**Hive Data Definitions**

1. **Creating a database schema:** How to create a database in Hive

The Create Database statement is used to create a database in Hive. By default, there is a database in Hive named default.

The general format of creating a database is as follows:

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

[COMMENT database\_comment]

[LOCATION hdfs\_path]

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

Ex:

Create database if not exists Hive\_learning

Comment 'This is my first DB'

Location '/my/directory'

With dbproperties ('Created by'='User','Created on'='1-Jan-2015');

1. **Dropping a database schema:** Drop a database in Hive

Drop Database statements drop the database and the objects inside that database. When a database is dropped, its directory is also deleted. The general format of dropping a database is as follows:

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

Where:

* DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.
* [IF EXISTS]: This is an optional clause. If not used, an error is thrown when there is an attempt to drop a database that does not exist.
* [RESTRICT|CASCADE]: This is an optional clause. RESTRICT is used to restrict the database from getting dropped if there are one or more tables present in the database. RESTRICT is the default behavior of the database. CASCADE is used to drop all the tables present in the database before dropping the database

Ex:

Drop database if exists Hive\_learning restrict;

1. **Altering a database schema:** Alter a database in Hive

The ALTER DATABASE command in Hive is used to alter dbproperties or set the dbproperties of a database. Using the ALTER DATABASE command, we can only alter dbproperties and nothing else.

The general format for altering a database is as follows:

ALTER (DATABASE|SCHEMA) database\_name SET DBPROPERTIES (property\_name=property\_value,..);

Where:

* DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably:
* SET DBPROPERTIES (property\_name=property\_value, ...)

Ex:

Alter database Hive\_learning set dbproperties ('Created by' = 'User1', 'Created on'= '15-Jan-2015');

1. **Using a database schema:** Use a database in Hive

The USE DATABASE command is used to switch to the database, or it sets the database as the working database. The general format of using a database is as follows:

USE (DATABASE|SCHEMA) database\_name;

Where:

* DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.

Ex:

Use database Hive\_learning;

1. **Showing database schemas:** Show databases in Hive

The SHOW DATABASE command is used to list all the databases in the Hive metastore. The general format is as follows:

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

Where:

* DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.
* [LIKE]: Is an optional clause. This clause is used to filter the databases with the help of a regular expression. There can only be two wildcards in the regular expression, which are \* for any character(s) or | for a choice.

Ex:

SHOW database like 'Hive\*';

1. **Describing a database schema:** Describe databases in Hive

The DESCRIBE DATABASE command is used to get information about the database, such as the name of the database, its comments, its location on the filesystem, and its dbproperties.

The general format of using the command is as follows:

DESCRIBE DATABASE/SCHEMA [EXTENDED] db\_name;

Where:

* DATABASE|SCHEMA: These are the same thing. These words can be used interchangeably.
* [EXTENDED]: This is an optional clause. This clause will list all the dbproperties attached to a particular database in Hive.

Ex:

Describe database extended Hive\_learning;

1. **Creating tables:** Create tables in Hive

The CREATE TABLE statement creates metadata in the database. The table in Hive is the way to read data from files present in HDFS in the table or a structural format. The general format of using the command is as follows:

CREATE [TEMPORARY] [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]

[SKEWED BY (col\_name, col\_name, ...)

ON ((col\_value, col\_value,..), (col\_value, col\_value,..), ...)

[STORED AS DIRECTORIES]

[

[ROW FORMAT row\_format]

[STORED AS file\_format]

| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]

]

[LOCATION hdfs\_path]

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

[AS select\_statement];

Where:

* [TEMPORARY]: This is an optional clause. This clause is used to create temporary tables.
* [EXTERNAL]: This is an optional clause. This clause is used to create external tables the same as in the case of RDBMS.
* [IF NOT EXISTS]: This is an optional clause. If there is an attempt to create a table that is already present in the database, an error is thrown.
* [COMMENT col/table\_comment]: This is an optional clause. This is used to attach comments to a particular column.
* [PARTITIONED BY]: This is an optional clause. This clause is used to create partitioned tables.
* [CLUSTERED BY]: This is an optional clause. This clause is used for bucketing purposes.
* [SKEWED BY]: This option is used to improve performance for tables where one or more columns have skewed values.
* [LOCATION hdfs\_path]: This option is used while creating external tables. This is the location where files are placed, which is referred to by the external table for the data.
* [TBLPROPERTIES]: This is an optional clause. This clause allows you to attach more information about the table in the form of a key-value pair.
* [AS select\_statement]: Create Table As Select, popularly known as CTAS, is used to create a table based on the output of the other table or existing

**8) Dropping tables:** Drop a table in Hive

DROP TABLE command removes the table from the database, including the data from the table.

The general format of using the DROP TABLE command is as follows:

DROP TABLE [IF EXISTS] table\_name [PURGE];

Where:

* [IF EXISTS]: Is an optional clause. If not used, an error is thrown when there is an attempt to drop a table that does not exist in the database.
* [PURGE]: Is an optional clause. If specified, the data is not saved in the Trash folder under the home directory and is lost forever.

Ex:

Drop table if exists Hive\_Test\_table1 purge;

**9) Truncating tables:** Truncate a table in Hive

The TRUNCATE command removes all rows from the table as well as from the partition, but keeps the table structure as it is.

The general format of using the truncate table command is as follows:

TRUNCATE TABLE table\_name [PARTITION partition\_spec];

Where:

* partition\_spec: (partition\_column = partition\_col\_value, partition\_column = partition\_col\_value,...)

Ex:

Truncate table Sales;

**10) Renaming tables:** Rename a table in Hive

The renaming command renames the old table name with a new table name. The general format of using the RENAME table command is as follows:

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

Ex:

Alter Table Hive\_Test\_table1 RENAME TO Hive\_Test\_table;

**11) Altering table properties:** Alter table properties in Hive

The ALTER TABLE properties command alters the table properties. The general format of using the command is as follows:

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties;

Ex:

Alter Table Hive\_Test\_table SET TBLPROPERTIES ('comment' = 'This is a new comment');

**12) Creating views:** Create a view in Hive

A view is a virtual table that acts as a window to the data for the underlying table commonly known as the base table. It consists of rows and columns but no physical data.

The general syntax of creating a view is as follows:

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

[COMMENT view\_comment]

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

AS SELECT ...;

Where:

* [IF NOT EXISTS]: Is an optional clause. If there is an attempt to create a view that is already present in the database, then an error is thrown.
* [COMMENT col\_comment]: Is an optional clause. This is used to attach comments to a particular column. This comment clause can be used to add a description about the column.
* [COMMENT table\_comment]: Is an optional clause. This is used to attach comments to a view.
* [TBLPROPERTIES (property\_name = property\_value, ...)]: Is an optional clause. This clause allows you to attach more information about the table in the form of a key-value pair.

Ex:

Create view if not exists Hive\_view\_2

As select id, firstname from Hive\_learningWhere firstname = 'John';

**13) Dropping views:** Drop a view in Hive

The DROP VIEW command removes the view from the database. It removes the metadata, but the base table remains intact. If a base table is a view that is dropped, then the dependent view remains in an invalid state, which is either dropped or recreated. The general syntax for dropping a view is as follows:

DROP VIEW [IF EXISTS] view\_name;

Where:

* [IF EXISTS]: Is an optional clause. If there is an attempt to drop a view that does not exist, an error is thrown. To prevent this error, the IF EXISTS clause is specified.

Ex:

Drop view Hive\_view;

**14) Altering the view properties:** Alter the view properties in Hive

This command is used to alter the view properties, the same as in the case of tables. The general syntax for altering a view is as follows:

ALTER VIEW view\_name SET TBLPROPERTIES table\_properties;

Where:

* table\_properties is defined as : (property\_name = property\_value, property\_name = property\_value, ...)

Ex:

Alter View Hive\_view SET TBLPROPERTIES ('comment' = 'This is a new comment');

**15) Altering the view as select:** Alter the view as select in Hive

This command is used to change the SELECT query for the view. The general syntax for altering a view is as follows:

ALTER VIEW view\_name AS select\_statement;

Where:

* + select\_statement: This is the new SELECT statement for the existing view.

Ex:

alter view hive\_view as select id, firstname from sales;

**16) Showing tables:** List tables in Hive

This command lists all the tables and views in a database. We can also use wildcards for listing specific tables. The general syntax for showing tables is as follows:

SHOW TABLES [IN database\_name] ['identifier\_with\_wildcards'];

Where:

* [IN database\_name]: Is an optional clause. This clause is used to list all the tables and views from a different database that is currently not in use.
* ['identifier\_with\_wildcards']: Is an optional clause. There can only be two wildcards used in this command: \* for any character(s) or | for a choice.

Ex:

Show tables;

Show tables in Hive\_learning;

Show tables 'Hive\*';

**17) Showing partitions:** List all the partitions in Hive

This command lists all the partitions for a table. The general syntax for is as follows:

SHOW PARTITIONS [db\_name.]table\_name [PARTITION(partition\_spec)];

Where:

* [db\_name.]: Is an optional clause. This is used to list partitions of the table from a given database.
* [PARTITION(partition\_spec)]: Is an optional clause. This is used to list a specific partition of a table.

Ex:

Show partitions Sales;

Show partitions Sales partition(dop='2015-01-01'); (Specific partition for Sales table)

Show partitions Hive\_learning. Sales partition(dop='2015-01-01'); (Same as above, but from Hive\_learning DB)

**18) Show the table properties:** List all the properties of a table in Hive

This command lists the properties of a table. The general syntax for showing table properties is as follows:

SHOW TBLPROPERTIES tblname;

Ex:

Show tblproperties Sales;

Show partitions Sales ('numFiles'); (Will list only for numFiles)

**19) Showing create table:** Create statement of a table in Hive

This command shows the CREATE TABLE statement of a table. The general syntax for showing the CREATE TABLE statement is as follows:

SHOW CREATE TABLE ([db\_name.]table\_name|view\_name);

Where:

* [db\_name.]: Is an optional clause. This is used when you want to see the CREATE TABLE statement of a table from a different database.

Ex:

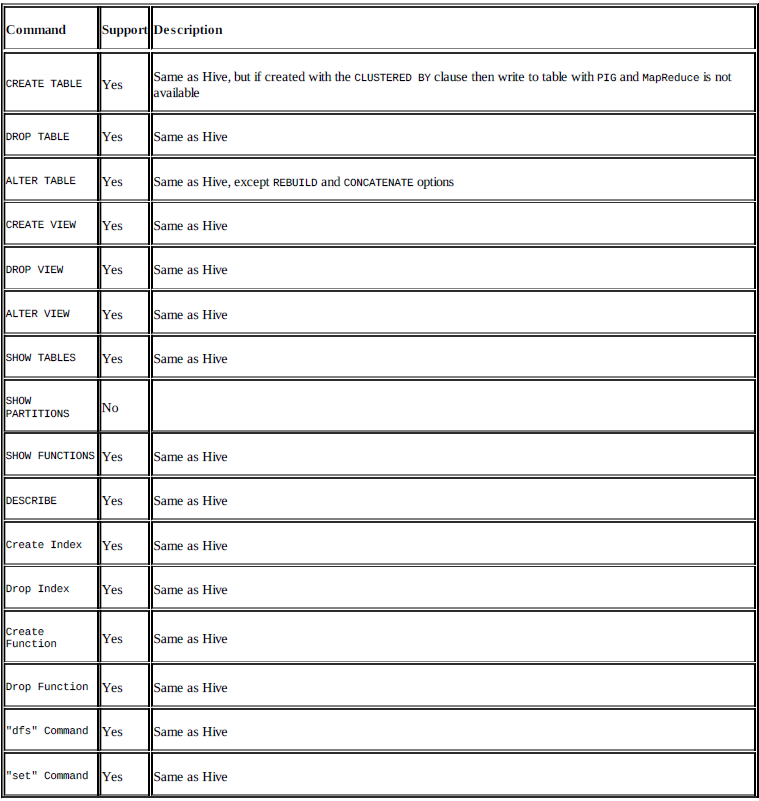
Show create table Hive\_learning.Sales;

**20) HCatalog:** Define tables in HCatalog

HCatalog is a storage management tool that enables frameworks other than Hive to leverage a data model to read and write data. HCatalog tables provide an abstraction on the data format in HDFS and allow frameworks such as PIG and MapReduce to use the data without being concerned about the data format, such as RC, ORC, and text files.

HCatInputFormat and HCatOutputFormat, which are the implementations of Hadoop InputFormat and OutputFormat, are the interfaces provided to PIG and MapReduce.

The following are the metrics of DMLs supported by HCatalog:



**21) WebHCat:** Define tables using WebHCat APIs

WebHCat, formerly called Templeton, allows access to the HCatalog service using REST APIs. Unlike HCatalog, which executed the command directly, WebHCat keeps the Hive, PIG, and MapReduce jobs in queues. The jobs can then be monitored and stopped as needed.

HCatlog resources can be accessed by REST APIs using the following URI format:

http://www.myserver.com/templeton/v1/resource.

In the preceding URL, www.myserver.com is the URL where your WebHCat is running and the resource is the HCatalog resource name.

The following is a CURL command to get all databases in Hive:

curl –s 'http://localhost:50111/templeton/v1/ddl/database?user.name=shrey'

**Hive Data Manipulations**

* Loading files into tables
* Inserting data into Hive tables from queries
* Inserting data into dynamic partitions
* Writing data into files from queries
* Enabling transactions in Hive
* Inserting values into tables from SQL
* Updating data
* Deleting data

1. **Loading files into tables:**

Loading data into a Hive table is one of the variants of inserting data into a Hive table. In this method, the entire file is copied/moved to a directory that corresponds to Hive tables. If the table is partitioned, then data is loaded into partitions one at a time. The general syntax of loading the data into a table is as follows:

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

Where:

* [LOCAL]: This is an optional clause. If this clause is specified, the preceding command will look for the file in the local filesystem.
* FILEPATH: This is the path where files reside either in the local filesystem or HDFS.
* [OVERWRITE]: Is an optional clause. If this clause is specified, the data in the table or partition is deleted and new data is loaded based on the file path in the statement.
* tablename: This is the name of the table.
* [PARTITION (partcol1=val1, partcol2=val2 ...)]: This is an optional clause for partitioned tables.

Ex:

LOAD DATA LOCAL INPATH '/tmp/sales.txt' INTO TABLE sales;

LOAD DATA INPATH '/sales.txt' INTO TABLE sales;

LOAD DATA INPATH ' /sales.txt' OVERWRITE INTO TABLE sales;

1. **Inserting data into Hive tables from queries:**

This is another variant of inserting data into a Hive table. Data can be appended into a Hive table that already contains data or can be overwritten in the Hive table or inserted into multiple tables through a single statement only.

The general format of inserting data into a table from queries is as follows:

INSERT OVERWRITE TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...) [IF NOT EXISTS]] select select\_statement FROM from\_statement;

INSERT INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)] select

select\_statement FROM from\_statement;

Where:

* tablename: This is the name of the table
* OVERWRITE: This is used to overwrite existing data in the table
* [IF NOT EXISTS]: This is an optional clause
* INTO: This is used to insert data into the Hive table. If the data is already present, new data will be appended.
* [PARTITION (partcol1=val1]: This option is used when data needs to be inserted into a partitioned table.

Ex:

INSERT INTO sales SELECT \* FROM sales\_rgn;

INSERT INTO sales SELECT \* FROM sales\_rgn WHERE state='Maryland';

INSERT OVERWRITE TABLE sales SELECT \* FROM sales\_rgn;

INSERT OVERWRITE TABLE sales SELECT \* FROM sales\_rgn WHERE id=1;

1. **Inserting data into dynamic partitions:**

This method shows how to insert data into multiple partitions through a single statement.

Dynamic partitioning is disabled by default. So we have to first enable it. The minimum configuration to enable dynamic partitioning is as follows:

SET hive.exec.dynamic.partition = true;

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

The general syntax of inserting data into multiple partitions is as follows:

FROM tablename

INSERT OVERWRITE TABLE tablename1

PARTITION(root\_partition\_name='value',child\_partition\_name)

SELECT select\_statment;

Where:

* tablename: This is the name of the table from which the value is to be taken by the select statement
* tablename1: This is the name of the table in which the data will be inserted
* root\_partition\_name: This is the static partition column
* child\_partition\_name: This is the dynamic partition column

Ex:

FROM sales\_region slr

INSERT OVERWRITE TABLE sales PARTITION(dop='2015-10-20', city) SELECT slr.id, slr.firstname, slr.lastname, slr.city;

1. **Writing data into files from queries:**

This part helps in inserting data into a file with the help of a query; that is, the output of a query to be saved into a file. The general format of inserting data into a file is as follows:

Standard syntax:

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 [ROW FORMAT row\_format] [STORED AS file\_format]SELECT select\_statment FROM from\_statment.

Hive extension (multiple inserts):

FROM from\_statement

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2] ...

Where:

* [LOCAL]: Is an optional clause. If this clause is specified, the preceding command will look for the file in the local filesystem.
* [ROW FORMAT row\_format]: Is an optional clause. With the help of this, we can specify the row format; that is, the delimiters or the fields terminated by any character.
* [STORED AS file\_format]: Is an optional clause. With the help of this clause, we can specify the file format in which we want to save the data.
* Select\_statment: This is the column in the clause will be inserted into the file.
* from\_statment: This part contains the table name along with the filter condition, if any.

Ex:

INSERT OVERWRITE LOCAL DIRECTORY '/sales'

SELECT sle.id, sle.fname, sle.lname, sle.address

FROM sales sle;

1. **Enabling transactions in Hive:**

This method shows how to configure the Hive metastore to enable **Atomicity**, **Consistency**, **Isolation**, **Durability** (**ACID**) properties for a Hive table. Insert, Update and Delete are not possible in Hive until the ACID properties are not enabled. Also table must to be Bucketed in Hive if Insert, Update and Delete feature are to be used.

To allow the user to execute transactional commands, the user needs to configure the metastore with transactional tables.

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://localhost:3306/hivedb</value>

<description>metadata is stored in a MySQL server</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

<description>MySQL JDBC driver class</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>root</value>

<description>user name for connecting to mysql server</description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>root</value>

<description>password for connecting to mysql server</description>

</property>

<property>

<name>hive.support.concurrency</name>

<value>true</value>

</property>

<property>

<name>hive.enforce.bucketing</name>

<value>true</value>

</property>

<property>

<name>hive.exec.dynamic.partition.mode</name>

<value>nonstrict</value>

</property>

<property>

<name>hive.txn.manager</name>

<value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>

</property>

<property>

<name>hive.compactor.initiator.on</name>

<value>true</value>

</property>

<property>

<name>hive.compactor.worker.threads</name>

<value>1</value>

</property>

</configuration>

Once the properties are configured in hive-site.xml, the user needs to run the following command to create metastore tables in RDBMS:

$HIVE\_HOME/bin/schematool -dbType mysql -initSchema

This command will create the transactional tables in the hivedb metastore, along with other schema tables.

1. **Inserting values into tables from SQL:**

Inserting data into a Hive table through a SQL statement is the third variant of inserting data. This is the traditional way of inserting data into a table in any RDBMS. Inserting in a table through SQL statements can only be performed if the table supports ACID. The general format of inserting data into a table is as follows:

INSERT INTO TABLE table\_name [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values\_row [, values\_row ...]

Where:

* tablename: This is the name of the table
* values\_row: This is the value that is to be inserted into the table

Ex:

INSERT INTO sales VALUES (1, 'John', 'Terry', 'H-43 Sector-23', 'Delhi', 'India','10.10.10.10', 'P\_1', '15-11-1985');

1. **Updating Data:**

Updating data in a Hive table is the traditional way of updating data in a table in any RDBMS. Updating data in a table can only be performed if the table supports **Atomicity**, **Consistency**, **Isolation**, **Durability** (**ACID**) properties.

The general format of updating data in a table is as follows:

UPDATE tablename SET column = value [, column = value ...] [WHERE expression]

Where:

* tablename: This is the name of the table
* values\_row: This is the value that is to be inserted into the table.
* WHERE expression: This is an optional clause. Only rows that match the WHERE clause will be updated

Ex:

UPDATE sales SET lname = 'Thomas' WHERE id = 1;

UPDATE sales SET ip = '20.20.20.20' WHERE id = 2;

1. **Deleting Data:**

Deleting data from a Hive table is the traditional way of deleting data in a table in any RDBMS. Deleting data in a table can only be performed if the table supports ACID properties.

DELETE FROM tablename [WHERE expression]

Where:

* tablename: This is the name of the table
* WHERE expression: This is an optional clause. Only rows that match the WHERE clause will be deleted

Ex:

DELETE FROM sales WHERE id = 1;

**HiveQL Manipulation**

* Loading Data into Managed Tables
* Inserting Data into Tables from Queries
* Dynamic Partition Inserts
* Creating Tables and Loading Them in One Query
* Exporting Data

**Loading Data into Managed Tables:**

Since Hive has no row-level insert, update, and delete operations, the only way to put data into a table is to use one of the “bulk” load operations. Or you can just write files in the correct directories by other means.

**LOAD DATA LOCAL** INPATH '${env:HOME}/california-employees'

OVERWRITE **INTO TABLE** employees

PARTITION (country = 'US', **state** = 'CA');

This command will first create the directory for the partition, if it doesn’t already exist, then copy the data.

If the target table is not partitioned, you omit the PARTITION clause.

If the LOCAL keyword is used, the path is assumed to be in the local filesystem. If LOCAL is omitted, the path is assumed to be in the distributed filesystem. In this case, the data is *moved* from the path to the final location.

If you specify the OVERWRITE keyword, any data already present in the target directory will be deleted first. Without the keyword, the new files are simply added to the target directory.

The PARTITION clause is required if the table is partitioned and you must specify a value for each partition key.

**Inserting Data into Tables from Queries:**

The INSERT statement lets you load data into a table from a query.

**INSERT** OVERWRITE **TABLE** employees

PARTITION (country = 'US', **state** = 'OR')

**SELECT** \* **FROM** staged\_employees se

**WHERE** se.cnty = 'US' **AND** se.st = 'OR';

With OVERWRITE, any previous contents of the partition (or whole table if not partitioned) are replaced.

If you drop the keyword OVERWRITE or replace it with INTO, Hive appends the data rather than replaces it.

If a record satisfied a given SELECT … WHERE … clause, it gets written to the specified table and partition. To be clear, each INSERT clause can insert into a different table, when desired, and some of those tables could be partitioned while others aren’t.

You can mix INSERT OVERWRITE clauses and INSERT INTO clauses, as well.

**Dynamic Partition Inserts:**

Hive also supports a *dynamic partition* feature, where it can infer the partitions to create based on query parameters.

Dynamic partitioning is not enabled by default. When it is enabled, it works in “strict” mode by default, where it expects at least some columns to be static. This helps protect against a badly designed query that generates a gigantic number of partitions.

**For Example:**

Here both values, country and state are dynamic as opposed to the previous example which is static in nature has a static value for the fields.

hive> **set** hive.**exec**.**dynamic**.partition=**true**;

hive> **set** hive.**exec**.**dynamic**.partition.**mode**=nonstrict;

hive> **set** hive.**exec**.**max**.**dynamic**.partitions.pernode=1000;

hive> **INSERT** OVERWRITE **TABLE** employees

> PARTITION (country, **state**)

> **SELECT** ..., se.cty, se.st

> **FROM** staged\_employees se;

**Creating Tables and Loading Them in One Query:**

You can also create a table and insert query results into it in one statement:

Ex:

**CREATE TABLE** ca\_employees

**AS SELECT** name, salary, address

**FROM** employees

**WHERE** se.**state** = 'CA';

This table contains just the name, salary, and address columns from the employee table records for employees in California. The schema for the new table is taken from the SELECT clause.

A common use for this feature is to extract a convenient subset of data from a larger, more unwieldy table.

**Exporting Data:**

How do we get data out of tables? If the data files are already formatted the way you want, then it’s simple enough to copy the directories or files:

hadoop fs -cp source\_path target\_path

Otherwise, you can use INSERT … DIRECTORY …, as in this example:

**INSERT** OVERWRITE **LOCAL** DIRECTORY '/tmp/ca\_employees'

**SELECT** name, salary, address

**FROM** employees

**WHERE** se.**state** = 'CA';

Independent of how the data is actually stored in the source table, it is written to files with all fields serialized as strings. Hive uses the same encoding in the generated output files as it uses for the tables internal storage.

Just like inserting data to tables, you can specify multiple inserts to directories:

**FROM** staged\_employees se

**INSERT** OVERWRITE DIRECTORY '/tmp/or\_employees'

**SELECT** \* **WHERE** se.cty = 'US' **and** se.st = 'OR'

**INSERT** OVERWRITE DIRECTORY '/tmp/ca\_employees'

**SELECT** \* **WHERE** se.cty = 'US' **and** se.st = 'CA'

**INSERT** OVERWRITE DIRECTORY '/tmp/il\_employees'

**SELECT** \* **WHERE** se.cty = 'US' **and** se.st = 'IL';