**Introduction**

In Apache Hive there are two types of tables , one is Managed table and second is External table known also as User Managed Tables. The main difference is how Hive will manage the metadata and the data of the table.

In this tutorial, you will learn how to:

* Create Hive User Managed Table
* Load Data into Hive User Managed Table
* Query and Drop Hive User Managed Table

To perform the HiveQL code covered in this tutorial you can create a Zeppelin note and use **hive** interpreter (**%hive**).

%hive

show databases;

**Hive Tables Types**

There are two types of tables in Hive:

1. The **Hive** Managed tables
2. The **User** Managed tables.

Each table has as metadata and data. The difference is, who will manage and what. Is it Hive who will manage both or only one the metadata?

You use an **External** table, which is a table that Hive does not manage, to import data from existing data or file on a file system, into Hive. In contrast to the Hive **Managed** table, an External table keeps its data outside the Hive metastore. Hive metastore stores only the schema metadata of the external table. Hive does not manage, or restrict access, to the actual external data.

**Creating an External Table in Hive**

When creating an External table in Hive, you need to provide the following information:

* **Name of the table** – The **create external table** command creates the table. If a table of the same name already exists in the system, this will cause an error. To avoid this, add **if not exists** to the statement. Table names are case insensitive.
* **Column names and types** – Just like table names, column names are case insensitive. Column types are values such as **int**, **char**, **string**, etc.
* **Row format** – Rows use native or custom **SerDe** (Serializer/Deserializer) formats. Native SerDe will be used if the row format is not defined, or if it is specified as delimited.
* **Field termination character** – This is a **char** type character which separates table values in a row.
* **Storage format** – You can specify storage formats such as textfile, sequencefile, jsonfile, etc.
* **Location** – This is the HDFS directory location of the file containing the table data.

The correct syntax for providing this information to Hive is:

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

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

[COMMENT 'table\_comment']

[ROW FORMAT row\_format]

[FIELDS TERMINATED BY char]

[STORED AS file\_format]

LOCATION '[location]';;

**Preparing the Dataset**

When you work with User Managed Hive Tables (External), we assume that the data is already on the storage system (e.g. HDFS). This is a very common scenario when the data is imported into HDFS or processed and stored on HDFS by another tool such as Spark, Nifi, etc..

* **Input File Description**

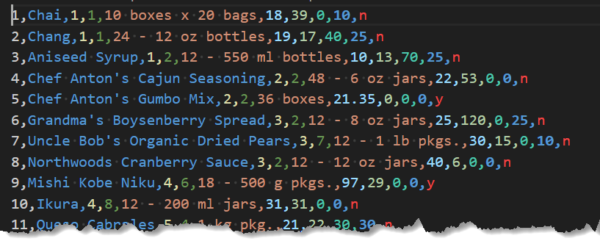
Let’s explore the products.csv file you will  load into the Hive external table.

The “products.csv“ file in located on the sandbox local file system in the **/home/training/Data/northwind** directory. The file contains data about products.

Columns description:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Column 1 | Column 2 | Column 3 | Column 4 | Column 5 | Column 6 | Column 7 | Column 8 | Column 9 | Column 10 |
| ProductID | ProductName | SupplierID | CategoryID | QuantityPerUnit | UnitPrice | UnitsInStock | UnitsOnOrder | ReorderLevel | Discontinued |

Data sample from the file:

[](http://localhost/wp-content/uploads/2023/07/products-csv.png)

* **Import the File to HDFS**

Before creating the Hive table, you need to import the products.csv file into HDFS.

1. Create an HDFS target directory. You will use this directory as an HDFS location of the file you will load into the Hive table.

2. Place the product.csv file into this directory.

%sh

# Create the HDFS target directory

# Upload the products.csv file to HDFS target directory

hdfs dfs -mkdir -p /tutorials/hive/external

hdfs dfs -put /home/training/Data/northwind/products.csv /tutorials/hive/external

You have uploaded the products.csv file to HDFS.  It is time to create the Hive table and explore the data.

**Creating User Managed Tables**

Follow the steps below to create a User Managed table (External) in Hive.

**Step 1: Create a Database**

1. Create a database named “tutorials” by running the **create database** command:

%hive

-- Create a database

create database if not exists tutorials;

Zeppelin prints a confirmation message and the time needed to perform the action.

2. Next, verify the database is created by running the **show** command:

%hive

-- verify the database is created

show databases;

**Step 2: Create The Table**

1. After you import the data file to HDFS, initiate Hive and use the syntax explained above to create an external table.

Let’s create a table whose columns identifiers will match the products.csv file you will  load into the Hive external table.

2. Create an external table schema definition that specifies columns names and typs that match data from products.csv. In this file, columns are separated by a ','.

Create the table by running the following command: (We added the *\_ext* suffix to the table name for learning purposes).

%hive

-- Create External Hive table

Create External Table tutorials.products\_ext (

ProductID bigint,

ProductName string,

SupplierID bigint,

CategoryID int,

QuantityPerUnit string,

UnitPrice double,

UnitsInStock int,

UnitsOnOrder int,

ReorderLevel int,

Discontinued char(1))

Row Format Delimited

fields terminated by ','

Stored as Textfile

Location '/tutorials/hive/external'

Zeppelin prints out a confirmation message.

3. Verify if the table is created by running the **show** command:

%hive

-- verify the table is created

show tables in tutorials;

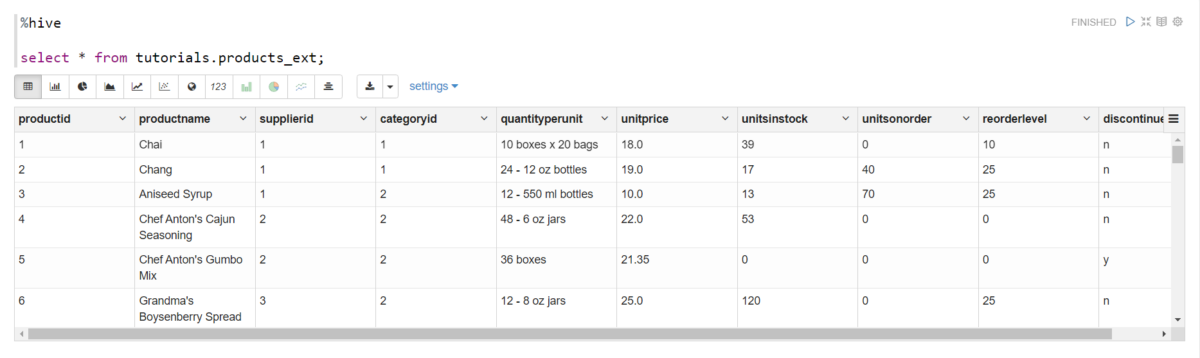
After the table creation, data is immediately available. You do **not need** to run any **load** command.

4. Verify if the data is available by running the **select**command:

%hive

-- Show the table content

select \* from tutorials.products\_ext;

[](http://localhost/wp-content/uploads/2023/07/select-products.png)

**Querying  User Managed Table**

The SELECT statement is the most commonly used command in HiveQL. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the particular record from the particular column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

* **SELECT Statement with WHERE clause**

The WHERE clause is used with SELECT statement to return only those rows from the table, which satisfy the specified condition in the query.

In HiveQL, the WHERE clause is not only used with SELECT, but it is also used with other SQL statements such as UPDATE, ALTER, and DELETE statements.

**Syntax of SELECT Statement with WHERE clause**

SELECT \* FROM Name\_of\_Table WHERE [condition];

* **Display Selected Data**

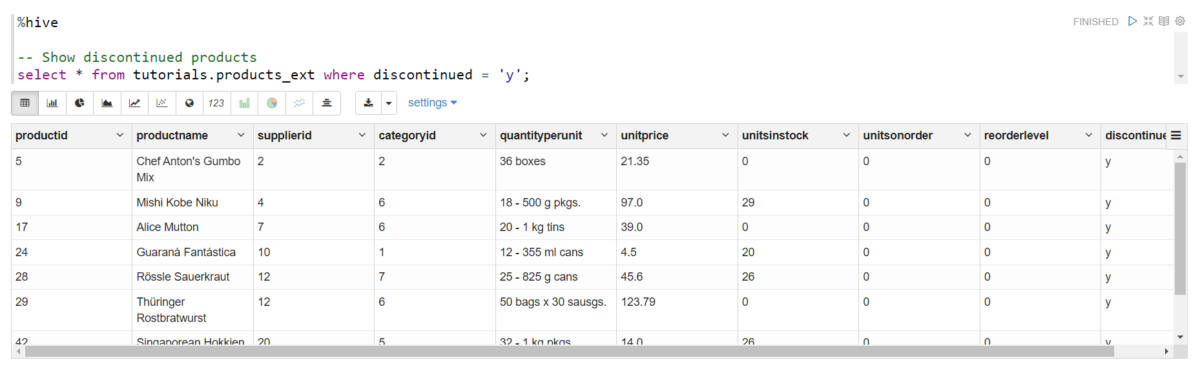
Let’s assume that you want to display discontinued products only. Select and display data by running the **select**command with **where**condition:

%hive

-- Show discontinued products

select \* from tutorials.products\_ext where discontinued = 'y';

The output contains the list of discontinued products.

[](http://localhost/wp-content/uploads/2023/07/discontinued-products.png)

Instead of the **asterisk** character which stands for “all data”, you can use more specific determiners. Replacing the asterisk with a column name (such as *ProductName*, from the example above) will show you only the data from the chosen column.

Here are some other useful query functions and their syntax:

| **Function** | **Syntax** |
| --- | --- |
| Query a table according to multiple conditions | **select \* from [table\_name] where [condition1] and [condition2];** |
| Order table data | **select [column1\_name], [column2\_name] from [table\_name] order by [column\_name];** |
| Order table data in descending order | **select [column1\_name], [column2\_name] from [table\_name] order by [column\_name] desc;** |
| Show the row count | **select count(\*) from [table\_name];** |

**Droping a User Managed Table**

When you run DROP TABLE on an external table, by default Hive drops only the metadata (schema). So after dropping an external table, the data is not gone. To retrieve it again, you issue another CREATE EXTERNAL TABLE statement to load the data from the file system.

%hive

-- Drop the table

drop table tutorials.products\_ext;

If you want the DROP TABLE command to also remove the actual data in the external table, as DROP TABLE does on a managed table, you need to configure the table properties accordingly.

Copy to Clipboard

Syntax Highlighter

%hive

-- Create External Hive table with External data purge

Create External Table tutorials.products\_ext (

ProductID bigint,

ProductName string,

SupplierID bigint,

CategoryID int,

QuantityPerUnit string,

UnitPrice double,

UnitsInStock int,

UnitsOnOrder int,

ReorderLevel int,

Discontinued char(1))

Row Format Delimited

fields terminated by ','

Stored as Textfile

Location '/tutorials/hive/external'

TBLPROPERTIES ('external.table.purge'='true');

**Summary**

In this tutorial we covered User Managed Hive tables. We covered the basics concepts. You learned how to create a User Managed table and the main difference with the second type of Hive tables (Hive Managed tables). In case of external table you need to use *External* keyword and also specify a Location when creating the table. In external table, Hive only manage meta data and data can be stored at any location in HDFS.

Open the Zeppelin note

[Hive User Managed Tables Basics](http://localhost:19995/#/notebook/2J8563TTT)