer is a float, therefore float is a containing type of integer so the + operator on a float and an int will result in a float. |
| A - B | All number types | Gives the result of subtracting B from A. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A \* B | All number types | Gives the result of multiplying A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. Note that if the multiplication causing overflow, you will have to cast one of the operators to a type higher in the type hierarchy. |
| A / B | All number types | Gives the result of dividing B from A. The result is a double type. |
| A % B | All number types | Gives the reminder resulting from dividing A by B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A & B | All number types | Gives the result of bitwise AND of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A | B | All number types | Gives the result of bitwise OR of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| A ^ B | All number types | Gives the result of bitwise XOR of A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. |
| ~A | All number types | Gives the result of bitwise NOT of A. The type of the result is the same as the type of A. |

**Logical Operators**

The following operators provide support for creating logical expressions. All of them return boolean TRUE, FALSE, or NULL depending upon the boolean values of the operands. NULL behaves as an "unknown" flag, so if the result depends on the state of an unknown, the result itself is unknown.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand types** | **Description** |
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE. NULL if A or B is NULL |
| A && B | boolean | Same as A AND B |
| A OR B | boolean | TRUE if either A or B or both are TRUE; FALSE OR NULL is NULL; otherwise FALSE |
| A || B | boolean | Same as A OR B |
| NOT A | boolean | TRUE if A is FALSE or NULL if A is NULL. Otherwise FALSE. |
| ! A | boolean | Same as NOT A |

**Complex Type Constructors**

The following functions construct instances of complex types.

|  |  |  |
| --- | --- | --- |
| **Constructor Function** | **Operands** | **Description** |
| map | (key1, value1, key2, value2, ...) | Creates a map with the given key/value pairs |
| struct | (val1, val2, val3, ...) | Creates a struct with the given field values. Struct field names will be col1, col2, ... |
| array | (val1, val2, ...) | Creates an array with the given elements |

**Operators on Complex Types**

The following operators provide mechanisms to access elements in 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 e.g. if A is an array comprising of ['foo', 'bar'] then A[0] returns 'foo' and A[1] returns 'bar' |
| M[key] | M is a Map<K, V> and key has type K | Returns the value corresponding to the key in the map e.g. if M is a map comprising of {'f' -> 'foo', 'b' -> 'bar', 'all' -> 'foobar'} then M['all'] returns 'foobar' |
| S.x | S is a struct | Returns the x field of S. e.g for struct foobar {int foo, int bar} foobar.foo returns the integer stored in the foo field of the struct. |

**Built-in Functions**

**Mathematical Functions**

The following built-in mathematical functions are supported in hive; most return NULL when the argument(s) are NULL:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| BIGINT | round(double a) | Returns the rounded BIGINT value of the double |
| DOUBLE | round(double a, int d) | Returns the double rounded to d decimal places |
| BIGINT | floor(double a) | Returns the maximum BIGINT value that is equal or less than the double |
| BIGINT | ceil(double a), ceiling(double a) | Returns the minimum BIGINT value that is equal or greater than the double |
| double | rand(), rand(int seed) | Returns a random number (that changes from row to row) that is distributed uniformly from 0 to 1. Specifiying the seed will make sure the generated random number sequence is deterministic. |
| double | exp(double a) | Returns e^a where e is the base of the natural logarithm |
| double | ln(double a) | Returns the natural logarithm of the argument |
| double | log10(double a) | Returns the base-10 logarithm of the argument |
| double | log2(double a) | Returns the base-2 logarithm of the argument |
| double | log(double base, double a) | Return the base "base" logarithm of the argument |
| double | pow(double a, double p) power(double a, double p) | Return a^p |
| double | sqrt(double a) | Returns the square root of a |
| string | bin(BIGINT a) | Returns the number in binary format (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_bin>]) |
| string | hex(BIGINT a) hex(string a) | If the argument is an int, hex returns the number as a string in hex format. Otherwise if the number is a string, it converts each character into its hex representation and returns the resulting string. (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_hex>]) |
| string | unhex(string a) | Inverse of hex. Interprets each pair of characters as a hexidecimal number and converts to the character represented by the number. |
| string | conv(BIGINT num, int from\_base, int to\_base) | Converts a number from a given base to another (see [<http://dev.mysql.com/doc/refman/5.0/en/mathematical-functions.html#function_conv>]) |
| double | abs(double a) | Returns the absolute value |
| int double | pmod(int a, int b) pmod(double a, double b) | Returns the positive value of a mod b |
| double | sin(double a) | Returns the sine of a (a is in radians) |
| double | asin(double a) | Returns the arc sin of x if -1<=a<=1 or null otherwise |
| double | cos(double a) | Returns the cosine of a (a is in radians) |
| double | acos(double a) | Returns the arc cosine of x if -1<=a<=1 or null otherwise |
| int double | positive(int a) positive(double a) | Returns a |
| int double | negative(int a) negative(double a) | Returns -a |

**Collection Functions**

The following built-in collection functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| int | size(Map<K.V>) | Returns the number of elements in the map type |
| int | size(Array<T>) | Returns the number of elements in the array type |

**Type Conversion Functions**

The following type conversion functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| **Expected "=" to follow "type"** | cast(expr as <type>) | Converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) will convert the string '1' to it integral representation. A null is returned if the conversion does not succeed. |

**Date Functions**

The following built-in date functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| string | from\_unixtime(bigint unixtime[, string format]) | Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00" |
| bigint | unix\_timestamp() | Gets current time stamp using the default time zone. |
| bigint | unix\_timestamp(string date) | Converts time string in format yyyy-MM-dd HH:mm:ss to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20 11:30:01') = 1237573801 |
| bigint | unix\_timestamp(string date, string pattern) | Convert time string with given pattern (see [<http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>]) to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20', 'yyyy-MM-dd') = 1237532400 |
| string | to\_date(string timestamp) | Returns the date part of a timestamp string: to\_date("1970-01-01 00:00:00") = "1970-01-01" |
| int | year(string date) | Returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970 |
| int | month(string date) | Returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11 |
| int | day(string date) dayofmonth(date) | Return the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1 |
| int | hour(string date) | Returns the hour of the timestamp: hour('2009-07-30 12:58:59') = 12, hour('12:58:59') = 12 |
| int | minute(string date) | Returns the minute of the timestamp |
| int | second(string date) | Returns the second of the timestamp |
| int | weekofyear(string date) | Return the week number of a timestamp string: weekofyear("1970-11-01 00:00:00") = 44, weekofyear("1970-11-01") = 44 |
| int | datediff(string enddate, string startdate) | Return the number of days from startdate to enddate: datediff('2009-03-01', '2009-02-27') = 2 |
| int | date\_add(string startdate, int days) | Add a number of days to startdate: date\_add('2008-12-31', 1) = '2009-01-01' |
| int | date\_sub(string startdate, int days) | Subtract a number of days to startdate: date\_sub('2008-12-31', 1) = '2008-12-30' |

**Conditional Functions**

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| T | if(boolean testCondition, T valueTrue, T valueFalseOrNull) | Return valueTrue when testCondition is true, returns valueFalseOrNull otherwise |
| T | COALESCE(T v1, T v2, ...) | Return the first v that is not NULL, or NULL if all v's are NULL |
| T | CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END | When a = b, returns c; when a = d, return e; else return f |
| T | CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END | When a = true, returns b; when c = true, return d; else return e |

**String Functions**

The following are built-in String functions are supported in hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| int | length(string A) | Returns the length of the string |
| string | reverse(string A) | Returns the reversed string |
| string | concat(string A, string B...) | Returns the string resulting from concatenating the strings passed in as parameters in order. e.g. concat('foo', 'bar') results in 'foobar'. Note that this function can take any number of input strings. |
| string | concat\_ws(string SEP, string A, string B...) | Like concat() above, but with custom separator SEP. |
| string | substr(string A, int start) substring(string A, int start) | Returns the substring of A starting from start position till the end of string A e.g. substr('foobar', 4) results in 'bar' (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | substr(string A, int start, int len) substring(string A, int start, int len) | Returns the substring of A starting from start position with length len e.g. substr('foobar', 4, 1) results in 'b' (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | upper(string A) ucase(string A) | Returns the string resulting from converting all characters of A to upper case e.g. upper('fOoBaR') results in 'FOOBAR' |
| string | lower(string A) lcase(string A) | Returns the string resulting from converting all characters of B to lower case e.g. lower('fOoBaR') results in 'foobar' |
| string | trim(string A) | Returns the string resulting from trimming spaces from both ends of A e.g. trim(' foobar ') results in 'foobar' |
| string | ltrim(string A) | Returns the string resulting from trimming spaces from the beginning(left hand side) of A e.g. ltrim(' foobar ') results in 'foobar ' |
| string | rtrim(string A) | Returns the string resulting from trimming spaces from the end(right hand side) of A e.g. rtrim(' foobar ') results in ' foobar' |
| string | regexp\_replace(string A, string B, string C) | Returns the string resulting from replacing all substrings in B that match the Java regular expression syntax(See Java regular expressions syntax) with C e.g. regexp\_replace("foobar", "oo|ar", "") returns 'fb.' 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. |
| string | regexp\_extract(string subject, string pattern, int index) | Returns the string extracted using the pattern. e.g. regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) returns 'bar.' 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. See docs/api/java/util/regex/Matcher.html for more information on the 'index' or Java regex group() method. |
| string | parse\_url(string urlString, string partToExtract [, string keyToExtract]) | Returns the specified part from the URL. Valid values for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO. e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') returns 'facebook.com'. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') returns 'v1'. |
| string | get\_json\_object(string json\_string, string path) | Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid. **NOTE: The json path can only have the characters [0-9a-z\_], i.e., no upper-case or special characters. Also, the keys \*cannot** start with numbers.\* This is due to restrictions on Hive column names. |
| string | space(int n) | Return a string of n spaces |
| string | repeat(string str, int n) | Repeat str n times |
| int | ascii(string str) | Returns the numeric value of the first character of str |
| string | lpad(string str, int len, string pad) | Returns str, left-padded with pad to a length of len |
| string | rpad(string str, int len, string pad) | Returns str, right-padded with pad to a length of len |
| array | split(string str, string pat) | Split str around pat (pat is a regular expression) |
| int | find\_in\_set(string str, string strList) | Returns the first occurance of str in strList where strList is a comma-delimited string. Returns null if either argument is null. Returns 0 if the first argument contains any commas. e.g. find\_in\_set('ab', 'abc,b,ab,c,def') returns 3 |
| array<array<string>> | sentences(string str, string lang, string locale) | Tokenizes a string of natural language text into words and sentences, where each sentence is broken at the appropriate sentence boundary and returned as an array of words. The 'lang' and 'locale' are optional arguments. e.g. sentences('Hello there! How are you?') returns ( ("Hello", "there"), ("How", "are", "you") ) |
| array<struct<string,double>> | ngrams(array<array<string>>, int N, int K, int pf) | Returns the top-k N-grams from a set of tokenized sentences, such as those returned by the sentences() UDAF. See [StatisticsAndDataMining](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining) for more information. |
| array<struct<string,double>> | context\_ngrams(array<array<string>>, array<string>, int K, int pf) | Returns the top-k contextual N-grams from a set of tokenized sentences, given a string of "context". See [StatisticsAndDataMining](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining) for more information. |

**Misc. Functions**

**xpath**

The following functions are described in [LanguageManual XPathUDF](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+XPathUDF):

* xpath, xpath\_short, xpath\_int, xpath\_long, xpath\_float, xpath\_double, xpath\_number, xpath\_string

**get\_json\_object**

A limited version of JSONPath is supported:

* $ : Root object
* . : Child operator
* [] : Subscript operator for array
* \* : Wildcard for []

Syntax not supported that's worth noticing:

* : Zero length string as key
* .. : Recursive descent
* @ : Current object/element
* () : Script expression
* ?() : Filter (script) expression.
* [,] : Union operator
* [start:end.step] : array slice operator

Example: src\_json table is a single column (json), single row table:

+----+

json

+----+

{"store":

{"fruit":\[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],

"bicycle":{"price":19.95,"color":"red"}

},

"email":"amy@only\_for\_json\_udf\_test.net",

"owner":"amy"

}

+----+

The fields of the json object can be extracted using these queries:

hive> SELECT get\_json\_object(src\_json.json, '$.owner') FROM src\_json;

amy

hive> SELECT get\_json\_object(src\_json.json, '$.store.fruit\[0]') FROM src\_json;

{"weight":8,"type":"apple"}

hive> SELECT get\_json\_object(src\_json.json, '$.non\_exist\_key') FROM src\_json;

NULL

**Built-in Aggregate Functions (UDAF)**

The following are built-in aggregate functions are supported in Hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| bigint | count(\*), count(expr), count(DISTINCT expr[, expr\_.]) | count(\*) - Returns the total number of retrieved rows, including rows containing NULL values; count(expr) - Returns the number of rows for which the supplied expression is non-NULL; count(DISTINCT expr[, expr]) - Returns the number of rows for which the supplied expression(s) are unique and non-NULL. |
| double | sum(col), sum(DISTINCT col) | Returns the sum of the elements in the group or the sum of the distinct values of the column in the group |
| double | avg(col), avg(DISTINCT col) | Returns the average of the elements in the group or the average of the distinct values of the column in the group |
| double | min(col) | Returns the minimum of the column in the group |
| double | max(col) | Returns the maximum value of the column in the group |
| double | var\_pop(col) | Returns the variance of a numeric column in the group |
| double | var\_samp(col) | Returns the unbiased sample variance of a numeric column in the group |
| double | stddev\_pop(col) | Returns the standard deviation of a numeric column in the group |
| double | stddev\_samp(col) | Returns the unbiased sample standard deviation of a numeric column in the group |
| double | covar\_pop(col1, col2) | Returns the population covariance of a pair of numeric columns in the group |
| double | covar\_samp(col1, col2) | Returns the sample covariance of a pair of a numeric columns in the group |
| double | corr(col1, col2) | Returns the Pearson coefficient of correlation of a pair of a numeric columns in the group |
| double | percentile(col, p) | Returns the exact p^th^ percentile of an integer column in the group (does not work with floating point types). p must be between 0 and 1. |
| array<double> | percentile(col, array(p~1,, [, p,,2,,]...)) | Returns the exact percentiles p,,1,,, p,,2,,, ... of an integer column in the group (does not work with floating point types). p,,i~ must be between 0 and 1. |
| double | percentile\_approx(col, p [, B]) | Returns an approximate p^th^ percentile of a numeric column (including floating point types) in the group. The B parameter controls approximation accuracy at the cost of memory. Higher values yield better approximations, and the default is 10,000. When the number of distinct values in col is smaller than B, this gives an exact percentile value. |
| array<double> | percentile\_approx(col, array(p~1,, [, p,,2\_]...) [, B]) | Same as above, but accepts and returns an array of percentile values instead of a single one. |
| array<struct {'x','y'}> | histogram\_numeric(col, b) | Computes a histogram of a numeric column in the group using b non-uniformly spaced bins. The output is an array of size b of double-valued (x,y) coordinates that represent the bin centers and heights |
| array | collect\_set(col) | Returns a set of objects with duplicate elements eliminated |

**Built-in Table-Generating Functions (UDTF)**

Normal user-defined functions, such as concat(), take in a single input row and output a single output row. In contrast, table-generating functions transform a single input row to multiple output rows.

**explode**

explode() takes in an array as an input and outputs the elements of the array as separate rows. UDTF's can be used in the SELECT expression list and as a part of LATERAL VIEW.

An example use of explode() in the SELECT expression list is as follows:

Consider a table named myTable that has a single column (myCol) and two rows:

|  |
| --- |
| **Array<int> myCol** |
| [1,2,3] |
| [4,5,6] |

Then running the query:

SELECT explode(myCol) AS myNewCol FROM myTable;

Will produce:

|  |
| --- |
| **(int) myNewCol** |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

Using the syntax "SELECT udtf(col) AS colAlias..." has a few limitations:

* No other expressions are allowed in SELECT
  + SELECT pageid, explode(adid\_list) AS myCol... is not supported
* UDTF's can't be nested
  + SELECT explode(explode(adid\_list)) AS myCol... is not supported
* GROUP BY / CLUSTER BY / DISTRIBUTE BY / SORT BY is not supported
  + SELECT explode(adid\_list) AS myCol ... GROUP BY myCol is not supported

Please see [LanguageManual LateralView](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView) for an alternative syntax that does not have these limitations.

The following are built-in table-generating functions are supported in Hive:

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Name(Signature)** | **Description** |
| Array Type | explode(array<*TYPE*> a) | For each element in a, explode() generates a row containing that element |

**json\_tuple**

A new json\_tuple() UDTF is introduced in hive 0.7. It takes a set of names (keys) and return a tuple of values in one function.  
If you are using get\_json\_object() and want to replace it with json\_tuple, the only changes is that your query will be using json\_tuple() in lateral view rather than multiple get\_json\_object() in the select clause.

For example,

select a.timestamp, get\_json\_object(a.appevents, '$.eventid'), get\_json\_object(a.appenvets, '$.eventname') from log a;

should be changed to

select a.timestamp, b.\*

from log a lateral view json\_tuple(a.appevent, 'eventid', 'eventname') b as f1, f2;

**GROUPing and SORTing on f(column)**

A typical OLAP pattern is that you have a timestamp column and you want to group by daily or other less granular date windows than by second. So you might want to select concat(year(dt),month(dt)) and then group on that concat(). But if you attempt to GROUP BY or SORT BY a column on which you've applied a function and alias, like this:

select f(col) as fc, count(\*) from table\_name group by fc

You will get an error:

FAILED: Error in semantic analysis: line 1:69 Invalid Table Alias or Column Reference fc

Because you are not able to GROUP BY or SORT BY a column alias on which a function has been applied. There are two workarounds. First, you can reformulate this query with subqueries, which is somewhat complicated:

select sq.fc,col1,col2,...,colN,count(\*) from

(select f(col) as fc,col1,col2,...,colN from table\_name) sq

group by sq.fc,col1,col2,...,colN

Or you can make sure not to use a column alias, which is simpler:

select f(col) as fc, count(\*) from table\_name group by f(col)

Contact Tim Ellis (tellis) at RiotGames dot com if you would like to discuss this in further detail.

**Statistics and Data Mining in Hive**

This page is the secondary documentation for the slightly more advanced statistical and data mining functions that are being integrated into Hive, and especially the functions that warrant more than one-line descriptions.

* [Statistics and Data Mining in Hive](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-StatisticsandDataMininginHive)
  + [ngrams() and context\_ngrams(): N-gram frequency estimation](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-ngrams%28%29andcontextngrams%28%29%3ANgramfrequencyestimation)
    - [Use Cases](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-UseCases)
    - [Usage](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-Usage)
    - [Example](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-Example)
  + [histogram\_numeric(): Estimating frequency distributions](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-histogramnumeric%28%29%3AEstimatingfrequencydistributions)
    - [Use Cases](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-UseCases)
    - [Usage](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-Usage)
    - [Example](https://cwiki.apache.org/confluence/display/Hive/StatisticsAndDataMining#StatisticsAndDataMining-Example)

**ngrams() and context\_ngrams(): N-gram frequency estimation**

[N-grams](http://en.wikipedia.org/wiki/N-gram) are subsequences of length **N** drawn from a longer sequence. The purpose of the ngrams() UDAF is to find the k most frequent n-grams from one or more sequences. It can be used in conjunction with the sentences() UDF to analyze unstructured natural language text, or the collect() function to analyze more general string data.

Contextual n-grams are similar to n-grams, but allow you to specify a 'context' string around which n-grams are to be estimated. For example, you can specify that you're only interested in finding the most common two-word phrases in text that follow the context "I love". You could achieve the same result by manually stripping sentences of non-contextual content and then passing them to ngrams(), but context\_ngrams() makes it much easier.

**Use Cases**

1. (ngrams) Find important topics in text in conjunction with a stopword list.  
   2. (ngrams) Find trending topics in text.  
   3. (context\_ngrams) Extract marketing intelligence around certain words (e.g., "Twitter is \_\_\_").  
   4. (ngrams) Find frequently accessed URL sequences.  
   5. (context\_ngrams) Find frequently accessed URL sequences that start or end at a particular URL.  
   6. (context\_ngrams) Pre-compute common search lookaheads.

**Usage**

SELECT context\_ngrams(sentences(lower(tweet)), 2, 100 [, 1000]) FROM twitter;

The command above will return the top-100 bigrams (2-grams) from a hypothetical table called twitter. The tweet column is assumed to contain a string with arbitrary, possibly meaningless, text. The lower() UDF first converts the text to lowercase for standardization, and thensentences() splits up the text into arrays of words. The optional fourth argument is the **precision factor** that control the tradeoff between memory usage and accuracy in frequency estimation. Higher values will be more accurate, but could potentially crash the JVM with an OutOfMemory error. If omitted, sensible defaults are used.

SELECT context\_ngrams(sentences(lower(tweet)), array("i","love",null), 100, [, 1000]) FROM twitter;

The command above will return a list of the top 100 words that follow the phrase "i love" in a hypothetical database of Twitter tweets. Each null specifies the position of an n-gram component to estimate; therefore, every query must contain at least one null in the context array.

Note that the following two queries are identical, but ngrams() will be slightly faster in practice.

SELECT ngrams(sentences(lower(tweet)), 2, 100 [, 1000]) FROM twitter;

SELECT context\_ngrams(sentences(lower(tweet)), array(null,null), 100, [, 1000]) FROM twitter;

**Example**

SELECT explode(ngrams(sentences(lower(val)), 2, 10)) AS x FROM kafka;

{"ngram":[of","the],"estfrequency":23.0}

{"ngram":[on","the],"estfrequency":20.0}

{"ngram":[in","the],"estfrequency":18.0}

{"ngram":[he","was],"estfrequency":17.0}

{"ngram":[at","the],"estfrequency":17.0}

{"ngram":[that","he],"estfrequency":16.0}

{"ngram":[to","the],"estfrequency":16.0}

{"ngram":[out","of],"estfrequency":16.0}

{"ngram":[he","had],"estfrequency":16.0}

{"ngram":[it","was],"estfrequency":15.0}

SELECT explode(context\_ngrams(sentences(lower(val)), array("he", null), 10)) AS x FROM kafka;

{"ngram":[was],"estfrequency":17.0}

{"ngram":[had],"estfrequency":16.0}

{"ngram":[thought],"estfrequency":13.0}

{"ngram":[could],"estfrequency":9.0}

{"ngram":[would],"estfrequency":7.0}

{"ngram":[lay],"estfrequency":5.0}

{"ngram":[s],"estfrequency":4.0}

{"ngram":[wanted],"estfrequency":4.0}

{"ngram":[did],"estfrequency":4.0}

{"ngram":[felt],"estfrequency":4.0}

**histogram\_numeric(): Estimating frequency distributions**

Histograms represent frequency distributions from empirical data. The kind that is referred to here are histograms with variable-sized bins. Specifically, this UDAF will return a list of (x,y) pairs that represent histogram bin centers and heights. It's up to you to then plot them in Excel / Gnuplot / Matlab / Mathematica to get a nice graphical display.

**Use Cases**

1. Estimating the frequency distribution of a column, possibly grouped by other attributes.  
   2. Choosing discretization points in a continuous valued column.

**Usage**

SELECT histogram\_numeric(age) FROM users GROUP BY gender;

The command above is self-explanatory. Converting the output into a graphical display is a bit more involved. The following [Gnuplot](http://www.gnuplot.info/) command should do it, assuming that you've parsed the output from histogram() into a text file of (x,y) pairs called data.txt.

plot 'data.txt' u 1:2 w impulses lw 5

**Example**

SELECT explode(histogram\_numeric(val, 10)) AS x FROM normal;

{"x":-3.6505464999999995,"y":20.0}

{"x":-2.7514727901960785,"y":510.0}

{"x":-1.7956678951954481,"y":8263.0}

{"x":-0.9878507685761995,"y":19167.0}

{"x":-0.2625338380837097,"y":31737.0}

{"x":0.5057392319427763,"y":31502.0}

{"x":1.2774146480311135,"y":14526.0}

{"x":2.083955560712489,"y":3986.0}

{"x":2.9209550254545484,"y":275.0}

{"x":3.674835214285715,"y":14.0}

窗体顶端

窗体底端

[**AdminManual Configuration**](https://cwiki.apache.org/confluence/display/Hive/AdminManual+Configuration)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362070&originalId=27820454))

= Configuring Hive =

A number of configuration variables in Hive can be used by the administrator to change the behavior for their installations and user sessions. These variables can be configured in any of the following ways, shown in the order of preference:

* Using the set command in the cli for setting session level values for the configuration variable for all statements subsequent to the set command. e.g.
* set hive.exec.scratchdir=/tmp/mydir

sets the scratch directory (which is used by hive to store temporary output and plans) to /tmp/mydir for all subseq

* Using -hiveconf option on the cli for the entire session. e.g.
* bin/hive -hiveconf hive.exec.scratchdir=/tmp/mydir
* In hive-site.xml. This is used for setting values for the entire Hive configuration. e.g.
* <property>
* <name>hive.exec.scratchdir</name>
* <value>/tmp/mydir</value>
* <description>Scratch space for Hive jobs</description>
* </property>
* hive-default.xml - This configuration file contains the default values for various configuration variables that come with prepackaged in a Hive distribution. These should not be changed by the administrator. In order to override any of the values, create hive-site.xml instead and set the value in that file as shown above.

hive-default.xml is located in the conf directory in your installation root. hive-site.xml should also be created in the same directory.

Broadly the configuration variables are categorized into:

**Hive Configuration Variables**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Default Value** |
| hive.exec.script.wrapper | Wrapper around any invocations to script operator e.g. if this is set to python, the script passed to the script operator will be invoked as python <script command>. If the value is null or not set, the script is invoked as <script command>. | null |
| hive.exec.plan |  | null |
| hive.exec.scratchdir | This directory is used by hive to store the plans for different map/reduce stages for the query as well as to stored the intermediate outputs of these stages. | /tmp/<user.name>/hive |
| hive.querylog.location | Directory where structured hive query logs are created. One file per session is created in this directory. If this variable set to empty string structured log will not be created. | /tmp/<user.name> |
| hive.exec.submitviachild | Determines whether the map/reduce jobs should be submitted through a separate jvm in the non local mode. | false - By default jobs are submitted through the same jvm as the compiler |
| hive.exec.script.maxerrsize | Maximum number of serialization errors allowed in a user script invoked through TRANSFORM or MAP or REDUCE constructs. | 100000 |
| hive.exec.compress.output | Determines whether the output of the final map/reduce job in a query is compressed or not. | false |
| hive.exec.compress.intermediate | Determines whether the output of the intermediate map/reduce jobs in a query is compressed or not. | false |
| hive.jar.path | The location of hive\_cli.jar that is used when submitting jobs in a separate jvm. |  |
|  | hive.aux.jars.path | The location of the plugin jars that contain implementations of user defined functions and serdes. |
|  | hive.partition.pruning | A strict value for this variable indicates that an error is thrown by the compiler in case no partition predicate is provided on a partitioned table. This is used to protect against a user inadvertently issuing a query against all the partitions of the table. | nonstrict |
| hive.map.aggr | Determines whether the map side aggregation is on or not. | true |  |
| hive.join.emit.interval |  | 1000 |  |
| hive.map.aggr.hash.percentmemory |  | (float)0.5 |  |
| hive.default.fileformat | Default file format for CREATE TABLE statement. Options are TextFile, SequenceFile and RCFile | TextFile |  |
| hive.merge.mapfiles | Merge small files at the end of a map-only job. | true |  |
| hive.merge.mapredfiles | Merge small files at the end of a map-reduce job. | false |  |
| hive.merge.size.per.task | Size of merged files at the end of the job. | 256000000 |  |
| hive.merge.size.smallfiles.avgsize | When the average output file size of a job is less than this number, Hive will start an additional map-reduce job to merge the output files into bigger files. This is only done for map-only jobs if hive.merge.mapfiles is true, and for map-reduce jobs if hive.merge.mapredfiles is true. | 16000000 |  |
| hive.enforce.bucketing | If enabled, enforces inserts into bucketed tables to also be bucketed | false |  |

**Hive MetaStore Configuration Variables**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Default Value** |
| hive.metastore.metadb.dir |  |  |
| hive.metastore.warehouse.dir | Location of the default database for the warehouse |  |
| hive.metastore.uris |  |  |
| hive.metastore.usefilestore |  |  |
| hive.metastore.rawstore.impl |  |  |
| hive.metastore.local |  |  |
| javax.jdo.option.ConnectionURL | JDBC connect string for a JDBC metastore |  |
| javax.jdo.option.ConnectionDriverName | Driver class name for a JDBC metastore |  |
| javax.jdo.option.ConnectionUserName |  |  |
| javax.jdo.option.ConnectionPassword |  |  |
| org.jpox.autoCreateSchema | Creates necessary schema on startup if one doesn't exist. (e.g. tables, columns...) Set to false after creating it once. |  |
| org.jpox.fixedDatastore | Whether the datastore schema is fixed. |  |
| hive.metastore.checkForDefaultDb |  |  |
| hive.metastore.ds.connection.url.hook | Name of the hook to use for retriving the JDO connection URL. If empty, the value in javax.jdo.option.ConnectionURL is used as the connection URL |  |
| hive.metastore.ds.retry.attempts | The number of times to retry a call to the backing datastore if there were a connection error | 1 |
| hive.metastore.ds.retry.interval | The number of miliseconds between datastore retry attempts | 1000 |
| hive.metastore.server.min.threads | Minimum number of worker threads in the Thrift server's pool. | 200 |
| hive.metastore.server.max.threads | Maximum number of worker threads in the Thrift server's pool. | 10000 |

**Hive Configuration Variables used to interact with Hadoop**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Default Value** |
| hadoop.bin.path | The location of hadoop script which is used to submit jobs to hadoop when submitting through a separate jvm. | $HADOOP\_HOME/bin/hadoop |
| hadoop.config.dir | The location of the configuration directory of the hadoop installation | $HADOOP\_HOME/conf |
| fs.default.name |  | file:/// |
| map.input.file |  | null |
| mapred.job.tracker | The url to the jobtracker. If this is set to local then map/reduce is run in the local mode. | local |
| mapred.reduce.tasks | The number of reducers for each map/reduce stage in the query plan. | 1 |
| mapred.job.name | The name of the map/reduce job | null |

**Hive Variables used to pass run time information**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Default Value** |
| hive.session.id | The id of the Hive Session. |  |
| hive.query.string | The query string passed to the map/reduce job. |  |
| hive.query.planid | The id of the plan for the map/reduce stage. |  |
| hive.jobname.length | The maximum length of the jobname. | 50 |
| hive.table.name | The name of the hive table. This is passed to the user scripts through the script operator. |  |
| hive.partition.name | The name of the hive partition. This is passed to the user scripts through the script operator. |  |
| hive.alias | The alias being processed. This is also passed to the user scripts through the script operator. |  |

**Temporary Folders**

Hive uses temporary folders both on the machine running the Hive client and the default HDFS instance. These folders are used to store per-query temporary/intermediate data sets and are normally cleaned up by the hive client when the query is finished. However, in cases of abnormal hive client termination, some data may be left behind. The configuration details are as follows:

* On the HDFS cluster this is set to */tmp/hive-<username>* by default and is controlled by the configuration variable *hive.exec.scratchdir*
* On the client machine, this is hardcoded to */tmp/<username>*

Note that when writing data to a table/partition, Hive will first write to a temporary location on the target table's filesystem (using hive.exec.scratchdir as the temporary location) and then move the data to the target table. This applies in all cases - whether tables are stored in HDFS (normal case) or in file systems like S3 or even NFS.

**Log Files**

Hive client produces logs and history files on the client machine. Please see [Error Logs](http://wiki.apache.org/hadoop/Hive/GettingStarted#Error_Logs) on configuration details.

[**AdminManual MetastoreAdmin**](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Ashoat Tevosyan](https://cwiki.apache.org/confluence/display/~ashoat) on Jun 29, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362076&originalId=27820458))

== Hive Metastore ==

* + [Introduction](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-Introduction)
  + [Embedded Metastore](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-EmbeddedMetastore)
  + [Local Metastore](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-LocalMetastore)
  + [Remote Metastore](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-RemoteMetastore)
* [Metastore Deployment Options in Pictures](https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-MetastoreDeploymentOptionsinPictures)

**Introduction**

All the metadata for Hive tables and partitions are stored in Hive Metastore. Metadata is persisted using [JPOX](http://www.datanucleus.org/) ORM solution so any store that is supported by it. Most of the commercial relational databases and many open source datstores are supported. Any datastore that has JDBC driver can probably be used.

You can find an E/R diagram for the metastore [here](https://issues.apache.org/jira/secure/attachment/12471108/HiveMetaStore.pdf).

There are 3 different ways to setup metastore server using different Hive configurations. The relevant configuration parameters are

|  |  |
| --- | --- |
| Config Param | Description |
| javax.jdo.option.ConnectionURL | JDBC connection string for the data store which contains metadata |
| javax.jdo.option.ConnectionDriverName | JDBC Driver class name for the data store which contains metadata |
| hive.metastore.uris | Hive connects to this URI to make metadata requests for a remote metastore |
| hive.metastore.local | local or remote metastore |
| hive.metastore.warehouse.dir | URI of the default location for native tables |

Default configuration sets up an embedded metastore which is used in unit tests and is described in the next section. More practical options are described in the subsequent sections.

**Embedded Metastore**

Mainly used for unit tests and only one process can connect to metastore at a time. So it is not really a practical solution but works well for unit tests.

|  |  |  |
| --- | --- | --- |
| Config Param | Config Value | Comment |
| javax.jdo.option.ConnectionURL | jdbc:derby:;databaseName=../build/test/junit\_metastore\_db;create=true | derby database located at hive/trunk/build... |
| javax.jdo.option.ConnectionDriverName | org.apache.derby.jdbc.EmbeddedDriver | Derby embeded JDBC driver class |
| hive.metastore.uris | not needed since this is a local metastore |  |
| hive.metastore.local | true | embeded is local |
| hive.metastore.warehouse.dir | [Unknown macro: {user.dir} /../build/ql/test/data/warehouse" class="external-link" rel="nofollow">file://$](file://%24%3Cdiv%20class%3D/)  [Unknown macro: {user.dir}](file://%24%3Cdiv%20class%3D/)  [/../build/ql/test/data/warehouse](file://%24%3Cdiv%20class%3D/) | unit test data goes in here |

If you want to run the metastore as a network server so it can be accessed from multiple nodes try HiveDerbyServerMode.

**Local Metastore**

In local metastore setup, each Hive Client will open a connection to the datastore and make SQL queries against it. The following config will setup a metastore in a MySQL server. Make sure that the server accessible from the machines where Hive queries are executed since this is a local store. Also the jdbc client library is in the classpath of Hive Client.

|  |  |  |
| --- | --- | --- |
| Config Param | Config Value | Comment |
| javax.jdo.option.ConnectionURL | jdbc:mysql://<host name>/<database name>?createDatabaseIfNotExist=true | metadata is stored in a MySQL server |
| javax.jdo.option.ConnectionDriverName | com.mysql.jdbc.Driver | MySQL JDBC driver class |
| javax.jdo.option.ConnectionUserName | <user name> | user name for connecting to mysql server |
| javax.jdo.option.ConnectionPassword | <password> | password for connecting to mysql server |
| hive.metastore.uris | not needed because this is local store |  |
| hive.metastore.local | true | this is local store |
| hive.metastore.warehouse.dir | <base hdfs path> | default location for Hive tables. |

**Remote Metastore**

In remote metastore setup, all Hive Clients will make a connection a metastore server which in turn queries the datastore (MySQL in this example) for metadata. Metastore server and client communicate using [Thrift](http://incubator.apache.org/thrift) Protocol. Starting with Hive 0.5.0, you can start a thrift server by executing the following command:

hive --service metastore

In versions of Hive earlier than 0.5.0, it's instead necessary to run the thrift server via direct execution of Java:

$JAVA\_HOME/bin/java -Xmx1024m -Dlog4j.configuration=file://$HIVE\_HOME/conf/hms-log4j.properties -Djava.library.path=$HADOOP\_HOME/lib/native/Linux-amd64-64/ -cp $CLASSPATH org.apache.hadoop.hive.metastore.HiveMetaStore

If you execute Java directly, then JAVA\_HOME, HIVE\_HOME, HADOOP\_HOME must be correctly set; CLASSPATH should contain Hadoop, Hive (lib and auxlib), and Java jars.

Server Configuration Parameters

|  |  |  |
| --- | --- | --- |
| Config Param | Config Value | Comment |
| javax.jdo.option.ConnectionURL | jdbc:mysql://<host name>/<database name>?createDatabaseIfNotExist=true | metadata is stored in a MySQL server |
| javax.jdo.option.ConnectionDriverName | com.mysql.jdbc.Driver | MySQL JDBC driver class |
| javax.jdo.option.ConnectionUserName | <user name> | user name for connecting to mysql server |
| javax.jdo.option.ConnectionPassword | <password> | password for connecting to mysql server |
| hive.metastore.warehouse.dir | <base hdfs path> | default location for Hive tables. |

Client Configuration Parameters

|  |  |  |
| --- | --- | --- |
| Config Param | Config Value | Comment |
| hive.metastore.uris | thrift://<host\_name>:<port> | host and port for the thrift metastore server |
| hive.metastore.local | false | this is local store |
| hive.metastore.warehouse.dir | <base hdfs path> | default location for Hive tables. |

If you are using MySQL as the datastore for metadata, put MySQL client libraries in HIVE\_HOME/lib before starting Hive Client or HiveMetastore Server.

# [AdminManual SettingUpHiveServer](https://cwiki.apache.org/confluence/display/Hive/AdminManual+SettingUpHiveServer)

[**HiveServer**](https://cwiki.apache.org/confluence/display/Hive/HiveServer)

* Added by [Confluence Administrator](https://cwiki.apache.org/confluence/display/~admin), last edited by [Travis Powell](https://cwiki.apache.org/confluence/display/~tpowell) on Aug 08, 2011  ([view change](https://cwiki.apache.org/confluence/pages/diffpages.action?pageId=27362111&originalId=27823726))
* [show comment](https://cwiki.apache.org/confluence/display/Hive/HiveServer)

Once Hive has been built using steps in [Getting Started](https://cwiki.apache.org/confluence/display/Hive/GettingStarted), the thrift server can be started by running the following:

$ build/dist/bin/hive --service hiveserver --help

usage HIVE\_PORT=xxxx ./hive --service hiveserver

HIVE\_PORT : Specify the server port

$ bin/hive --service hiveserver

After starting the server, to test if the server is working well, run the hiveserver and jdbc tests in 'standalone' mode. The HIVE\_PORT is assumed to be 10000 on localhost for this case.

$ ant test -Dtestcase=TestJdbcDriver -Dstandalone=true

$ ant test -Dtestcase=TestHiveServer -Dstandalone=true

The service supports clients in multiple languages. For more details see [Hive Client](https://cwiki.apache.org/confluence/display/Hive/HiveClient)

**NOTE**:

* HiveServer is multithreaded and was designed to support multiple concurrent client connections. However, there is currently no test coverage for this scenario. If you want to play it safe we recommend running one HiveServer process per client connection. Please see [JIRA: HIVE-80](http://issues.apache.org/jira/browse/HIVE-80)for more information.
* Hive server and clients communicates through Thrift and FB303 services. In some distributions, both the Hadoop and Hive distributions have different versions of libthrift.jar and libfb303.jar. If they are incompatible, it may cause Thrift connection error when running the unit test on standalone mode. The solution is to remove the Hadoop's version of libthrift.jar and libfb303.jar.

**Hive JDBC Driver**

* [Hive JDBC Driver](https://cwiki.apache.org/confluence/display/Hive/HiveJDBCInterface#HiveJDBCInterface-HiveJDBCDriver)
  + [Integration with Pentaho](https://cwiki.apache.org/confluence/display/Hive/HiveJDBCInterface#HiveJDBCInterface-IntegrationwithPentaho)
  + [Integration with SQuirrel SQL Client](https://cwiki.apache.org/confluence/display/Hive/HiveJDBCInterface#HiveJDBCInterface-IntegrationwithSQuirrelSQLClient)

The current JDBC interface for Hive only supports running queries and fetching results. Only a small subset of the metadata calls are supported.

To see how the JDBC interface can be used, see [sample code](https://cwiki.apache.org/confluence/display/Hive/HiveClient).

**Integration with Pentaho**

1. Download pentaho report designer from the [pentaho website](http://sourceforge.net/project/showfiles.php?group_id=140317&package_id=192362)
2. Overwrite report-designer.sh with the code provided below.
3. #!/bin/sh
4. HADOOP\_CORE={{ls $HADOOP\_HOME/hadoop-\*-core.jar}}
5. CLASSPATH=.:$HADOOP\_CORE:$HIVE\_HOME/conf
6. for i in ${HIVE\_HOME}/lib/\*.jar ; do
7. CLASSPATH=$CLASSPATH:$i
8. done
9. CLASSPATH=$CLASSPATH:launcher.jar
10. echo java -XX:MaxPermSize=512m -cp $CLASSPATH -jar launcher.jar
11. java -XX:MaxPermSize=512m -cp $CLASSPATH org.pentaho.commons.launcher.Launcher
12. Build and start the hive server with instructions from [HiveServer](https://cwiki.apache.org/confluence/display/Hive/HiveServer)
13. Compile and run the hive jdbc client code to load some data (I haven't figured out how to do this in report designer yet). See [sample code](https://cwiki.apache.org/confluence/display/Hive/HiveClient) for loading the data.
14. Run the report designer (note step 2)
15. $ sh reporter-designer.sh
16. Select 'Report Design Wizard'
17. select a template - say 'fall template' - next
18. create a new data source - JDBC (custom), Generic database
19. Provide hive jdbc parameters. Give the connection a name 'hive'.
20. URL: jdbc:hive://localhost:10000/default
21. Driver name: org.apache.hadoop.hive.jdbc.HiveDriver
22. Username and password are empty
23. Click on 'Test'. The test should succeed
24. Edit the query: select 'Sample Query', click edit query, click on the connection 'hive'. create a new query. Write a query on the table testHiveDriverTable: eg: select \* from testHiveDriverTable. Click next.
25. Layout Step: Add PageOfPages to Group Items By. Add key and value as Selected Items. Click next. And Finish.
26. Change the Report header to 'hive-pentaho-report'. Change the type of the header to 'html'
27. Run the report and generate pdf. You should get something like the report attached here.

**Integration with SQuirrel SQL Client**

1. Download, install and start the SQuirrel SQL Client from the [SQuirrel SQL website](http://squirrel-sql.sourceforge.net/).
2. Select 'Drivers -> New Driver...' to register the Hive JDBC driver.
   1. Enter the driver name and example URL:
   2. Name: Hive
   3. Example URL: jdbc:hive://localhost:10000/default
3. Select 'Extra Class Path -> Add' to add the following jars from your local Hive and Hadoop distribution. You will need to build Hive from the trunk after the commit of [HIVE-679](https://issues.apache.org/jira/browse/HIVE-679).
4. HIVE\_HOME/build/dist/lib/\*.jar

HADOOP\_HOME/hadoop-\*-core.jar

1. Select 'List Drivers'. This will cause SQuirrel to parse your jars for JDBC drivers and might take a few seconds. From the 'Class Name' input box select the Hive driver:
2. org.apache.hadoop.hive.jdbc.HiveDriver

1. Click 'OK' to complete the driver registration.
2. Select 'Aliases -> Add Alias...' to create a connection alias to your Hive server.
   1. Give the connection alias a name in the 'Name' input box.
   2. Select the Hive driver from the 'Driver' drop-down.
   3. Modify the example URL as needed to point to your Hive server.
   4. Leave 'User Name' and 'Password' blank and click 'OK' to save the connection alias.
3. To connect to the Hive server, double-click the Hive alias and click 'Connect'.

When the connection is established you will see errors in the log console and might get a warning that the driver is not JDBC 3.0 compatible. These alerts are due to yet-to-be-implemented parts of the JDBC metadata API and can safely be ignored. To test the connection enter *SHOW TABLES* in the console and click the run icon.

Also note that when a query is running, support for the 'Cancel' button is not yet available.

**Hive ODBC Driver**

The Hive ODBC Driver is a software library that implements the Open Database Connectivity (ODBC) API standard for the Hive database management system, enabling ODBC compliant applications to interact seamlessly (ideally) with Hive through a standard interface. This driver will NOT be built as a part of the typical Hive build process and will need to be compiled and built separately according to the instructions below.

**Suggested Reading**

This guide assumes you are already familiar with the following:

* [Hive](https://cwiki.apache.org/confluence/display/Hive/Home)
* [Hive Server](https://cwiki.apache.org/confluence/display/Hive/HiveServer)
* [Thrift](http://wiki.apache.org/thrift/)
* [ODBC API](http://msdn.microsoft.com/en-us/library/ms714177(VS.85).aspx)
* [unixODBC](http://www.unixodbc.org/)

**Software Requirements**

The following software components are needed for the successful compilation and operation of the Hive ODBC driver:

* **Hive Server** - a service through which clients may remotely issue Hive commands and requests. The Hive ODBC driver depends on Hive Server to perform the core set of database interactions. Hive Server is built as part of the Hive build process. More information regarding Hive Server usage can be found [here](https://cwiki.apache.org/confluence/display/Hive/HiveServer).
* **Apache Thrift** - a scalable cross-language software framework that enables the Hive ODBC driver (specifically the Hive client) to communicate with the Hive Server. See this link for the details on [Thrift Installation](http://wiki.apache.org/thrift/ThriftInstallation). The Hive ODBC driver was developed with Thrift trunk version r790732, but the latest revision should also be fine. Make sure you note the Thrift install path during the Thrift build process as this information will be needed during the Hive client build process. The Thrift install path will be referred to as THRIFT\_HOME.

**Driver Architecture**

Internally, the Hive ODBC Driver contains two separate components: Hive client, and the unixODBC API wrapper.

* **Hive client** - provides a set of C-compatible library functions to interact with Hive Server in a pattern similar to those dictated by the ODBC specification. However, Hive client was designed to be independent of unixODBC or any ODBC specific headers, allowing it to be used in any number of generic cases beyond ODBC.
* **unixODBC API wrapper** - provides a layer on top of Hive client that directly implements the ODBC API standard. The unixODBC API wrapper will be compiled into a shared object library, which will be the final form of the Hive ODBC driver. The wrapper files will remain a file attachment on the associated JIRA until it can be checked into the unixODBC code repository: [HIVE-187](https://issues.apache.org/jira/browse/HIVE-187), [HIVE-1101](https://issues.apache.org/jira/browse/HIVE-1101).

NOTE: Hive client needs to be built and installed before the unixODBC API wrapper can compile successfully.

**Hive Client Build/Setup**

In order to build the Hive client:

1. Checkout and setup the latest version of Apache Hive. For more details, see [Getting Started with Hive](https://cwiki.apache.org/confluence/display/Hive/GettingStarted). From this point onwards, the path to the Hive root directory will be referred to as HIVE\_HOME.
2. Build the Hive client by running the following command from HIVE\_HOME. This will compile and copy the libraries and header files to HIVE\_HOME/build/odbc/. Please keep in mind that all paths should be fully specified (no relative paths). If you encounter an "undefined reference to vtables" error, make sure that you have specified the absolute path for thrift.home.
3. $ ant compile-cpp -Dthrift.home=<THRIFT\_HOME>

You can optionally force Hive client to compile into a non-native bit architecture by specifying the additional parameter (assuming you have the proper compilation libraries):

$ ant compile-cpp -Dthrift.home=<THRIFT\_HOME> -Dword.size=<32 or 64>

You can verify the entire Hive compilation by running the Hive test suite from HIVE\_HOME. Specifying the argument '-Dthrift.home=<THRIFT\_HOME>' will enable the tests for the Hive client. If you do NOT specify thrift.home, the Hive client tests will not be run and will just return successful.

$ ant test -Dthrift.home=<THRIFT\_HOME>

You can specifically execute the Hive client tests by running the above command from HIVE\_HOME/odbc/. NOTE: Hive client tests require that a local Hive Server be operating on port 10000.  
1.#3 To install the Hive client libraries onto your machine, run the following command from HIVE\_HOME/odbc/. NOTE: The install path defaults to /usr/local. While there is no current way to change this default directory from the ant build process, a manual install may be performed by skipping the command below and copying out the contents of HIVE\_HOME/build/odbc/lib and HIVE\_HOME/build/odbc/include into their local file system counterparts.

$ sudo ant install -Dthrift.home=<THRIFT\_HOME>

NOTE: The compiled static library, libhiveclient.a, requires linking with stdc++ as well as thrift libraries to function properly.  
NOTE: Currently, there is no way to specify non-system library and header directories to the unixODBC build process. Thus, the Hive client libraries and headers MUST be installed to a default system location in order for the unixODBC build process to detect these files. This issue may be remedied in the future.

**unixODBC API Wrapper Build/Setup**

After you have built and installed the Hive client, you can now install the unixODBC API wrapper:

1. In the unixODBC root directory, run the following command:
2. $ ./configure --enable-gui=no --prefix=<unixODBC\_INSTALL\_DIR>

If you encounter the the errors: "redefinition of 'struct \_hist\_entry'" or "previous declaration of 'add\_history' was here" then re-execute the configure with the following command:

$ ./configure --enable-gui=no --enable-readline=no --prefix=<unixODBC\_INSTALL\_DIR>

To force the compilation of the unixODBC API wrapper into a non-native bit architecture, modify the CC and CXX environment variables to include the appropriate flags. For example:

$ CC="gcc -m32" CXX="g++ -m32" ./configure --enable-gui=no --enable-readline=no --prefix=<unixODBC\_INSTALL\_DIR>

1.#2 Compile the unixODBC API wrapper with the following:

$ make

1.#3 If you want to completely install unixODBC and all related drivers:

1. Run the following from the unixODBC root directory:
2. $ sudo make install

a.#2 If your system complains about undefined symbols during unixODBC testing (such as with isql or odbcinst) after installation, try running ldconfig to update your dynamic linker's runtime libraries.  
1.#4 If you only want to obtain the Hive ODBC driver shared object library:

1. After compilation, the driver will be located at <unixODBC\_BUILD\_DIR>/Drivers/hive/.libs/libodbchive.so.1.0.0.
2. This may be copied to any other location as desired. Keep in mind that the Hive ODBC driver has a dependency on the Hive client shared object library: libhiveclient.so and libthrift.so.0.
3. You can manually install the unixODBC API wrapper by doing the following:
4. $ cp <unixODBC\_BUILD\_DIR>/Drivers/hive/.libs/libodbchive.so.1.0.0 <SYSTEM\_INSTALL\_DIR>
5. $ cd <SYSTEM\_INSTALL\_DIR>
6. $ ln -s libodbchive.so.1.0.0 libodbchive.so
7. $ ldconfig

**Connecting the Driver to a Driver Manager**

This portion assumes that you have already built and installed both the Hive client and the unixODBC API wrapper shared libraries on the current machine. To connect the Hive ODBC driver to a previously installed Driver Manager (such as the one provided by unixODBC or a separate application):

1. Locate the odbc.ini file associated with the Driver Manager (DM):
   1. If you are installing the driver on the system DM, then you can run the following command to print the locations of DM configuration files.
   2. $ odbcinst -j
   3. unixODBC 2.2.14
   4. DRIVERS............: /usr/local/etc/odbcinst.ini
   5. SYSTEM DATA SOURCES: /usr/local/etc/odbc.ini
   6. FILE DATA SOURCES..: /usr/local/etc/ODBCDataSources
   7. USER DATA SOURCES..: /home/ehwang/.odbc.ini
   8. SQLULEN Size.......: 8
   9. SQLLEN Size........: 8
   10. SQLSETPOSIROW Size.: 8

a.#2 If you are installing the driver on an application DM, then you have to help yourself on this one https://cwiki.apache.org/confluence/images/icons/emoticons/wink.gif. Hint: try looking in the installation directory of your application.

1. Keep in mind that an application's DM can exist simultaneously with the system DM and will likely use its own configuration files, such as odbc.ini.
2. Also, note that some applications do not have their own DMs and simply use the system DM.
3. Add the following section to the DM's corresponding odbc.ini:
4. [Hive]
5. Driver = <path\_to\_libodbchive.so>
6. Description = Hive Driver v1
7. DATABASE = default
8. HOST = <Hive\_server\_address>
9. PORT = <Hive\_server\_port>
10. FRAMED = 0

**Testing with ISQL**

Once you have installed the necessary Hive ODBC libraries and added a Hive entry in your system's default odbc.ini, you will be able to interactively test the driver with isql:

$ isql -v Hive

If your system does not have isql, you can obtain it by installing the entirety of unixODBC. If you encounter an error saying that the shared libraries cannot be opened by isql, use the ldd tool to ensure that all dynamic library dependencies are resolved and use the file tool to ensure that isql and all necessary libraries are compiled into the same architecture (32 or 64 bit).

**Build libodbchive.so for 3rd Party Driver Manager**

If you want to build libodbchive.so for other Driver Manager (for example, MicroStrategy uses DataDirect ODBC libraries which contains its own Driver Manager), you need to configure and build libodbchive.so against that Driver Manager (libodbc.so and libodbcinst.so).

If you have the 3rd party Driver Manager installed, the easiest way to do that is to find the installation directory containing libodbc.so and libodbcinst.so, and set that directory to LD\_LIBRARY\_PATH. Then you need to run configure and make for the Hive ODBC driver. After you get the libodbchive.so, make sure the 3rd party application can access the dynamic library libodbchive.so, libthrift.so and libhiveclient.so (through LD\_LIBRARY\_PATH or ldconfig).

If you build libodbchive.so for the 3rd party Driver Manager, isql may not work with the same set of .so files. So you may need to compile a different libodbchive.so for each Driver Manager.

**Troubleshooting**

* Hive client build process
  + "libthrift.a: could not read symbols: Bad value" or "relocation R\_X86\_64\_32 against `a local symbol' can not be used when making a shared object"?
    - Try recompiling your Apache Thrift libraries with the -fPIC option for your C++ compiler
  + "undefined reference to vtable" ?
    - Make sure that your Apache Thrift libraries are being included from the proper Thrift directory and that it has the same architecture (32 or 64 bit) as the Hive client.
    - Also, check to make sure you are providing a fully qualified path for the thrift.home parameter.
  + In general, ldd, file, and nm are essential unix tools for debugging problems with shared object libraries. If you don't know what they are, use man to get more details.

**Current Status**

* Comments: Please keep in mind that this is still an initial version and is still very rough around the edges. However, it provides basic ODBC 3.51 API support for connecting, executing queries, fetching, etc. This driver has been successfully tested on 32-bit and 64-bit linux machines with iSQL. It has also been tested with partial success on enterprise applications such as MicroStrategy. Due to licensing reasons, the unixODBC API wrapper files will be uploaded as a separate JIRA attachment that will not be part of this code repository.
* Limitations:
  + Only support for Linux operating systems
  + No support for Unicode
  + No support for asynchronous execution of queries
  + Does not support pattern matching for functions such as SQLColumns and SQLTables; requires exact matches.
  + Hive Server is currently not thread safe (see JIRA HIVE-80: <https://issues.apache.org/jira/browse/HIVE-80>). This will prevent the driver from safely making multiple connections to the same Hive Server. We need to resolve this issue to allow the driver to operate properly.
  + Hive Server's getSchema() function seems to have trouble with certain types of queries (such as "SELECT \* ..." or "EXPLAIN"), and so the Hive ODBC driver sometimes has difficulties with these queries as well.
* ODBC API Function Support (does anyone know how to remove the linking from the function names?):

|  |  |
| --- | --- |
| SQLAllocConnect | supported |
| SQLAllocEnv | supported |
| SQLAllocHandle | supported |
| SQLAllocStmt | supported |
| SQLBindCol | supported |
| SQLBindParameter | NOT supported |
| SQLCancel | NOT supported |
| SQLColAttribute | supported |
| SQLColumns | supported |
| SQLConnect | supported |
| SQLDescribeCol | supported |
| SQLDescribeParam | NOT supported |
| SQLDisconnect | supported |
| SQLDriverConnect | supported |
| SQLError | supported |
| SQLExecDirect | supported |
| SQLExecute | supported |
| SQLExtendedFetch | NOT supported |
| SQLFetch | supported |
| SQLFetchScroll | NOT supported |
| SQLFreeConnect | supported |
| SQLFreeEnv | supported |
| SQLFreeHandle | supported |
| SQLFreeStmt | supported |
| SQLGetConnectAttr | NOT supported |
| SQLGetData | supported (however, SQLSTATE not returning values) |
| SQLGetDiagField | NOT supported |
| SQLGetDiagRec | supported |
| SQLGetInfo | partially supported; (to get MSTR v9 running) |
| SQLMoreResults | NOT supported |
| SQLNumParams | NOT supported |
| SQLNumResultCols | supported |
| SQLParamOptions | NOT supported |
| SQLPrepare | supported; but does not permit parameter markers |
| SQLRowCount | NOT supported |
| SQLSetConnectAttr | NOT supported |
| SQLSetConnectOption | NOT supported |
| SQLSetEnvAttr | Limited support |
| SQLSetStmtAttr | NOT supported |
| SQLSetStmtOption | NOT supported |
| SQLTables | supported |
| SQLTransact | NOT supported |

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端

窗体顶端

窗体底端