Big Data Analytics Docker Hive Project

This project is created on a **Docker** Image of **Apache Hive**.
Along with creating a **multinode environment** of these containers.
Showing all the commands that are required for any **naïve user** to understand the basics of **Apache HIVE**.

This project is being followed from the initialization process of starting **HIVE** containers to handle large dataset (having size of **7 GB**), each command has been delivered with a complete explanation.

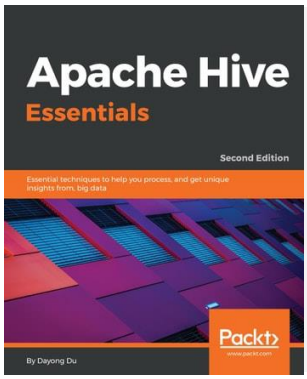
Created By
Fareed Hassan Khan
ERP 25367

IBA - MS Data Science

Table of Contents

Resources Used

This project is created using the following resources from different platforms.

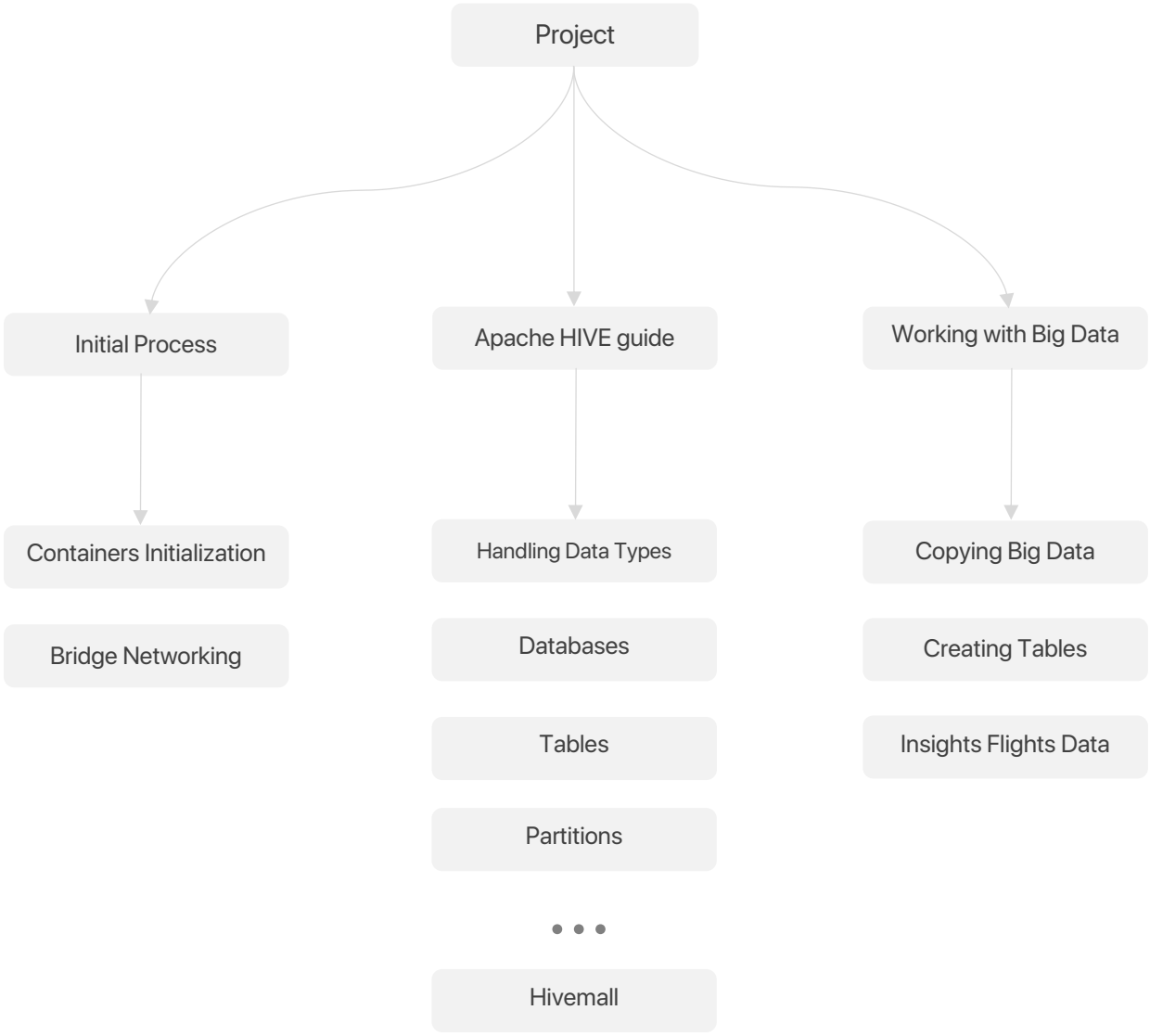


Apache Hive Essentials
Essential techniques to help you process, and get unique insights from, big data, 2nd Edition.
by **Dayong Du**

Cheat Sheet - Hive for SQL Users
Use this handy cheat sheet (based on this original MySQL cheat sheet) to get going with Hive and Hadoop
by **HortonWorks**

Project Distribution

This project is distributed in three sections.



Project Folder Structure

```
Project Folder:
|———datasets
|        employee.txt
|———datasets
|        Project Video Presentation.mp4
|        employee_id.txt
└———Report
          Project Report.pdf
          Project Report.docx
```

Dataset Description

Three datasets are used for this project.

1. employee.txt
2. employee_id.txt
3. flights dataset

employee.txt can be download from this link (download link) or you can access this file from the dataset folder. employee_id.txt can be download from this link (download link) or it is present in dataset folder.

Flights dataset is collected from Kaggle website. Following is the information about the dataset.

Element	Description
Dataset Filesize	8 Gigabytes
Dataset Shape	Sixty-two million flights
Dataset (Flights Description)	Domestic Flights (USA) 2009 - 2018
Dataset File Format	Csv
Dataset Source	Kaggle
Dataset Drive Download Link	drive.google...../view
Dataset Kaggle Download Link	Kaggle.com/...-2019

Initializing The Hive Container

Cloning GitHub Repository

```
git clone https://github.com/big-data-europe/docker-hive.git
```

Explanation: Cloning GitHub repository which contains HIVE along with HADOOP container.

Navigating to the Cloned Folder (i.e., docker-hive)

```
cd docker-hive
```

Important: Compose commands only execute if user is in docker-hive directory (Where yml file exist), otherwise it will throw error.

Composing the Docker File

```
docker-compose up –d
```

Output:

```
Creating docker-hive_hive-metastore_1                ... done
Creating docker-hive_namenode_1                      ... done
Creating docker-hive_presto-coordinator_1            ... done
Creating docker-hive_datanode_1                      ... done
Creating docker-hive_hive-server_1                   ... done
Creating docker-hive_hive-metastore-postgresql_1     ... done
```

Listing all Docker containers

```
docker ps
```

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f965945c0c67	bde:.. etastore	"entrypoint.sh /bin/..."	20 min ago	Up 20 min	0.0.0.0:1...10002/tcp	docker-hive_hive-server_1
6eafdfb92d2c	bde2... resql:2.3.0	"/docker-rypoint...."	20 min ago	Up 20 min	5432/tcp	docker-hive_hive-metastore-postgresql_1
Sasfa325235	bd/... p2.7.4-java8	"/entrypoint.sh /run..."	20 min ago	Up 20 min	0.0.0 ->50070/tcp	docker-hive_namenode_1
Eg36536221	bd/a-....7.4-java8	"/entrypoint.sh /run..."	20 min ago	Up 20 min	0.0.0...50075/tcp	docker-hive_datanode_1
Sacas325236	bd/... metastore	"entrypoint.sh /opt/..."	20 min ago	Up 20 min	10000... 10002/tcp	docker-hive_hive-metastore_1
Db3462322s	Sha ... edb:0.181	"./bin/launcher run"	20 mi ago	Up 20 min	0.0.0.... 8080/tcp	docker-hive_presto-coordinator_1

Bridge Networking of Hive Containers

List down all the networks

```
docker network ls
```

Output:

Network Id	Name	Driver	Scope
a284707fa6a2	bridge	bridge	local
651a6ad8bd97	host	host	local
150754ca8e0a	none	null	local

Creating a bridge network of HIVE containers.

```
docker network create --driver bridge  docker-hive_default
```

Output:

```
f6d9918cc9b1...  # Network ID
```

List down all the networks after creating bridge network i.e., docker-hive_default.

```
docker network ls
```

Output:

Network Id	Name	Driver	Scope
f6d9918cc9b1	docker-hive_default	bridge	local
a284707fa6a2	bridge	bridge	local
651a6ad8bd97	host	host	local
150754ca8e0a	none	null	local

Explanation: Bridge network with name of docker-hive_default has been created.

Connecting all the containers of HIVE inside docker-hive_default bridge network.

```
1. network connect docker-hive_default docker-hive_hive-metastore-postgresql_1
2. network connect docker-hive_default docker-hive_hive-server_1
3. network connect docker-hive_default docker-hive_namenode_1
4. network connect docker-hive_default docker-hive_presto-coordinator_1
5. network connect docker-hive_default docker-hive_hive-metastore_1
6. network connect docker-hive_default docker-hive_datanode_1
```

Inspecting the docker-hive_default after creating networks between the containers.

```
docker network inspect docker-hive_default
```

Output:


```
{
 "Name": "docker-hive_default",
 "Id": "f6d9918cc9b1c27b3e8e816ab121ee72750aacb8d4488ad75deed8d64742e2c7",
 "Created": "2022-05-26T01:44:41.403699376Z",

                                                    •••
 "ConfigOnly": false,
 "Containers": {
  "74acf1cae12f15245c08daa87883959058d96eec7602d566ccb16c0767fbd4e7": {
      "Name": "docker-hive_namenode_1",
      "EndpointID": "f2d5b858084a5f26710ca08f8c3b7ec1e73be2beaefaa853c82284a21a10234c",
      "MacAddress": "02:42:ac:12:00:03",
      "IPv4Address": "172.18.0.3/16",
      "IPv6Address": ""
  },
  "7e01b804e17656b348c21dbdc763fb846e833a5921a53f4eb9875a2b1ebd5b45": {
   "Name": "docker-hive_datanode_1",

                                                    •••
  },
  "964b988431d23ce7d47843425fe421697ffa5d02f2417e748f702f1730f47bc9": {
   "Name": "docker-hive_hive-metastore-postgresql_1",

                                                    •••
  },
  "a11f78febb7f574d33c02e490b822be00b9bd332b534b201a586c2c0ee78c319": {
   "Name": "docker-hive_hive-server_1",

                                                    •••
  },
  "bdb7ce12a1428367fa151c1054f373bb2bad7403eaa4421572a16573260b8084": {
   "Name": "docker-hive_hive-metastore_1",

                                                    •••
  },
  "eb0120358c26aa1d7d94777348867ac8ec70df04941e2ebee3076187cee3280a": {
   "Name": "docker-hive_presto-coordinator_1",

                                                    •••
  }
 },
 "Options": {},
 "Labels": {
  "com.docker.compose.network": "default",
  "com.docker.compose.project": "docker-hive",
  "com.docker.compose.version": "1.29.2"
 }
}
```


Explanation: Now, all the containers are being networked together under a bridge name docker-hive_default, we can use our Airlines Big Data to load on HIVE and HDFS and work with HiveSQL.

Basic Guide of Apache Hive

1. Understanding Data Types

Copy the employee.txt file from local computer and pasting it to HIVE container.

```
docker cp employee.txt docker-hive_hive-server_1:/home
```

Starting HIVE container and entering in bash environment.

```
docker-compose exec hive-server bash
```

Important: In-case the above command is not taking you in bash environment, then you must restart container using docker-compose restart command and be sure to check that you must be in docker-hive (where yml exist) folder while running any of the compose commands.

Copy employee.txt file from HIVE Container to HDFS.

```
hadoop fs -put -f /home/employee.txt /user/
```

Printing the content of employee.txt.

```
cat  /home/employee.txt
```

Output:

```
Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer^DLead
Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect
```

Access HIVE command prompt.

```
/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000/
```

Output:

```
0: jdbc:hive2://localhost:10000>
```

Creating table of employee.txt created using Create Table command.

```
CREATE TABLE employee (
    name STRING, work_place ARRAY<STRING>, gender_age STRUCT<gender:STRING,age:INT>,
    skills_score MAP<STRING,INT>, depart_title MAP<STRING,ARRAY<STRING>>
)   ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' COLLECTION ITEMS TERMINATED BY ','
     MAP KEYS TERMINATED BY ':'STORED AS TEXTFILE;
```

Loading the employee.txt data in employee table.

```
LOAD DATA INPATH '/user/employee.txt' INTO TABLE employee;
```

Describing the table employee.

```
describe employee;
```

Output:

col_name	data_type	comment
name	string	
work_place	array	
gender_age	structgender:string,age:int	
skills_score	map<string,int>	
depart_title	map<string,array>	

Query the whole array and each array element in the table.

```
1. SELECT work_place FROM employee;
2. SELECT work_place[0] as col_1, work_place[1] as col_2, work_place[2] as col_3 FROM employee;
```

Output:

work_place
["Montreal" ,"Toronto"]
["Montreal"]
["New York"]
["Vancouver"]

Execution Time: (0.02 seconds)

col_1	col_2	col_3
Montreal	Toronto	NULL
Montreal	NULL	NULL
New York	NULL	NULL
Vancouver	NULL	NULL

Execution Time: (0.8 seconds)

Query the whole struct and each struct attribute in the table.

```
1. SELECT gender_age FROM employee;
2. SELECT gender_age.gender, gender_age.gender FROM employee;
```

Output:

work_place
{"gender":"Male","age":30}
{"gender":"Male","age":35}
{"gender":"Female","age":27}
{"gender":"Female","age":57}

Execution Time: (0.018 seconds)

gender	age
Male	30
Male	35
Female	27
Female	57

Execution Time: (0.075 seconds)

Query the whole map and each map element in the table:

```
1. SELECT skills_score FROM employee;
2. SELECT name,
       skills_score['DB'] AS DB,
       skills_score['Perl'] AS Perl,
       skills_score['Python'] AS Python,
       skills_score['Sales'] AS Sales,
       skills_score['HR'] AS HR
   FROM employee;
```

Output:

skills_score
{"DB":80}
{"Perl":85}
{"Python":80}
{"Sales":89,"HR":94}

Execution Time (0.02 seconds)

name	db	perl	python	sales	hr
Michael	80	NULL	NULL	NULL	NULL
Will	NULL	85	NULL	NULL	NULL
Shelley	NULL	NULL	80	NULL	NULL
Lucy	NULL	NULL	NULL	89	94

Execution Time: (0.0120 seconds)

Query the composite type in the table.

```
1. SELECT depart_title FROM employee;
2. SELECT name, depart_title['Product'] as Product, depart_title['Test'] as Test, depart_title['COE'] as
   COE, depart_title['Sales'] as Sales FROM employee;
3. SELECT name, depart_title['Product'][0] as product_col0, depart_title['Test'][0] as test_col0
   FROM employee;
```

Output:

depart_title
{"Product":["Developer^DLead"]}
{"Product":["Lead"],"Test":["Lead"]}
{"Test":["Lead"],"COE":["Architect"]}
{"Sales":["Lead"]}

Execution Time (0.10 seconds)

name	product_col0	test_col0
Michael	Developer^DLead	NULL
Will	Lead	Lead
Shelley	NULL	Lead
Lucy	NULL	NULL

Execution Time: (0.21 seconds)

name	product	test	coe	sales
Michael	["Developer^DLead"]	NULL	NULL	NULL
Will	["Lead"]	["Lead"]	NULL	NULL
Shelley	NULL	["Lead"]	["Architect"]	NULL
Lucy	NULL	NULL	NULL	["Lead"]

Execution Time (0.41 seconds)

2. Databases

Create the database/schema if it doesn't exist.

```
1. create database myhivedatabase;
2. create schema if not exists myhivebook;
```

Create the database with the location, comments, and metadata information.

```
create database if not exists myhivedatabase
comment 'My test Database'
location '/hdfs/directory'
with dbproperties ('creator'=fareed, 'date'='2022-05-28');
```

To show the DDL, you can use show create database, this command is available since v2.1.0

```
show create database default;
```

Output:

createdb_stmt
CREATE DATABASE default
COMMENT 'Default Hive database'
LOCATION 'hdfs://namenode:8020/user/hive/warehouse'

List down all the databases.

```
show databases;
```

Output:

database_name
default
myhivebook
myhivedatabase

List down all the databases starting with letters **my**.

```
show DATABASES LIKE 'my.*';
```

Output:

database_name
myhivebook
myhivedatabase

Describing the database named default.

```
describe database default;
```

Output:

db_name	comment	location	owner_name	owner_type	parameters
default	Default Hive database	hdfs://namenode:8020/user/hive/warehouse	public	role	

Using the database name myhivebook.

```
USE myhivebook;
```

Directly qualify the table name with the database name.

```
--SELECT * FROM myhivebook.table_name;
```

Show the current database.

```
SELECT current_database();
```

Output:

_c0
default

Drop the database.

```
DROP DATABASE IF EXISTS myhivebook;
```

Explanation: The above command gets failed when database is not empty.

Drop the database with tables.

```
DROP DATABASE IF EXISTS myhivebook CASCADE;
```

Explanation: The above commands drop database and along with its tables.

Alter the database properties. The ALTER DATABASE statement can only apply to dbproperties, owner, and location on the database. The other database properties cannot be changed

```
alter database myhivebook set dbproperties ('edited-by'='fareed');
alter database myhivebook set owner user fareed;
alter database myhivebook set location '/tmp/data/myhivebook';
```

3. Tables

Printing the content of employee.txt.

```
cat  /home/employee.txt
```

Output:

```
Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer^DLead
Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect
```

Create an Internal table.

```
CREATE TABLE IF NOT EXISTS employee_internal (
    name STRING COMMENT 'this is optinal column comments',
    work_place ARRAY<STRING>,                    -- table column names are NOT case sensitive
    gender_age STRUCT<gender:STRING,age:INT>,
    skills_score MAP<STRING,INT>,                -- columns names are lower case
    depart_title MAP<STRING,ARRAY<STRING>>       -- No "," for the last column
    )
    COMMENT 'This is an internal table'          -- This is optional table comments
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'                     -- Symbol to seperate columns
    COLLECTION ITEMS TERMINATED BY ','           -- Seperate collection elements
    MAP KEYS TERMINATED BY ':'                   -- Symbol to seperate keys and values
    STORED as TEXTFILE;                          -- Table file format
```

Explanation: It is the default table in Hive. When the user creates a table in Hive without specifying it as external, then by default, an internal table gets created in a specific location in HDFS.

Create an external table.

```
CREATE EXTERNAL TABLE employee_external (   -- Use EXTERNAL keywords
   name string,
   work_place ARRAY<string>,
   gender_age STRUCT<gender:string,age:int>,
   skills_score MAP<string,int>,
   depart_title MAP<STRING,ARRAY<STRING>>
   )
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
   COLLECTION ITEMS TERMINATED BY ','
   MAP KEYS TERMINATED BY ':'
   STORED as TEXTFILE
   LOCATION '/user/myexternaldata/employee';   -- data folder location
```

Explanation: We create an external table when we want to use the data outside the Hive. Whenever we drop the external table, then only the metadata associated with the table will get deleted, the table data remains untouched by Hive.

Create a temporary table.

```
CREATE TEMPORARY TABLE IF NOT EXISTS tmp_emp1 (
  name string,
  work_place ARRAY<string>,
  gender_age STRUCT<gender:string,age:int>,
  skills_score MAP<string,int>,
  depart_title MAP<STRING,ARRAY<STRING>>
);
```

Explanation: A temporary table is only visible to the current user session. It's automatically deleted at the end of the session. The data of the temporary table is stored in the user's scratch directory, such as /tmp/hive-<username>.

Create a temporary table (second way)

```
CREATE TEMPORARY TABLE tmp_emp2 as SELECT * FROM tmp_emp1;
```

Create a temporary table (third way)

```
CREATE TEMPORARY TABLE tmp_emp3 like tmp_emp1;
```

Tables can also be created and populated by the results of a query in one statement, called Create-Table-As-Select (CTAS). CTAS has the restrictions that it cannot be a partitioned table, external table, or a list-bucketing table.

Create a table with CTAS

```
CREATE TABLE ctas_employee as SELECT * FROM employee_external;
```

Create a table with CTAS and CTE (Common Table Expression)

```
CREATE TABLE cte_employee AS WITH r1 AS
 (SELECT name
  FROM r2
  WHERE name = 'Michael' ),
                    r2 AS
 (SELECT name
  FROM employee
  WHERE gender_age.gender= 'Male' ),
                    r3 AS
 (SELECT name
  FROM employee
  WHERE gender_age.gender= 'Female' )
SELECT *
FROM r1
UNION ALL SELECT * FROM r3;
```

Printing the table that was created with CTAS and CTE

```
Select * from cte_employee;
```

Output:

cte_employee.name
Michael
Shelley
Lucy

Execution Time (0.01 seconds)

We can also use CREATE TABLE LIKE to create an empty table.

```
CREATE TABLE empty_like_employee LIKE employee_internal;
```

Explanation: This is a faster way to copy the table schema since it does not trigger any jobs but only copies metadata.

Show all tables of database

```
show tables;
```

Output:

tab_name
cte_employee
employee
employee_external
employee_internal
flights_data

Execution Time (0.09 seconds)

Show all tables of current database containing name fareed

```
show tables '*sam*';
```

Show all tables of current database containing name fareed or tariq

```
show tables '*fareed|tariq*';
```

Detailed table information for all tables starting with the expression 'employee_int'.

```
SHOW TABLE EXTENDED LIKE 'employee_int*';
```

Output:

tab_name
tableName:employee_internal
owner:root
location:hdfs://namenode:8020/... e_internal
inputformat:org.apache.ha...xtInputFormat
outputformat:org.apache...KeyTextOutputFormat
columns:struct columns { string name
partitioned:false
partitionColumns:
totalNumberFiles:0
totalFileSize:0
maxFileSize:0
minFileSize:0
lastAccessTime:0
lastUpdateTime:1653742744177

Execution Time (0.09 seconds)

Display all columns of a table.

```
show columns in employee;
```

Output:

field
name
work_place
gender_age
skills_score
depat_title

Execution Time (0.02 seconds)

Display all columns of a table. (Second way)

```
describe employee;
```

Output:

col_name	data_type	comment
name	string	
work_place	array	
gender_age	structgender:string,age:int	
skills_score	map<string,int>	
depart_title	map<string,array>	

Execution Time (0.04 seconds)

Display the DDL statements of the employee table

```
show create table employee;
```

Output:

createtab_stmt
CREATE TABLE employee(name string work_place array gender_age struct<gender:string skills_score map<string depart_title map<string ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (... 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' LOCATION 'hdfs://namenode:8020/user/hive/warehouse/employee' TBLPROPERTIES ('transient_lastDdlTime'='1653716027')

Execution Time (0.01 seconds)

Show table properties for the specified table.

```
show tblproperties employee;
```

Output:

prpt_name	prpt_value
numFiles	1
numRows	0
rawDataSize	0
totalSize	230
transient_lastDdlTime	1653716027

Execution Time (0.0212 seconds)

Dropping a table only if it exists.

```
DROP TABLE IF EXISTS empty_ctas_employee;
```

Explanation: The drop-table statement on an internal table removes the table completely and moves data to trash in the current user directory. The drop-table statement on an external table will only remove the table definition but keeps data.

Truncating a table.

```
truncate TABLE cte_employee;        -- Only apply to internal tables
```

Explanation: truncate table statement only removes data from the table. The table still exists but is empty. Note, truncate table can only apply to an internal table

Displaying a table after truncating it.

```
Select name FROM cte_employee;
```

Explanation: No data left, but empty table exists

Rename a table with the ALTER statement.

```
ALTER TABLE cte_employee RENAME TO cte_employee_backup;
```

Change the table properties with TBLPROPERTIES command.

```
ALTER TABLE employee SET TBLPROPERTIES ('comment'='New comments');
```

Change the table's file format with FILEFORMAT command.

```
ALTER TABLE employee SET FILEFORMAT RCFILE;
```

Enable/Disable the NO_DROP prevents a table from being dropped.

1. ALTER TABLE employee ENABLE NO_DROP;
2. ALTER TABLE employee DISABLE NO_DROP;

Enable/Disable the OFFLINE prevents data (not metadata) from being queried in a table.

```
1. ALTER TABLE employee ENABLE OFFLINE;
2. ALTER TABLE employee DISABLE OFFLINE;
```

Enable concatenation in an RCFile, or ORC table.

```
ALTER TABLE employee CONCATENATE;
```

Creating table employee_test

```
CREATE TABLE employee_test (
      name STRING,
      work_place ARRAY<STRING>,
      gender_age STRUCT<gender:STRING,age:INT>,
      skills_score MAP<STRING,INT>,
      depart_title MAP<STRING,ARRAY<STRING>>
    )
      ROW FORMAT DELIMITED FIELDS
      TERMINATED BY '|' COLLECTION ITEMS
      TERMINATED BY ','
      MAP KEYS TERMINATED BY ':'
      STORED AS TEXTFILE;
```

Add new columns to a table.

```
ALTER TABLE employee_test ADD COLUMNS (mynewcolumn string);
```

Verifying table after adding a new column.

```
describe employee_test;
```

Output:

col_name	data_type	comment
name	string	
work_place	array	
gender_age	structgender:string,age:int	
skills_score	map<string,int>	
depart_title	map<string,array>	
mynewcolumn	string	

Explanation: The new column does exist inside the table.

4. Partitions

Creating table name employee_partitioned with defining the PARTITIONED BY property.

```
CREATE TABLE employee_partitioned (
    name STRING, work_place ARRAY<STRING>, gender_age STRUCT<gender:STRING,age:INT>,
    skills_score MAP<STRING,INT>, depart_title MAP<STRING,ARRAY<STRING>>
)   PARTITIONED BY (year INT, month INT)  -- Use lower case partition column
    ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' COLLECTION ITEMS TERMINATED BY ','
    MAP KEYS TERMINATED BY ':';
```

Describing the table employee_partitioned.

```
describe employee_partitioned;
```

Output:

col_name	data_type	comment
name	string	
work_place	array	
gender_age	structgender:string,age:int	
skills_score	map<string,int>	
depart_title	map<string,array>	
year	int	
month	int	
	NULL	NULL
# Partition Information	NULL	NULL
# col_name	data_type	comment
	NULL	NULL
year	int	
month	int	

Execution Time (0.014 seconds)

Check Partitions.

```
SHOW PARTITIONS employee_partitioned;        -- Check partitions
```

Output:

partitions

Execution Time (0.001 seconds)

Explanation: Since no partitions has been created yet, so it shows empty table

Adding multiple partitions.

```
ALTER TABLE employee_partitioned ADD          -- Add multiple static partitions
PARTITION (YEAR=2018, MONTH=11)
PARTITION (YEAR=2018, MONTH=12);
```

Show created partitions.

```
SHOW PARTITIONS employee_partitioned;
```

Output:

Partitions
year=2018/month=11
year=2018/month=12

Execution Time (0.0112 seconds)

Dropping the partition.

```
ALTER TABLE employee_partitioned  DROP IF EXISTS PARTITION (year=2018, month=11);
```

Explanation: Drop partition will NOT remove data for external table while Drop partition will remove data with partition for internal table

Printing the partitions after the drop.

```
SHOW PARTITIONS employee_partitioned;
```

Output:

Partitions
year=2018/month=12

Execution Time (0.0212 seconds)

Renaming partitions.

```
ALTER TABLE employee_partitioned -- Rename exisiting partition
PARTITION (YEAR=2018, MONTH=12)
RENAME TO PARTITION (YEAR=2018, MONTH=03);
```

Explanation: Renaming partition name from (year=2018/month=12) to (year=2018/month=06).

Printing the partitions after update.

```
SHOW PARTITIONS employee_partitioned;
```

Output:

Partitions
year=2018/month=06

Execution Time (0.1412 seconds)

Copy employee.txt file from HIVE Container to HDFS

```
hadoop fs -put -f /home/employee.txt /user/
```

Load data into a table partition once the partition is created.

```
load data inpath '/user/employee.txt'
overwrite into table employee_partitioned
partition (year=2018, month=06);
```

Important: In-case the above command is showing you an error of employee.txt not found than you must load the file in HDFS then after going inside Apache Hive command, you can run the above command without any error.

Verifying that data has been loaded.

```
SELECT name, year, month FROM employee_partitioned;
```

Output:

name	year	month
Michael	2018	6
Will	2018	6
Shelley	2018	6
Lucy	2018	6

Execution Time (0.351 seconds)

Remove data from the partition.

```
1. TRUNCATE TABLE employee_partitioned PARTITION (year=2018, month=12);  -- For internal table
2. dfs -rm -r -f /user/employee_partitioned;       -- For external table
```

Explanation: Remove data from the partition. Note, removing data will not remove the partition information. To do a complete data cleaning, we can drop the partition discussed earlier in Dropping the Partition command.

Changing the existing partition column data type.

```
ALTER TABLE employee_partitioned PARTITION COLUMN(year string);
```

Describing the table employee_partitioned after changing the column datatype.

```
describe employee_partitioned;
```

Output:

col_name	data_type	comment
name	string	
work_place	array	
gender_age	structgender:string,age:int	
skills_score	map<string,int>	
depart_title	map<string,array>	
year	string	
month	int	
	NULL	NULL
# Partition Information	NULL	NULL
# col_name	data_type	comment
	NULL	NULL
year	int	
month	int	

Execution Time (0.014 seconds)

Changing the partition's other properties in terms of file format, location, protections, and concatenation have the same syntax to alter the table statement.

```
1. ALTER TABLE employee_partitioned PARTITION (year=2018) SET FILEFORMAT ORC;
2. ALTER TABLE employee_partitioned PARTITION (year=2018) SET LOCATION '/tmp/data';
3. ALTER TABLE employee_partitioned PARTITION (year=2018) ENABLE NO_DROP;
4. ALTER TABLE employee_partitioned PARTITION (year=2018) ENABLE OFFLINE;
5. ALTER TABLE employee_partitioned PARTITION (year=2018) DISABLE NO_DROP;
6. ALTER TABLE employee_partitioned PARTITION (year=2018) DISABLE OFFLINE;
7. ALTER TABLE employee_partitioned PARTITION (year=2018) CONCATENATE;
```

5. Buckets

Besides partition, the bucket is another technique to cluster datasets into more manageable parts to optimize query performance. Different from a partition, a bucket corresponds to segments of files in HDFS.

Copy the employee_id.txt file from local computer and pasting it to HIVE container.

```
docker cp employee_id.txt docker-hive_hive-server_1:/home
```

Starting HIVE container and entering in bash environment.

```
docker-compose exec hive-server bash
```

Important: In-case the above command is not taking you in bash environment, then you must restart container using docker-compose restart command and be sure to check that you must be in docker-hive (where yml exist) folder while running any of the compose commands.

Copy employee_id.txt file from HIVE Container to HDFS.

```
hadoop fs -put -f /home/employee_id.txt /user/
```

Access HIVE command prompt.

```
/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000/
```

Preparing table employee_id

```
CREATE TABLE employee_id (
        name STRING, employee_id INT, work_place ARRAY<STRING>, gender_age
        STRUCT<gender:STRING,age:INT>,
        skills_score MAP<STRING,INT>, depart_title MAP<STRING,ARRAY<STRING>> )
        ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' COLLECTION ITEMS TERMINATED BY ','
        MAP KEYS TERMINATED BY ':';
```

Load data into an employee_id table.

```
load data inpath '/user/employee_id.txt' into table employee_id;
```

Important: In-case the above command is showing you an error of employee_id.txt not found than you must load the file in HDFS then after going inside Apache Hive command, you can run the above command without any error.

Preparing bucket table

```
CREATE TABLE employee_id_buckets (
    name STRING,
    employee_id INT,                                    -- Use this table column as bucket column later
    work_place ARRAY<STRING>,
    gender_age STRUCT<gender:string,age:int>,
    skills_score MAP<string,int>,
    depart_title MAP<string,ARRAY<string >>
    )
    CLUSTERED BY (employee_id) INTO 2 BUCKETS       -- Support more columns
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    COLLECTION ITEMS TERMINATED BY ','
    MAP KEYS TERMINATED BY ':';
```

Output:
No rows affected, Execution Time (0.512 seconds)

Setting the maximum number of reducers to the same number of buckets specified in the table creation.

```
set map.reduce.tasks = 2;
set hive.enforce.bucketing = true;              -- This is recommended - enable enforce bucketing
```

Output:
No rows affected, Execution Time (1.273 seconds)

INSERT should be used to populate the bucket table all the time.

```
INSERT OVERWRITE TABLE employee_id_buckets SELECT * FROM employee_id;
```

Output:
No rows affected, Execution Time (2.479 seconds)

Verify the buckets in the HDFS from shell.

```
hdfs dfs -ls /user/hive/warehouse/employee_id_buckets
```

Output:

```
-rwxrwxr-x   3 root supergroup   191 2022-05-30 04:51 /user/hive/warehouse/employee_id_buckets/000000_0

-rwxrwxr-x   3 root supergroup    55 2022-05-30 04:51 /user/hive/warehouse/employee_id_buckets/000001_0
```

Explanation: Two buckets are created inside HDFS shell.

6. Views

Views are logical data structures that can be used to simplify queries by hiding the complexities, such as joins, subqueries, and filters.

Creating a VIEW from employee_id table.

```
CREATE VIEW IF NOT EXISTS employee_skills AS
    SELECT
        name, skills_score['DB'] as DB,
        skills_score['Perl'] as Perl,
        skills_score['Python'] as Python,
        skills_score['Sales'] as Sales,
        skills_score['HR'] as HR
    FROM employee_id;
```

Explanation: When creating views, there is no yarn job triggered since this is only a metadata change. However, the job will be triggered when querying the view.

Printing all the VIEWS

```
SHOW views;
```

Output:

Tab_name
employee_skills

Execution Time (0.01 seconds)

Printing all the VIEWS matching a regular expression, i.e., starting from employee_ word

```
SHOW views 'employee_*';
```

Output:

Tab_name
employee_skills

Execution Time (0.02 seconds)

Redefining the VIEW by using ALTER VIEW command

```
ALTER VIEW employee_skills as SELECT * from employee;
```

Output:
No rows affected, Execution Time (0.113 seconds)

Printing the VIEW definition

```
DESC FORMATTED employee_skills;
```

Output:

col_name	data_type	comment
# col_name	data_type	comment
	NULL	NULL
name	string	
•••	•••	•••
# Detailed Table Information	NULL	NULL
View Expanded Text:	SELECT	NULL
		employee_id.name, employee_id.skills_score['DB'] as DB,
		employee_id.skills_score['Perl'] as Perl,
		employee_id.skills_score['Python'] as Python,
		employee_id.skills_score['Sales'] as Sales,
		employee_id.skills_score['HR'] as HR
		FROM default.employee_id
View Rewrite Enabled:	No	NULL

Execution Time: (0.0251 seconds)

Dropping the VIEW.

```
DROP VIEW employee_skills;
```

Using LATERALVIEW for data

```
SELECT name, workplace FROM employee_id
LATERAL VIEW explode(work_place) wp as workplace;
```

Output:

name	workplace
Michael	Montreal
Michael	Toronto
Will	Montreal
Shelley	Montreal
Lucy	Vancouver

Execution Time: (0.0121 seconds)

Explanation: It is usually used with user-defined table-generating functions in Hive, for data normalization or processing JSON data. LateralView first applies the table-generation function to the data, and then joins the function's input and output together.

7. Performance Utilities

HQL provides the EXPLAIN and ANALYZE statements, which can be used as utilities to check and identify the performance of queries.

Hive provides an EXPLAIN statement to return a query execution plan without running the query. We can use it to analyse queries if we have concerns about their performance. The EXPLAIN statement helps us to see the difference between two or more queries for the same purpose.

EXPLAIN statement overview.

```
EXPLAIN SELECT gender_age.gender, count(*) FROM employee_partitioned WHERE year=2018
GROUP BY gender_age.gender LIMIT 2;
```

Output:

Explain
STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-0 depends on stages: Stage-1
STAGE PLANS:
Stage: Stage-1
Map Reduce
Map Operator Tree:
• • •
TableScan
alias: employee_partitioned
output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
Stage: Stage-0
Fetch Operator
limit: 2
Processor Tree:
ListSink

Execution Time: (0.0711 seconds)

Explanation: We can see that the AST section is shown as a Map/Reduce operator tree. In the STAGE DEPENDENCIES section, both Stage-0 and Stage-1 are independent root stages. In the STAGE PLANS section, Stage-1 has one map and reduce referred to by the Map Operator Tree and Reduce Operator Tree.

Inside each Map/Reduce Operator Tree section, all operators corresponding to the query keywords, as well as expressions and aggregations, are listed. The Stage-0 stage does not have map and reduce. It is just a Fetch operation

Hive statistics are a collection of data that describes more details, such as the number of rows, number of files, and raw data size of the objects in the database. Statistics are the metadata of data, collected and stored in the metastore database.

The statistics are gathered manually through the ANALYZE statement on tables, partitions, and columns.

Collect statistics on the existing table.

```
ANALYZE TABLE employee COMPUTE STATISTICS;
```

Collect statistics on the existing table with NOSCAN.

```
ANALYZE TABLE employee COMPUTE STATISTICS NOSCAN;
```

Explanation: When the NOSCAN option is specified, the command runs faster by ignoring file scanning but only collecting the number of files and their size

Check statistics in a partitioned table

```
DESCRIBE EXTENDED employee_partitioned PARTITION(year=2018, month=12);
```

Check statistics in a table

```
DESCRIBE EXTENDED employee_id;
```

Output:

```
Detailed Table Information  | Table(tableName:employee_id, dbName:default, owner:root,
createTime:1653886189, lastAccessTime:0, retention:0,
sd:StorageDescriptor(cols:[FieldSchema(name:name, type:string, comment:null),
FieldSchema(name:employee_id, type:int, comment:null), FieldSchema(name:work_place,
type:array<string>, comment:null), FieldSchema(name:gender_age, type:struct<gender:string,age:int>,
comment:null), FieldSchema(name:skills_score, type:map<string,int>, comment:null),
FieldSchema(name:depart_title
...
```

Check statistics of a specific column

```
DESCRIBE FORMATTED employee.name;
```

8. Logs

Logs provide detailed information to find out how a query/job runs. By checking the log details, we can identify runtime problems and issues that may cause bad performance.

GIT CLONE folder structure

```
D:
└──── docker-hive
  | .gitignore
  |   ...
  | saa.txt
  | startup.sh
  |   ...
  └──── conf
          ...
          hive-log4j2.properties
```

The system log contains the Hive running status and issues. It is configured in {HIVE_HOME}/conf/hive-log4j.properties.

The following three lines of the log properties can be found in the file

```
hive.root.logger=WARN,DRFA          --set logger level
hive.log.dir=/tmp/${user.name}      --set log file path
hive.log.file=hive.log              --set log file name
```

To modify the logger level, we can either modify the preceding property file that applies to all users, or set a Hive command-line config that only applies to the current user session,

```
--hiveconf hive.root.logger=DEBUG,console
```

9. Mask and encryption

For sensitive and legally protected data, such as Confidential Information.it is necessary to store data in encrypted or masked format in the filesystem.

Using HASH function.

```
SELECT name,
        -- 128 bit
        md5(name) as md5_name,
        -- 160 bit
        sha1(name) as sha1_name,
        -- 256 bit
        sha2(name, 256) as sha2_name
FROM employee_id;
```

Output:

name	md5_name	sha1_name	sha2_name
Michael	3e0...e86	f8c ... b6a	f08 ... 176
Will	2b8...1f9	3e3 ... 02a	6ce ... ae0
Shelley	e47...b61	2d4 ... d08	1e8 ... 26e
Lucy	80e...a7d	c5c ... 25b	a3f ... c4f

Execution Time: (0.1211 seconds)

Explanation: The hashed value is quite often used to secure columns, which are the unique identifiers for joining or comparing data. Built-in function, such as md5(...), sha1(...), and sha2(...), can be used for data hashing in HQL.

Masking data is quite often requested for user-sensitive data such as:

1. Credit card numbers
2. Bank account numbers
3. Passwords and much more.

Different from the hash function, the mask function in SQL can specify masking on partial data, which makes it more flexible when you want to keep part of the data unmasked for better understanding.

Using MASK function

```
SELECT
    -- big letter to U, small letter to l, number to #
    mask("Card-0123-4567-8910", "U", "l", "#") as m0,
    -- mask first n (4) values where X|x for big/small letter, n for number
    mask_first_n("Card-0123-4567-8910", 4) as m1,
    -- mask last n (4) values
    mask_last_n("Card-0123-4567-8910", 4) as m2,
    -- mask everthing except first n(4) values
    mask_show_first_n("Card-0123-4567-8910", 4) as m3,
    -- mask everthing except last n(4) values
    mask_show_last_n("Card-0123-4567-8910", 4) as m4,
    -- return a hash value - sha 256 hex
    mask_hash('Card-0123-4567-8910') as m5;
```

Output:

m0	m1	m2	m3	m4	m5
Ulll-...##	Xxxx...-8910	C-...nnnn	Ca... -nnnn	X n-...8910	f0673...ec0e7e64

Execution Time: (0.321 seconds)

Explanation: Different from the hash function, the mask function in SQL can specify masking on partial data, which makes it more flexible when you want to keep part of the data unmasked for better understanding.

Since Hive version of 1.3.0, aes_encrypt(input string/binary, key string/binary) and aes_decrypt(input binary, key string/binary) UDF have been provided to support data encryption and decryption.

Using ENCRYPTION, DECRYPTION function

```
SELECT name, aes_encrypt(name,'1234567890123456') as encrypted,
aes_decrypt(aes_encrypt(name,'1234567890123456'), '1234567890123456') as decrypted
FROM employee_id;
```

Output:

name	encrypted	decrypted
Michael	??.b??#????▲-??l	Michael
Will	?"??r{cgR?%???	Will
Shelley	??►W@???Dm?[-???	Shelley
Lucy	??/i¶???x???L?q~	Lucy

Execution Time: (0.822 seconds)

10. Hivemall

Apache Hivemall (https:/hivemall.incubator.apache.org/) is a collection of Hive UDFs for machine learning.

It contains several ML algorithm implementations across classification, regression, recommendations, loss functions, and feature engineering, all as UDFs.

This allows end users to use SQL and only SQL to apply machine learning algorithms to a large volume of training data.

Perform the following steps to set it up.

1. Download Hivemall from (https://hivemall.incubator.apache.org/download.html)
2. Copy the downloaded file from local computer and pasting it to HIVE container.

```
docker cp hivemall-all-xxx.jar docker-hive_hive-server_1:/home
```

3. Putting the downloaded file inside HDFS.

```
hdfs fs -mkdir -p /apps/hivemall
hdfs fs -put hivemall-all-xxx.jar /apps/hivemall
```

4. Create permanent functions using script here (https://github.com/apache/incubator-hivemall/blob/master/resources/ddl/define-all-as-permanent.hive)
5. Creating Database for UDFS.

```
CREATE DATABASE IF NOT EXISTS hivemall;
USE hivemall;
SET hivevar:hivemall_jar = hdfs:///apps/hivemall/hivemall-all-xxx.jar;
SOURCE define-all-as-permanent.hive;
```

6. Verify the functions are created.

```
SHOW functions "hivemall.*";
```

Working With Big Data

1. Copying Big Data to Hive Container

Copy the flights_2009_2018.csv file from local computer and pasting it to HIVE container.

```
docker cp flightsdata_data_2009_2018.csv docker-hive_hive-server_1:/home
```

Starting HIVE container and entering in bash environment.

```
docker-compose exec hive-server bash
```

Important: In-case the above command is not taking you in bash environment, then you must restart container using docker-compose restart command and be sure to check that you must be in docker-hive (where yml exist) folder while running any of the compose commands.

While in bash environment, confirming whether the file named flights_2009_2018.csv exist inside /home directory?

```
7. cd ..
8. cd home/
9. ls
```

Output:
flights_data_2009_2018.csv

Copy flights_2009_2018.csv file from HIVE Container to HDFS.

```
1. hadoop fs -put -f /home/flights_data_2009_2018.csv /user/
2. hadoop fs -ls /user/
```

Output:
Found 1 items
-rw-r--r-- 1 root supergroup 7624891161 2022-05-24 13:55 /user/flights_2009_2018.csv

Access HIVE command prompt.

```
/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000/
```

Output:
0: jdbc:hive2://localhost:10000>

Explanation: Now we can create tables of CSV file and can analyse the data.

2. Creating Tables of Flights Dataset

Creating Table of flights_2009_2018.csv using Create Table command.

```
CREATE TABLE IF NOT EXISTS flights_data (
        FL_DATE string, OP_CARRIER string, OP_CARRIER_FL_NUM int, ORIGIN string, DEST string,
        CRS_DEP_TIME int, DEP_TIME int, DEP_DELAY int, TAXI_OUT int, WHEELS_OFF int, WHEELS_ON int,
        TAXI_IN int, CRS_ARR_TIME int, ARR_TIME int, ARR_DELAY int, CANCELLED int,
        CANCELLATION_CODE string, DIVERTED int, CRS_ELAPSED_TIME int, ACTUAL_ELAPSED_TIME int,
        AIR_TIME int, DISTANCE int, CARRIER_DELAY int, WEATHER_DELAY int, NAS_DELAY int,
        SECURITY_DELAY int, LATE_AIRCRAFT_DELAY int)
    COMMENT 'flights_data'
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;
```

Loading the flights_2009_2018.csv data in flights_data table.

```
LOAD DATA INPATH '/user/flights_data_2009_2018.csv' INTO TABLE flights_data;
```

Confirming whether the flights_data table exist or not?

```
show tables;
```

Output:

tab_name
flights_data

Execution Time: (0.101 seconds)

Removing first row of table because it contains headers of CSV file.

```
ALTER TABLE flights_data SET TBLPROPERTIES ("skip.header.line.count"="1");
```

Calling the first row of table.

```
SELECT * from flights_data limit 1;
```

Output:

fl_date	op_carrier	carrier_fl_num	...	flights_data.origin	late_aircraft_delay
2009-01-01	XE	21	...	DCA	NULL

3. Insights of Flights Data

Use case: United States government decides to analyse their domestic airline services data, to improve its airlines. Currently, US has data of all airlines from 2009 to 2018. US wants to find loopholes in their flights.

List top 5 airlines names which are being cancelled the most in descending order.

```
 select
      op_carrier as airline_identifier,
      count(*) as total_flights_cancel
from
      flights_data
group by
      op_carrier
order by
      total_flights_cancel desc limit 1;
```

Output:

flights_identifier	total_flights
EV	137042
WN	133587
OO	114445
MQ	113693
AA	108555

Execution Time: (100.251 seconds)

Explanation: these are the top 5 airlines that are being cancelled the most.

Analysing the reason behind the cancellation of flights.

```
select cancellation_code, count(*) as flights_cancel from flights_data
where (cancellation_code = 'A' or cancellation_code = 'B' or cancellation_code = 'C' or
cancellation_code = 'D')
group by cancellation_code;
```

Output:

cancellation_code	flights_cancel
B	476446
A	315450
C	180605
D	708

Execution Time: (53.141 seconds)

Explanation: Code B i.e., Weather is the reason for the cancellation of most of the flights.

Since weather is the reason for the cancellation of most of the flights, analysing what part of year does flights got affected the most?

```
SELECT date_format(fl_date, 'MMMM') AS MONTH,
    count(cancelled) AS FLIGHTS_CANCEL
FROM flights_data WHERE cancelled = 1
GROUP BY date_format(fl_date, 'MMMM') ORDER BY flights_cancel desc;
```

Output:

month	flights_cancel
January	139616
February	136908
December	95190
March	92317
June	91087
August	78847
July	71662
April	64034
May	61889
September	56817
October	47058
November	37784

Execution Time: (119.311 seconds)

Explanation: January has the highest cancel flights count, which indicates that winter season of US is the one that affects the most.

Does long distance flight get delay in departure, or a short distance flight get a delay?

```
SELECT count(*) AS total_dept_delay_gret_100km
   FROM flights_data
WHERE dep_delay > 0
   AND distance > 100;

SELECT count(*) AS total_dept_delay_less _100km
   FROM flights_data
WHERE dep_delay > 0
   AND distance < 100;
```

Output:

total_dept_delay_gret_100km
18675104

Execution Time: (38.961 seconds)

total_dept_delay_less_100km
192868

Execution Time: (41.217 seconds)

Explanation: This indicates that from 2009 to 2018, ~35% of the long-distance flights get delayed in departure, while less than 5% short distance flights get delayed in departure.

Checking the average taxi in and out time based on total number of flights yearly.

```
SELECT date_format(fl_date, 'YYYY') AS YEAR,
    count(*) AS total_flights,
    round(avg(taxi_in), 2) AS avg_taxi_in,
    round(avg(taxi_out), 2) AS avg_taxi_out
FROM flights_data
GROUP BY date_format(fl_date, 'YYYY')
ORDER BY YEAR DESC;
```

Output:

year	total_flights	avg_taxi_in	avg_taxi_out
2019	37361	7.83	17.39
2018	7189106	7.6	17.41
2017	5661600	7.51	16.78
2016	5696071	7.46	16.2
2015	5802066	7.43	16.06
2014	5806236	7.12	15.67
2013	6351569	6.77	15.59
2012	6066850	6.75	15.07
2011	6174115	6.75	15.32
2010	6429420	6.92	15.63
2009	6342570	6.89	16.02

Execution Time: (93.121 seconds)

Explanation: Since 2017, average taxi in and our times have been increased, it can be due to either increase number of flights during past few years etc.

Most of the airline, that are departed with a delay, do they arrive with a delay?

```
SELECT count(*) AS total_dept_delay_gret_arr_delay
   FROM flights_data
WHERE dep_delay > 0
   AND arr_delay > 0;
```

Output:

total_dept_delay_gret_arr_delay
16027427

Execution Time: (88.961 seconds)

Explanation: This indicates that from 2009 to 2018, ~85% that are departed with a delay did arrive at their destination with a delay.

Checking whether number of delaying flights is increasing by year or not?

```
SELECT date_format(fl_date, 'YYYY') AS YEAR,
    count(cancelled) AS FLIGHTS_CANCEL
FROM flights_data WHERE cancelled = 1
GROUP BY date_format(fl_date, 'YYYY') ORDER BY YEAR desc;
```

Output:

year	flights_cancel
2019	360
2018	116308
2017	82609
2016	70204
2015	86550
2014	126701
2013	95479
2012	78669
2011	124901
2010	105609
2009	85819

Execution Time: (211.395 seconds)

Explanation: 2019 have a smaller number of flights cancel maybe due to missing data but 2018 has a large increase in number of flights as compared to previous years.

Checking which airline has been used mostly?

```
SELECT op_carrier AS flight_name,
    count(*) AS total_flights
FROM flights_data
GROUP BY op_carrier ORDER BY total_flights;
```

Output:

flight_name	total_flights
WN	12096540
DL	7841880
AA	6682161
•••	•••
YX	316090
NW	292400
G4	96221

Execution Time: (103.107 seconds)

Explanation: WN airline has been used mostly because it is the most low-cost carrier in US.

Checking how many flights have been diverted for each year?

```
SELECT date_format(fl_date, 'YYYY') AS YEAR,
    count(diverted) AS FLIGHTS_DIVERTED
FROM flights_data WHERE diverted = 1
GROUP BY date_format(fl_date, 'YYYY') ORDER BY YEAR desc;
```

Output:

year	flights_diverted
2018	17767
2010	15236
2009	15148
2015	14975
2011	14817
2014	14382
2013	14129
2016	14021
2017	12506
2012	12460
2019	116

Execution Time: (86.395 seconds)

Explanation: 2019 have a smaller number of flights diverted maybe due to missing data but 2018 has a large increase in number of flights as compared to previous years, this also indicates that flights diversion has been increasing by each year.

Which state has been the busiest state for United States?

```
SELECT origin,
    count(*) AS total_flights
FROM flights_data
GROUP BY origin
ORDER BY total_flights DESC
LIMIT 3;
```

Output:

origin	Total_flights
ATL	3903244
ORD	3001285
DFW	2546075

Explanation: Atlanta, Illinois and Texas are the busiest states for United State airline services. Most of the airline everyday departure take place from Atlanta.

References

[Hive Language Manual] (https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types)

[Hivemall User Guide) (https://hivemall.incubator.apache.org/userguide/docker/getting_started.html)

[Hivemall Video] (https://www.youtube.com/watch?v=cMUsuA9KZ_c)