Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale. Hive Metastore (HMS) provides a central repository of metadata that can easily be analysed to make informed, data driven decisions, and therefore it is a critical component of many data lake architectures. Hive is built on top of Apache Hadoop and supports storage on S3, ADLS, GS etc though HDFS. Hive allows users to read, write, and manage petabytes of data using SQL.

Features of Hive:

- **Scalability**: Hive is designed to handle large volumes of data
- **SQL-like interface**: Hive makes it easy for SQL users to learn and use
- **Integration with Hadoop**: Hive integrates well with the Hadoop ecosystem
- **User-defined functions**: Hive has built-in functions for manipulating dates, strings, and other data

HiveServer2 (HS2) :

HS2 supports multi-client concurrency and authentication. It is designed to provide better support for open API clients like JDBC and ODBC.

Hive Metastore Server (HMS):

The Hive Metastore (HMS) is a central repository of metadata for Hive tables and partitions in a relational database, and provides clients (including Hive, Impala and Spark) access to this information using the metastore service API. It has become a building block for data lakes that utilize the diverse world of open-source software, such as Apache Spark and Presto

Hive ACID:

Hive provides full ACID support for ORC tables and insert only support to all other formats.

Now let's play with Hive QL on Databricks:

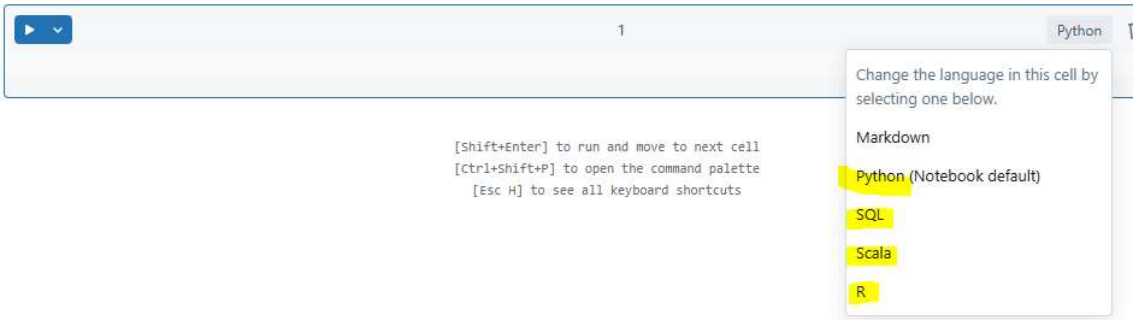
Database: Database is an organized collection of data consisting of tables, schemas, datafiles, etc.

Creation of Database:

First spin up the Spark engine and connect it to Notebook.

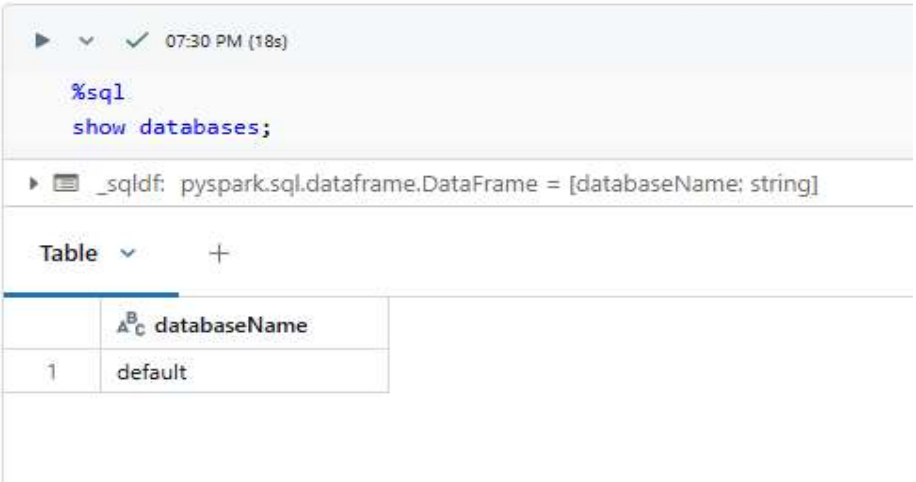
%sql is used to indicate that, in the cell we are using SQL language.

Likewise, the databricks supports Python, Scala, R and Markdown languages.



To check existing databases:

Show databases; → It will list all existing databases in the file system.



Default is one of the existing default database in a DBFS, if we did not specify any database all the queries like creation, deletion, insertion will be executed here only.

Let's create a directory for our database.

Here in Apache Hive, Database is a directory and the table also can be treated as directory and we will see that.

A new directory, Training was created. Below is the syntax for database creation:

Create database ‘name’

Location ‘location of file system’;

```
07:33 PM (2s)                                                    4
%sql
create database hive_practice
location '/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice'
_sqldf: pyspark.sql.dataframe.DataFrame
OK
```

Now list the databases and we can see that new database ‘hive_practice’ is listed.

```
07:33 PM (<1s)                                                   5
%sql
show databases
_sqldf: pyspark.sql.dataframe.DataFrame = [databaseName: string]
Table
```

	databaseName
1	default
2	hive_practice

As said, we can see that the database is created and represented as directory as below.

```
Just now (<1s)                                                   2
%sql
desc database hive_practice;
_sqldf: pyspark.sql.dataframe.DataFrame = [database_description_item: string, database_description_value: string]
Table
```

	database_description_item	database_description_value
1	Catalog Name	spark_catalog
2	Namespace Name	hive_practice
3	Comment	
4	Location	dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice
5	Owner	root

```
5 rows | 0.33s runtime
This result is stored as _sqldf and can be used in other Python cells.
```

To use the database for our transactions, we need mention use keyword as below.

```
07:34 PM (<1s)                                                    6
%sql
use database hive_practice;
_sqldf: pyspark.sql.dataframe.DataFrame
OK
```

Table: Table is a collection of rows and columns where the data is stored and organized.

There are 2 types of tables in Hive:

1. Internal tables or Managed tables
2. External tables or Unmanaged tables

Internal table: A table also called as managed table, where the metastore data will be managed by hive. And any table created with out mentioning specific database can be called as Internal. Also, if we create a table with out mentioning location, it can be treated as internal or managed table.

Features:

Hive takes the data file we load to the table to the /database-name/table-name inside our warehouse.

Internal table supports TRUNCATE command.

Internal tables also have ACID Support.

Internal tables also support query result caching means it can store the result of the already executed hive query for subsequent query.

Metadata and Table data both will be removed as soon as the table is dropped.

```
07:37 PM (2s)
%sql
create table emp_dtls
(
  id int,
  name varchar (20),
  location varchar (20)
)
row format delimited
fields terminated by ','
_sqldf: pyspark.sql.dataframe.DataFrame
OK
```

To see the tables, enter show command.

07:37 PM (<1s) 8

```
%sql
show tables;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [database: string, tableName: string ... 1 more field]

Table

	database	tableName	isTemporary
1	hive_practice	emp_dtls	false

To see the properties of a table, enter below command:

DESC FORMATTED 'Table_name';

We can see type of table is 'Managed' , also called as Internal.

And also, one directory is created inside the database directory as highlighted below,

07:38 PM (<1s) 9

```
%sql
desc formatted emp_dtls;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table

	col_name	data_type
7	Database	hive_practice
8	Table	emp_dtls
9	Owner	root
10	Created Time	Thu Mar 06 14:07:44 UTC 2025
11	Last Access	UNKNOWN
12	Created By	Spark 3.3.2
13	Type	MANAGED
14	Provider	hive
15	Table Properties	[transient_lastDdlTime=1741270064]
16	Location	dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls
17	Serde Library	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

To see, the content of table. Enter Select command. We can it is empty, as we did not loaded the data.

```
07:39 PM (3s)
%sql
select * from emp_dtls;
_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, n
```

Table

	id	name	location

There are 3 types of methods, we can load data into tables.

1. Uploading files into the table directory.
2. Loading data through file from another directory to table.
3. Through Insert command.

We can see, one by one below.

1. Uploading ',' separated data file, as we mentioned while creating the table as terminated by comma and delimited.

```
07:41 PM (<1s)                                                    11
dbutils.fs.ls('dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls')
Out[13]: []
```

I have uploaded the file, emp.txt. Having details of emp data with comma separated values.

```
07:44 PM (<1s)                                                    13
dbutils.fs.ls('dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/practice/emp_dtls/emp.txt')
Out[17]: [FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/practice/emp_dtls/emp.txt', name='emp.txt', size=96, modifi
cationTime=1741270323000)]
```

Now, lets see the select statement.

07:48 PM (4s) 17

```
%sql
select * from emp_dtls
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, name: string ... 1 more field]

Table

	id	name	location
1	1	'Pavan'	'Vijayawada'
2	2	'Naresh'	'Hyderabad'
3	3	'Srinivas'	'Guntur'
4	4	'Venkatesh'	'Hyderabad'

From above, we can see that the data populated from text file into table format. For demonstration, I have truncated table and used same dataset for all transactions.

2. Loading data through file from another directory:

For this, created a new directory as below and uploaded a emp.txt file.

07:46 PM (<1s) 15

```
dbutils.fs.mkdirs('/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound')
```

```
Out[19]: True
```

Below is the command to load the data from other directory to table, by file transfer.

Load data inpath ‘path of file from other directory’ into table ‘table_name;

07:48 PM (2s) 16

```
%sql
load data inpath 'dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/emp.txt' into table emp_dtls;
```

_sqldf: pyspark.sql.dataframe.DataFrame

OK

This result is stored as `_sqldf` and can be used in other Python cells.

Now, enter the select statement to check the data in the table.

07:48 PM (4s) 17

```
%sql
select * from emp_dtls
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, name: string ... 1 more field]

Table

	id	name	location
1	1	'Pavan'	'Vijayawada'
2	2	'Naresh'	'Hyderabad'
3	3	'Srinivas'	'Guntur'
4	4	'Venkatesh'	'Hyderabad'

3. Through insert statement: It is a normal SQL insert statement, we can insert data.

07:49 PM (3s) 18

```
%sql
insert into table emp_dtls values (5,'Kumar','Hyderabad')
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame

OK

This result is stored as `_sqldf` and can be used in other Python cells.

One row is inserted, we can see below.

07:49 PM (1s) 19

```
%sql
select * from emp_dtls;
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, name: string ... 1 more field]

Table

	id	name	location
1	1	'Pavan'	'Vijayawada'
2	2	'Naresh'	'Hyderabad'
3	3	'Srinivas'	'Guntur'
4	4	'Venkatesh'	'Hyderabad'
5	5	Kumar	Hyderabad

After inserting, four files will be created in the table directory. We can see as below.

1. Part file
2. Started file
3. Committed file
4. Success file

```
07:50 PM (<1s)                                                    20
dbutils.fs.ls('dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls')

Out[25]: [FileInfo(path='dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls/_SUCCESS', name='_SUCCESS',
size=0, modificationTime=1741270773000),
 FileInfo(path='dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls/_committed_3682650264682586446', nam
e='_committed_3682650264682586446', size=107, modificationTime=1741270772000),
 FileInfo(path='dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls/_started_3682650264682586446', name
='_started_3682650264682586446', size=0, modificationTime=1741270772000),
 FileInfo(path='dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls/emp.txt', name='emp.txt', size=96, m
odificationTime=1741270692000),
 FileInfo(path='dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls/part-00000-tid-3682650264682586446-9
5aef215-adf0-4c11-b782-73e883167856-1-1-c000', name='part-00000-tid-3682650264682586446-95aef215-adf0-4c11-b782-73e883167856-1-1-c000', size
=18, modificationTime=1741270772000)]
```

Let's see the real difference between managed and unmanaged table.

Now, executing drop command. For Internal or Managed table, drop table command will remove the table and file structure, metadata.

```
07:51 PM (2s)
%sql
drop table emp_dtls;

_sqldf: pyspark.sql.dataframe.DataFrame
OK
This result is stored as _sqldf and can be used in other Python cells.
```

Observe that below, table is deleted. Also, the data in directory structure also deleted. We can see file not found exception.

```
Last execution failed                                             22
%sql
select * from emp_dtls;

AnalysisException: [TABLE_OR_VIEW_NOT_FOUND] The table or view `emp_dtls` cannot be found. Verify the spelling and correctness of th
e schema and catalog.
If you did not qualify the name with a schema, verify the current_schema() output, or qualify the name with the correct schema and catalog...
```

```
Last execution failed                                             23
dbutils.fs.ls('dbfs:/Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls')

java.io.FileNotFoundException: /Filestore/shared_uploads/kpavankumar335@gmail.com/Training/hive_practice/emp_dtls
```

2.External tables or Unmanaged tables: These tables are called as External, as it is pointing to location while creation of table. Below are the features,

Hive won't take data to our warehouse.

The External table does not support the TRUNCATE command.

No support for ACID transaction property.

Doesn't support query result caching.

Only metadata will be removed when the External table is dropped.

Syntax:

We need to mention location, specifically to create a external table.

```
24
#creation of external table or unmanaged table
```

```
07:54 PM (<1s)                                                    25
%sql
create table emp_dtls(
  id int,
  name varchar(20),
  location varchar(20)
)
row format delimited
fields terminated by ','
location '/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/'
```

_sqldf: pyspark.sql.dataframe.DataFrame

OK

This result is stored as `_sqldf` and can be used in other Python cells.

```
07:55 PM (<1s)                                                    26
dbutils.fs.ls('/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/')
```

```
Out[32]: []
```

We can see below, the type of table is 'External' and location pointing to given path.

07:55 PM (<1s) 27

```
%sql
desc formatted emp_dtls
```

_sqldf: pyspark.sql.dataframe.DataFrame = [col_name: string, data_type: string ... 1 more field]

Table

	col_name	data_type	comment
6	Catalog	spark_catalog	
7	Database	hive_practice	
8	Table	emp_dtls	
9	Owner	root	
10	Created Time	Thu Mar 06 14:24:56 UTC 2025	
11	Last Access	UNKNOWN	
12	Created By	Spark 3.3.2	
13	Type	EXTERNAL	
14	Provider	hive	
15	Table Properties	[transient_lastDdlTime=1741271096]	
16	Location	dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbou...	
17	Serde Library	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	
18	InputFormat	org.apache.hadoop.mapred.TextInputFormat	

I have uploaded the file with details, separated by ‘,’. We can see that data is populated as below.

07:57 PM (<1s) 29

```
dbutils.fs.ls('dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound')
```

```
Out[35]: [FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/emp.txt', name='emp.txt', size=80, modificationTime
=1741271215000)]
```

07:57 PM (1s) 28

```
%sql
select * from emp_dtls;
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, name: string ... 1 more field]

Table

	id	name	location
1	1	Pavan	Vijayawada
2	2	Naresh	Hyderabad
3	3	Srinivas	Guntur
4	4	Venkatesh	Hyderabad

As usual, insert statement will create 4 files as mentioned above.

Just now (1s) 35

```
%sql
insert into emp_dtls values(5,'Sravan','Hyderbad');
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame

OK

This result is stored as `_sqldf` and can be used in other Python cells.

1 minute ago (1s) 36

```
%sql
select * from emp_dtls;
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: integer, name: string ... 1 more field]

Table

	id	name	location
1	1	Pavan	Vijayawada
2	2	Naresh	Hyderabad
3	3	Srinivas	Guntur
4	4	Venkatesh	Hyderabad
5	5	Sravan	Hyderbad

5 rows | 0.93s runtime

Just now (<1s) 37 Python

```
dbutils.fs.ls('dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/')
```

```
Out[43]: [FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_SUCCESS', name='_SUCCESS', size=0, modificationTim
e=1741274461000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_committed_6930605505481647363', name='_committed_6930605505
481647363', size=107, modificationTime=1741274460000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_started_6930605505481647363', name='_started_6930605505481647
363', size=0, modificationTime=1741274460000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/emp.txt', name='emp.txt', size=80, modificationTime=17412712
15000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/part-00000-tid-6930605505481647363-a74ae9cb-0c42-49e6-9e5b-d
77cb51a2a46-6-1-c000', name='part-00000-tid-6930605505481647363-a74ae9cb-0c42-49e6-9e5b-d77cb51a2a46-6-1-c000', size=18, modificationTime=17
41274460000)]
```

Deleting external table:

Drop command will delete the table only, the data pertaining to table schema will remain as it is and can be used for another table if needed.

```
07:58 PM (1s)

%sql
drop table emp_dtls;

_sqldf: pyspark.sql.dataframe.DataFrame
OK

This result is stored as _sqldf and can be used in other Python cells.
```

We can see that, table is deleted.

```
Last execution failed                                   31

1 %sql
2 select * from emp_dtls;

> AnalysisException: [TABLE_OR_VIEW_NOT_FOUND] The table or view `emp_dtls` cannot be found. Verify the spelling and correctness of th
e schema and catalog.
If you did not qualify the name with a schema, verify the current_schema() output, or qualify the name with the correct schema and catalog…
```

Observe that, the table data is remain there in the directory and can be used if required.

```
Just now (<1s)                                          36                                                    Python

dbutils.fs.ls('dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/')

Out[46]: [FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_SUCCESS', name='_SUCCESS', size=0, modificationTim
e=1741274461000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_committed_6930605505481647363', name='_committed_6930605505
481647363', size=107, modificationTime=1741274460000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/_started_6930605505481647363', name='_started_69306055054816
47363', size=0, modificationTime=1741274460000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/emp.txt', name='emp.txt', size=80, modificationTime=17412712
15000),
 FileInfo(path='dbfs:/FileStore/shared_uploads/kpavankumar335@gmail.com/inbound/part-00000-tid-6930605505481647363-a74ae9cb-0c42-49e6-9e5b-d
77cb51a2a46-6-1-c000', name='part-00000-tid-6930605505481647363-a74ae9cb-0c42-49e6-9e5b-d77cb51a2a46-6-1-c000', size=18, modificationTime=17
41274460000)]
```