HIVE

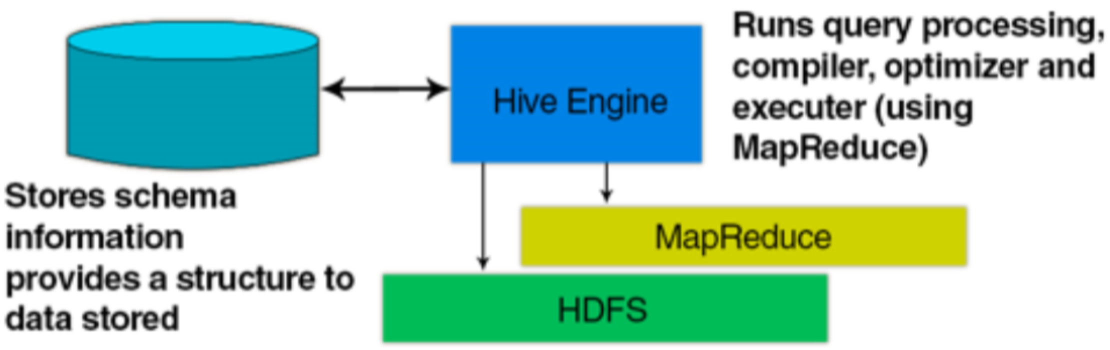
What is HIVE ?

- HIVE is a query interface on top of Hadoop's native Map-Reduce
- HIVE is a data warehouse
- HIVE allows users to write SQL style queries in a native language known as Hive Query Language (HQL)
- HIVE execution engine converts the scripts written in HQL into JAR files (map reduce) to execute in the cluster
- HIVE reads data from HDFS
- Allows creation of tables to operate on structured data
- The table’s schema information (table metadata) is saved in HIVE metastore which is borrowed from an RDBMS (Derby is default database)
- HIVE is not an RDBMS

Why HIVE ?

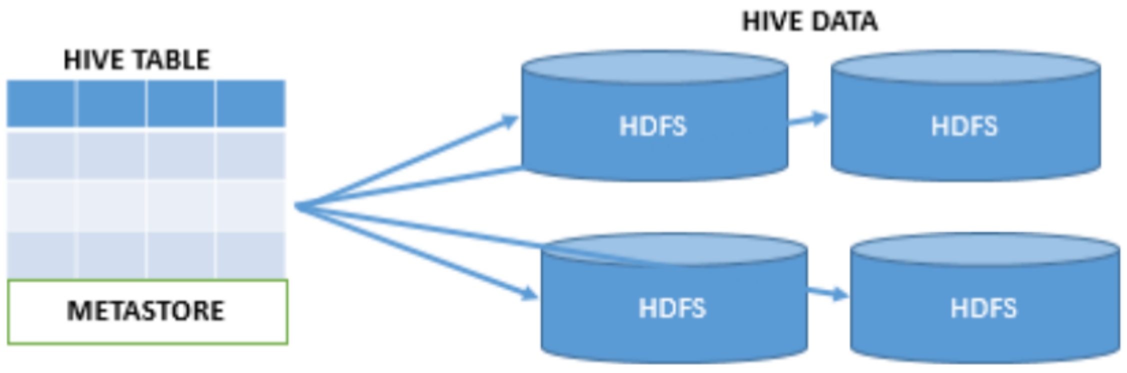
- Hadoop is known for its Map-Reduce engine for parallelizing data processing operations using HDFS as its native file storage system
- Map Reduce does not provide user friendly libraries or interfaces to deal with unstructured data handling
- Very tight dependency of JAVA if one needs to use the Map-Reduce framework
- An operation like left inner join would need around 200-300 lines of code in JAVA Map-Reduce whereas in SQL it would just be a couple of lines of code
- Analysts from SQL experience of having come from RDBMS world and DW/BI world cannot program in JAVA in order to use
- To enable SQL developers to exploit the power of Hadoop, an abstraction interface was developed on top of native Map-Reduce
- This interface (engine) was called HIVE and was officially developed by Facebook and initial release was in the year 2010

Architectural Overview

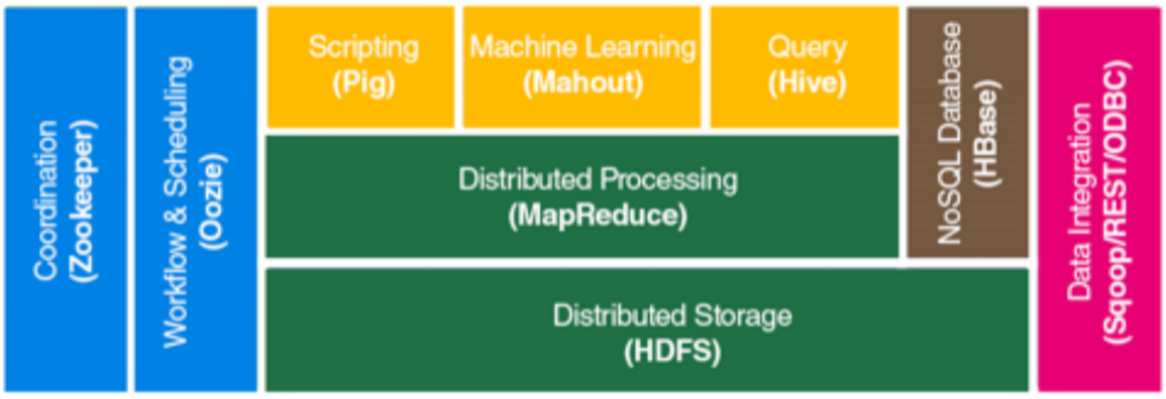

Runs query processing, compiler, optimizer and executer (using MapReduce)
Hive Engine
Stores schema information provides a structure to data stored
MapReduce
HDFS

Working of HIVE

- Hive allows a way to project a table structure on the data in HDFS (structured data in HDFS)
- The table metadata is saved separately from the data
- In reality, we do not actually load the data into the place where HIVE tables are created
- HIVE table information (table metadata is saved in meta store)



How does HIVE fit into the Hadoop ecosystem?


Coordination (Zookeeper)
Workflow & Scheduling (Oozie)
Scripting (Pig)
Machine Learning (Mahout)
Query (Hive)
NoSQL Database (HBase)
Data Integration (Sqoop/REST/ODBC)
Distributed Processing (MapReduce)
Distributed Storage (HDFS)

Things HIVE cannot do efficiently

- Ad hoc real time queries
- OLTP (Online Line Transaction Processing)
- No ACID support (ACID support is limited)
- Not suited for frequent updates and inserts (inserts and updates are allowed in recent releases of HIVE)
- Not recommended for small data sets
- Not meant for unstructured data analysis

Installation

References :
https://www.tutorialspoint.com/hive/hive_installation.htm
https://drive.google.com/drive/folders/1XdPbyAc9iWml0fPPNX91Yq3BRwkZAG2M

```
sudo apt update
sudo apt install openjdk-8-jdk -y

sudo nano ~/.bashrc
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
source ~/.bashrc

java -version; javac -version

sudo apt install openssh-server openssh-client -y
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
ssh localhost
(sudo service ssh start)

# Hadoop -
https://dlcdn.apache.org/hadoop/common/hadoop-3.2.3/hadoop-3.2.3.tar.gz
tar xzf hadoop-3.2.3.tar.gz
# put it in /usr/local/ as hadoop

# Editing 6 important files

sudo nano ~/.bashrc
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_OPTS="$HADOOP_OPTS -Djava.library.path=$HADOOP_HOME/lib/native"
source ~/.bashrc

cd $HADOOP_HOME/etc/hadoop

sudo nano core-site.xml
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

sudo nano hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>file:///home/archit/hadoopinfra/hdfs/namenode </value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>file:///home/archit/hadoopinfra/hdfs/datanode </value >
  </property>
</configuration>
```

sudo nano yarn-site.xml

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
</configuration>
```

sudo nano mapred-site.xml

```
<property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
</property>
```

# Verifying Hadoop Installation

```
cd ~
hdfs namenode -format

start-dfs.sh
start-yarn.sh
```

http://localhost:9870/

# HIVE -

https://dlcdn.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

```
tar xzf apache-hive-3.1.2-bin.tar.gz
# put it in /usr/local/ as hive

sudo nano ~/.bashrc

export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
source ~/.bashrc

sudo nano $HIVE_HOME/bin/hive-config.sh
export HADOOP_HOME=/usr/local/hadoop

# Create Hive directories in HDFS
hdfs dfs -mkdir /tmp
hdfs dfs -chmod g+w /tmp
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -chmod g+w /user/hive/warehouse

# Fixing guava problem
rm $HIVE_HOME/lib/guava-19.0.jar
cp /usr/local/hadoop/share/hadoop/hdfs/lib/guava-27.0-jre.jar $HIVE_HOME/lib/

# Initialize Derby and hive
schematool -initSchema -dbType derby

hive
```

optional Step – Edit hive-site.xml
=========================

```
cd $HIVE_HOME/conf
cp hive-default.xml.template hive-
site.xml
```

sudo nano hive-site.xml – change
metastore location to above created hdfs
path(/user/hive/warehouse)

```
# Starting HIVE:
sudo service ssh start
start-dfs.sh
start-yarn.sh
Hive
```

Basics

```
# List contents of HDFS home directory
hadoop fs -ls /

# Make directory
hadoop fs -mkdir foo

cd /home/archit/myhivedata
# Launch in the directory that we want to use
hive

# Create database
create database vin_emp;
use vin_emp;

# Create table
create table emp_global(id int,name string,city string,continent string)
row format delimited               # Data present row after row
fields terminated by ','           # Column elements separated by ,
stored as textfile;                # How HIVE internally stores the file(txt-default)

show tables;

# Load data into table
# (the txt file must be in the dir from where we launched HIVE or absolute path)
load data local inpath ‘employees.txt’ into table emp_global;
             ^ to specify data present in local FS(not HDFS, skip local)

select * from emp_global;
describe emp_global;               # show schema of table

describe extended emp_global;
# TableType:MANAGED_TABLE       # or internal tables
# (HIVE engine will take total control of data in HDFS, drop table; data in HDFS lost)
# TableType:EXTERNAL_TABLE

# Where is the database created, where the data is stored?

# def. location where HIVE engine is configured to store/create databases & tables
hadoop fs -ls /user/hive/warehouse
# this contains vin_emp.db (.db auto generated to show it is a database folder)
hadoop fs -ls /user/hive/warehouse/vin_emp.db
```

```
# this contains emp_global (table directory containing table data)

# Delete table
drop table emp_global;
# Delete database
drop database vin_emp;                  # error : vin_emp is not empty
drop database vin_emp cascade;          # drop all tables the drop database

# Create database other than default warehouse
create database vin_emp_loc location '/user/archit/myhivedata';
                                                   ^- should exist in HDFS
show databases;
```

External tables in hive

External tables point to the location in HDFS where the data is already present unlike loading the data like in managed/internal tables.

```
# copy file from local FS into HDFS location
hadoop fs -mkdir empdata            # creating a folder in HDFS home directory
hadoop fs -ls                       # check
hadoop fs -put empglobal.csv empdata
# empglobal.csv should be in the pwd & empdata is the folder
hadoop fs -cat empdata/empglobal.csv            # check file
```

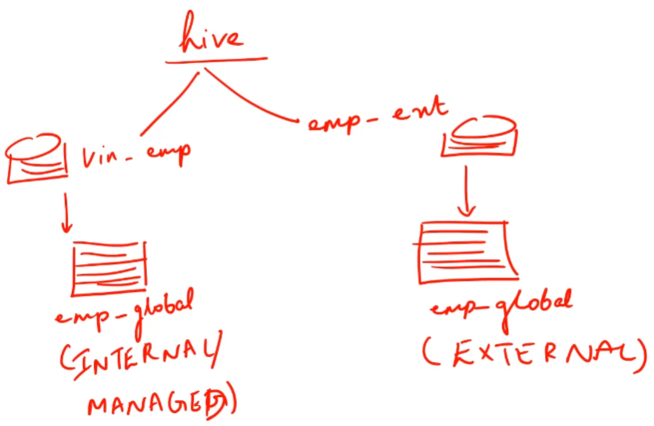
```
# create an external table
create database emp_ext location '/user/archit/myhive_ext';
hadoop fs -ls          # check created directory
```

```
create external table if not exists emp_global(id int,name string,city string,continent string)
         ^- now         ^- optional
row format delimited
fields terminated by ','
stored as textfile
location '/user/archit/empdata';
                          ^- containing empglobal.csv
                          (spec. that table(engine) must look in that HDFS folder for data)
```

NOTE: We do NOT load data in the table, instead we create tables with the same structure in which data is already present in the HDFS location and make the table point to that location. Then we can issue queries directly without loading.

```
select * from emp_global;
```

```
describe extended emp_global;
# TableType:EXTERNAL_TABLE
```



```
# Internal table
use vin_emp;
drop table emp_global;
hadoop fs -ls /user/archit/warehouse/vin_emp.db         # Empty
# corresponding HDFS gets auto deleted, HIVE takes total control of data as well

# External tables
use emp_ext;
drop table emp_global;
hadoop fs -ls empdata
# still have file empglobal.csv
```

Advantages -

1) Accidental dropping of data in HIVE will not result in the deletion of actual data present in HDFS.
2) If data belongs to some other team, we can still use data to perform queries by just creating table structure which point to that data.

Loading different file formats

HIVE supported file formats -
- RC (Row Column) FF
- ORC (Optimized Row Column) FF
- Sequence FF
- Avro FF
- Parquet FF

```
create table emp_global(id int,name string,city string,continent string)
row format delimited                # Data present row after row
fields terminated by ','            # Column elements separated by ,
stored as textfile;                 # How HIVE internally stores the file(txt-default)
```

# ORC

```
create table emp_global_orc(id int,name string,city string,continent string)
row format delimited
fields terminated by ','
stored as orcfile;
```

# Loading data into ORC table done indirectly, we run select * query (fetch data) from an existing table which is not in ORC FF (actually saved in text FF), then we try to push the result of the query into the ORC table.

```
insert into table emp_global_orc select * from emp_global;

select * from emp_global_orc;      # same data
```

# internal storage and file format varies

```
hadoop fs -ls /user/hive/warehouse/vin_emp.db/emp_global
# contains employees.txt (109 bytes)

hadoop fs -ls /user/hive/warehouse/vin_emp.db/emp_global_orc
# contains 000000_0 (644 bytes)
```

# ORC format supports ACID operations

```
hadoop fs -cat /user/hive/warehouse/vin_emp.db/emp_global_orc/000000_0
# cannot directly makes sense of what is present (looks encrypted)
```

# Sequence FF

```
create table emp_global_seq LIKE emp_global_orc stored as sequencefile;
                              ^- to clone table schema     ^- saved as
```

```
insert into table emp_global_seq select * from emp_global;

hadoop fs -ls /user/hive/warehouse/vin_emp.db/emp_global_seq
# contains 000000_0 (251 bytes)
```

<u>Avro FF</u>

```
create table emp_global_avro LIKE emp_global_orc stored as avrofile;
insert into table emp_global_avro select * from emp_global;
```

<u>Parquet FF</u>

```
create table emp_global_par LIKE emp_global_orc stored as parquetfile;
insert into table emp_global_par select * from emp_global;
```

Loading data into Hive tables

Load data into a table in HIVE, especially when data is not just in the form of rows and columns. (Complex data structure, multiple fields[array in 1 single column], structure within a column element)

<u>NOTE</u>: schema check in HIVE does take place while writing, it checks while querying. Operations on HIVE are called schema-on-read.

```
# Sample dataset (saved as /home/archit/myhivedata/siblings.txt)
Ram,43, India,Laxman
Alice,55,USA,Alex#Tim
Somu,22,India,Sham#Ram
Ted,67,Australia,Kevin#Jincy#Tanya#Kate
```

```
create table if not exists sibling_data (
name string,
age int,
country string,
siblings array<string>)
row format delimited
fields terminated by ','
collection items terminated by '#'
lines terminated by '\n'
stored as textfile;
```

```
load data local inpath '/home/archit/myhivedata/siblings.txt' into table sibling_data;
```

```
# Structure item loading into HIVE
```

```
# Sample dataset (stored as /home/archit/myhivedata/autodata.txt)
Maruthi-Suzuki,Swift,Petrol,hatchback#5#5,1197#85.00
Maruthi-Suzuki,Swift,Diesel,hatchback#5#5,1248#83.14
Ranault,Duster,Diesel,compact-suv#5#6,1498#106
```

```
create table auto_details(company string, model string, fuel string,
basic_specs struct<vehicle_type:string, doors:int, gears:int>,
engine_specs struct<cc:int, bhp:double>)
row format delimited
fields terminated by ','
collection items terminated by '#' ;
```

```
load data local inpath '/home/archit/myhivedata/autodata.txt' into table auto_details ;
```

Simple Operations on Hive tables

Alter, Add, Drop columns; rename table

```
# Rename existing table
alter table auto_details rename to auto_table;
# hadoop fs -ls /user/hive/warehouse/vin_emp.db/       (check name change)
```

```
# Rename column in table
alter table auto_details change fuel fuel_type string;
                                ^-old ^-new    ^-data type(recomm. to retain og)
```

```
# Add new column
alter table auto_details add columns (mileage double);
                                      ^- new  ^- data type
select * from auto_details;           # NULL in mileage column
```

```
# Drop existing columns
alter table auto_details replace columns(company string, model string, fuel_type string);
                                         ^- name of all columns that need to be retained
```

Query Operations on Hive tables

```
# Sample data at (/home/archit/myhivedata/empdata.txt)
emp_id,emp_name,salary,designation,department,age
564,Ram,300000,VP,Board,47
786,Bob,46000,Developer,Finance,32
987,Ravi,76000,Tech Lead,IT,43
001,Shiva,NA,CEO,Board,49
348,Aditya,100000,Project Manager,IT,46
567,Surya,6579,Intern,HR, 22
789,Sam,9877,Intern,IT,25
123,Asha,34000,HR,HR,32
121,Deepthi,45000,HR manager,HR,43
111,Krishna,40000,Manager,Infra,35
710,Soma,290600,QA,IT,26
698,Piyush,45000,QA Lead,IT,34
498,Manish,67000,Manager,Finance,48
302,Sita,200000,AVP,Board,42
002,Parvathy,NA,C00,Board,45
```

```
create database company;

create table if not exists company.empdata (
empid int,                    ^-specifying DB without ‘use’ command
empname string,
salary double,
designation string,
department string,
age int)
row format delimited
fields terminated by ','
lines terminated by '\n'
tblproperties('skip.header.line.count'='1');
  ^-builtin HIVE func      ^-specifying header column no.’s
                             (large tables can have headers in multiple[4/5] lines)
```

# Incase if the file has a header and footer which comprises of first 2 and last 2 lines of the file, then use this syntax to ignore them

```
tblproperties('skip.header.line.count'='2', 'skip.footer.line.count'='2');

load data local inpath '/home/archit/myhivedata/empdata.txt' into table empdata;
null
```

```
# Fundamental Queries -

# pick the rows of employees who work in HR department
select * from empdata where department="HR";

# pick rows of employees who work in the HR dept. & have salary more than 25000
select * from empdata where department="HR" and salary > 25000;

# select name and age of people in the HR dept. & have salary more than 25000
select empname, age from empdata where department="HR" and salary > 25000;

# pick employees in the increasing order of their salary
select * from empdata order by salary desc;

# count number of rows in the table
select count(*) from empdata;

# count number of employees in each department
select department, count(*) from empdata group by department;

# ignore columns that have NULL
select * from empdata where salary is not null;

# substring matching; pick employees whose designation is Manager/Lead
select * from empdata where designation rlike "Manager" OR designation rlike
"Lead" OR designation rlike "manager";

# average salary grouped by department
select department, avg(salary) from empdata group by department;
```

Querying complex structures

```
# query subfield of a column in case of complex structure (using auto_details)
select * from auto_details where engine_specs.bhp > 85;

# Joins

# Sample table at (/home/archit/myhivedata/employee_epf.txt)
emp_id,emp_name,EPF
001,Shiva,False
002,Parvathy,False
348,Aditya,False
710,Soma,False
498,Manish,False

create table emp_epf
(empid int,empname string,epf boolean)
row format delimited
fields terminated by ','
tblproperties('skip.header.line.count'='1');

load data local inpath '/home/archit/myhivedata/employee_epf.txt' into table emp_epf;

SET hive.auto.convert.join=false;          # To enable joins

select emp.empname, emp.salary from emp_epf pf join empdata emp on (pf.empid = emp.empid);
                                    —------------    —--------------    ^— alias —^
                                   (emp_epf as pf)  (empdata as emp)
```

```
# No. of mappers = 2 and reducers = 1 when we run it. Since both data is smaller
than block size in HDFS, if the table is spread across multiple blocks of data then
mappers > 2.

# Default join in HIVE is inner join

# Replace 'join' by the following
left outer join
right outer join
full outer join
```

Views

Pseudo tables that contain subset of a database or query which we do not want to re-run each time. They are logical tables which get automatically deleted when the parent table gets deleted.

```
# Creating a view
Create view if not exists high_sal as select * from empdata where salary > 50000;

show views;            OR             show tables;

describe formatted high_sal;
# Table Type:  VIRTUAL_VIEW

drop table empdata;                    # Deleting parent table
show tables;                           # Still shows high_sal
select * from high_sal;
# Table not found 'empdata' in definition of VIEW high_sal;
```

# Segregating managerial and non-managerial employees into 2 views

# query to create a view containing the data of employees who <u>are not</u> managers, leads, CEO, VP, AVP and CEO

```
create view regular_employees as
select empid, empname, salary, department from empdata where designation not
rlike "Manager" and designation not rlike "Lead" and designation not rlike "manager"
and designation not rlike "VP" and designation not rlike "AVP" and designation not
rlike "CEO" and designation not rlike "COO";
```

# query to create a view containing the data of employees who <u>are</u> managers, leads, CEO, VP, AVP and COO

```
create view mgmt_employees as
select empid, empname, salary, department from empdata where designation rlike
"Manager" or designation rlike "Lead" or designation rlike "manager" or designation
rlike "VP" or designation rlike "AVP" or designation rlike "CEO" or designation not
rlike "COO";
```

# we can now run any query on these virtual views treating them like tables

<u>Partitions</u>
Dividing data based on a particular column

```
# Ex: Create table organized based on continent

create table emp_global_part(id int,name string,city string, country string)
partitioned by (continent string)
row format delimited
fields terminated by ','
stored as textfile;

# Load data into partitioned table
insert into table emp_global_part partition(continent="Asia") select * from emp_global
where continent = "Asia";

insert into table emp_global_part partition(continent="North America") select * from
emp_global where continent = "North America";

insert into table emp_global_part partition(continent="European Union") select * from
emp_global where continent = "European Union";
```

PROJECT 1 : Yellow taxi trip analysis using Hive

archit.wadehra@gmail.com

Creating table

```
create table taxi(vendor_id string, pickup_datetime string, dropoff_datetime
string, passenger_count int, trip_distance DECIMAL(9,6), pickup_longitude
DECIMAL(9,6), pickup_latitude DECIMAL(9,6), rate_code int, store_and_fwd_flag
string, dropoff_longitude DECIMAL(9,6), dropoff_latitude DECIMAL(9,6),
payment_type string, fare_amount DECIMAL(9,6), extra DECIMAL(9,6), mta_tax
DECIMAL(9,6), tip_amount DECIMAL(9,6), tolls_amount DECIMAL(9,6), total_amount
DECIMAL(9,6), trip_time_in_secs int)
row format delimited
fields terminated by ','
tblproperties('skip.header.line.count'='1');
```

Loading data

```
load data local inpath '/home/archit/myhivedata/yellow_tripdata_2015 -01-06.csv'
into table taxi;
```

1) What is the total number of trips (equal to the number of rows)?

```
select count(*) from taxi;
```

10000

2) What is the total revenue generated by all the trips? The fare is stored in the column total_amount.

```
select sum(total_amount) from taxi;
```

160546.810000

3) What fraction of the total is paid for tolls? The toll is stored in tolls_amount.

```
select sum(tolls_amount) / sum(total_amount) from taxi;
```

0.0155530340341237549

4) What fraction of it is driver tips? The tip is stored in tip_amount.

```
select sum(tip_amount) / sum(total_amount) from taxi;
```

0.1078520339332808917

5) What is the average trip amount?

NOTE : No column is called "trip_amount" assuming it to be "tip_amount"

```sql
select AVG(tip_amount) from taxi;
```

1.7315300000000000000000000

6) What is the average distance of the trips? Distance is stored in the column trip_distance.

```sql
select AVG(trip_distance) from taxi;
```

3.2530330000000000000000000

7) How many different payment types are used?

```sql
select count(distinct payment_type) from taxi;
```

4

8) For each payment type, display the following details:

- Average fare generated (fare_amount)
- Average tip (tip_amount)
- Average tax (mta_tax)

```sql
select payment_type, avg(fare_amount), avg(tip_amount),
avg(mta_tax) from taxi group by payment_type;
```

PT	Average fare amount	Average tip amount	Average tax amount
1	13.5610182726846790566921755	2.7042480087459003592066219	0.4971107293456192409807903
2	11.3933830985915492957746479	0.0000000000000000000000000	0.4988732394366197183098592
3	13.2107894736842105263157895	0.0000000000000000000000000	0.4210526315789473684210526
4	12.2222222222222222222222222	0.0000000000000000000000000	0.5000000000000000000000000

9) On average which hour of the day generates the highest revenue?

```sql
select substr(pickup_datetime,12,2), avg(total_amount) from taxi
group by substr(pickup_datetime,12,2);
```

Hour	Revenue
00	15.3199724061810154525386313
22	16.2401006214856466410180527
23	16.1093846153846153846153846

22 (22nd hour generates the highest average revenue)