

Hive Cheat Sheet

Data Definition Language

- **CREATE DATABASE:**

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name [COMMENT database_comment]
[LOCATION hdfs_path] [WITH DBPROPERTIES (property_name=property_value, ...)];
```

- **CREATE TABLE:**

```
CREATE TABLE [IF NOT EXISTS] [db_name.] table_name [(col_name data_type [COMMENT
col_comment], ... [COMMENT col_comment])] [COMMENT table_comment] [ROW FORMAT
row_format] [STORED AS file_format] [LOCATION hdfs_path];
```

- **CREATE INDEX:**

```
CREATE INDEX index_name ON table_name (column_name [ASC|DESC]) [AS index_type] [WITH
DEFERRED REBUILD] [IDXPROPERTIES (property_name=property_value, ...)];
```

- **CREATE FUNCTION:**

```
CREATE FUNCTION function_name AS 'class_name' [USING JAR 'jar_file'] [AS 'resource_uri']
[COMMENT 'function_comment'];
```

- **DESCRIBE**

- DESCRIBE: Displays the schema of a table or view in Hive

```
DESCRIBE table_name;
```

- DESCRIBE DATABASE: Displays the properties of a database in Hive.

```
DESCRIBE DATABASE database_name;
```

- DESCRIBE FUNCTION: Displays the signature of a user-defined function (UDF) in Hive.

```
DESCRIBE FUNCTION function_name;
```

- **SHOW TABLES**

```
SHOW TABLES
```

- **USE DATABASE IN HIVE**

```
USE DATABASE_NAME
```

- **DROP DATABASE IN HIVE**

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

- **DROP DATABASE IN HIVE**

```
DROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];
```

- **DROP INDEX IN HIVE**

```
DROP INDEX index_name ON table_name;
```

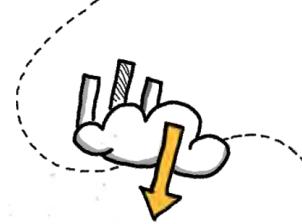
- **DROP TABLE IN HIVE**

```
DROP TABLE table_name;
```

- **TRUNCATE TABLE IN HIVE**

```
TRUNCATE TABLE table_name;
```





- **ALTER DATABASE IN HIVE**

- Changing database properties

```
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES (property_name=property_value,
...);
```

- Changing database owner

```
ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;
```

- Changing the location of a database in Hive.

```
ALTER DATABASE database_name SET LOCATION 'new_location';
```

- **ALTER TABLE IN HIVE**

- Rename table

```
ALTER TABLE table_name RENAME TO new_table_name;
```

- Add columns

```
ALTER TABLE table_name ADD COLUMNS (col_name data_type [COMMENT col_comment], ...);
```

- Replace columns

```
ALTER TABLE table_name REPLACE COLUMNS (col_name data_type [COMMENT col_comment], ...);
```

- Change column details

```
ALTER TABLE table_name CHANGE COLUMN old_col_name new_col_name new_data_type;
```

- Rename partition

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO partition_spec;
```

Data Manipulation Language

- **INSERT**

- Insert into

```
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1 FROM from_statement;
```

- Insert overwrite

```
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, ..) [IF NOT EXISTS]]
select_statement1 FROM from_statement;
```

- Insert values

```
INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES
values_row [, values_row ...];
```

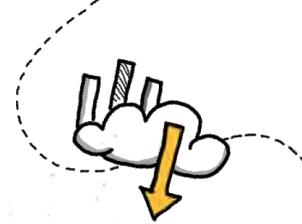
- **DELETE**

```
DELETE FROM tablename [WHERE expression];
```

- **LOAD**

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





- **UPDATE**

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

- **EXPORT**

```
EXPORT TABLE tablename [PARTITION (part_column="value"[, ...])] TO 'export_target_path'
[ FOR replication('eventid') ];
```

- **IMPORT**

```
IMPORT [[EXTERNAL] TABLE new_or_original_tablename [PARTITION (part_column="value"[,
...])]] FROM 'source_path' [LOCATION 'import_target_path'];
```

- **MERGE**

```
MERGE INTO target_table
USING source_table
ON merge_condition
WHEN MATCHED THEN
    UPDATE SET target_column = source_column
WHEN NOT MATCHED THEN
    INSERT VALUES (source_column1, source_column2, ...);
```

Hive Query Language

- **SELECT**

```
SELECT [ALL | DISTINCT] column1[, column2, ...] FROM table_name [WHERE condition];
```

- **WHERE**

```
SELECT column1[, column2, ...] FROM table_name WHERE condition;
```

- **GROUP BY**

```
SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1;
```

- **HAVING**

```
SELECT column1, aggregate_function(column2) FROM table_name GROUP BY column1 HAVING
condition;
```

- **ORDER BY**

```
SELECT column1[, column2, ...] FROM table_name ORDER BY column1 [ASC | DESC];
```

- **SORT BY**

```
SELECT column1[, column2, ...] FROM table_name SORT BY column1 [ASC | DESC];
```

- **DISTINCT**

```
SELECT DISTINCT column1[, column2, ...] FROM table_name;
```

- **UNION**

```
SELECT column_list FROM table_name1 UNION SELECT column_list FROM table_name2;
```

- **UNION ALL**

```
SELECT column_list FROM table_name1 UNION ALL SELECT column_list FROM table_name2;
```





AGGREGATE FUNCTIONS

- COUNT()

```
SELECT COUNT(*) FROM table_name;
```

```
SELECT COUNT(column_name) FROM table_name;
```

- SUM()

```
SELECT SUM(column_name) FROM table_name;
```

- AVG()

```
SELECT AVG(column_name) FROM table_name;
```

- MAX()

```
SELECT MAX(column_name) FROM table_name;
```

- MIN()

```
SELECT MIN(column_name) FROM table_name;
```

JOIN COMMANDS

- JOIN

```
SELECT * FROM table1 JOIN table2 ON table1.column = table2.column;
```

- LEFT JOIN

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
```

- RIGHT JOIN

```
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
```

- FULL OUTER JOIN

```
SELECT * FROM table1 FULL OUTER JOIN table2 ON table1.column = table2.column;
```

- CROSS JOIN

```
SELECT * FROM table1 CROSS JOIN table2;
```

Hive Built-in Functions

MATHEMATICAL FUNCTIONS

- ABS(x)
- CEIL(x)
- EXP(x)
- SQRT(x)

- POW(x,y)
- ROUND(x[,d])
- SIGN(x)

- FLOOR(x)
- LOG(x,base)
- CBRT(x):Cube root

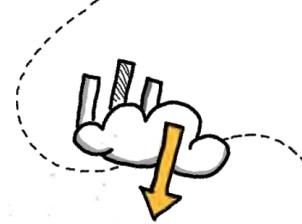
AGGREGATE FUNCTIONS

- COUNT(expr)
- SUM(expr)
- AVG(expr)
- MIN(expr)

- MAX(expr)
- STDDEV(expr)
- STDDEV_POP(expr)
- STDDEV_SAMP(expr)

- VARIANCE(expr)
- VAR_POP(expr)
- VAR_SAMP(expr)





STRING FUNCTIONS

- **CONCAT**(string str1, string str2, ...)
- **LOWER**(string str)
- **UPPER**(string str)
- **SUBSTR**(string str, int startIndex, int length)
- **REPLACE**(string str, string search, string replace)
- **REGEXP_REPLACE**(string str, string pattern, string replace)
- **CONCAT_WS**(string separator, string str1, string str2, ...) Returns the concatenation of the strings with the specified separator.

DATE FUNCTIONS

- **YEAR**(date/timestamp)
 - **MONTH**(date/timestamp)
 - **DAY**(date/timestamp)
 - **TO_DATE**(string/timestamp)
 - **DATE_ADD**(date, days)
 - **DATE_SUB**(date, days)
 - **DATEDIFF**(enddate, startdate)
 - **FROM_UNIXTIME**(unix_time, format)
- The format parameter is optional. If not specified, the default format is 'yyyy-MM-dd HH:mm:ss'.*

Hive User-defined Functions

UDFs User-Defined Functions

CREATE FUNCTION

```
CREATE [TEMPORARY] FUNCTION function_name AS class_name [USING JAR jar_file] [RESOURCE resource_file];
```

DROP FUNCTION

```
DROP FUNCTION IF EXISTS function_name;
```

DESCRIBE FUNCTION

```
DESCRIBE FUNCTION function_name;
```

USING CLAUSE

: used to load external libraries or dependencies required

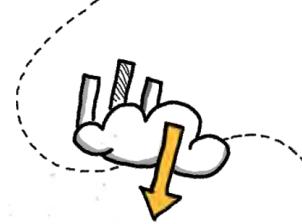
```
CREATE FUNCTION function_name AS class_name USING JAR jar_file;
```

REGISTER

```
ADD JAR jar_file;
```

```
CREATE TEMPORARY FUNCTION function_name AS class_name;
```





UDAFs User-Defined Aggregation Functions

- COUNT_DISTINCT

```
CREATE TEMPORARY FUNCTION count_distinct AS
'org.apache.hadoop.hive.ql.udf.generic.GenericUDAFCountDistinct';
SELECT count_distinct(column_name) FROM table_name;
```

- SUM_SQUARES

```
CREATE TEMPORARY FUNCTION sum_squares AS
'org.apache.hadoop.hive.ql.udf.generic.GenericUDAFSumSquares';
SELECT sum_squares(column_name) FROM table_name;
```

- CUSTOM_UDAF

```
CREATE TEMPORARY FUNCTION custom_udaf AS 'com.example.CustomUDAF';
SELECT custom_udaf(column_name) FROM table_name;
```

• Syntax of a UDAF may vary depending on the implementation of the function.

UDTFs User-Defined Table-Generating Functions

- EXPLODE()

```
SELECT explode(array_column_name) FROM table_name;
SELECT explode(map_column_name) FROM table_name;
```

- LATERAL VIEW

```
SELECT col1, col2,... FROM table_name LATERAL VIEW udtf_name(arg1, arg2,...) AS
udtf_alias_table_name;
```

- INLINE

```
SELECT col1, col2,... inline(udtf_name(arg1, arg2,...)) FROM table_name;
```

- TRANSPOSE

```
SELECT transpose(col_name) FROM table_name;
```

- BRIDGE

```
SELECT bridge(udtf_name(arg1, arg2,...)) FROM table_name;
```

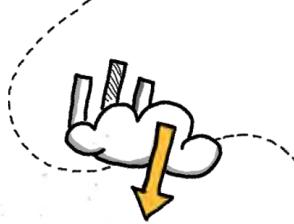
Partitioning

STATIC PARTITIONING

- CREATE A TABLE WITH STATIC PARTITIONING

```
CREATE TABLE table_name (
    column1 datatype,
    ...
)
PARTITIONED BY (partition_column1 datatype, partition_column2 datatype, ...);
```





- **INSERT DATA INTO A STATIC PARTITIONED TABLE**

```
INSERT INTO TABLE table_name PARTITION (partition_column1=value1,
partition_column2=value2, ...) VALUES (value1, value2, ..., valueN);
```

- **ADD DATA TO A STATIC PARTITIONED TABLE BY LOADING FROM A FILE**

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE table_name PARTITION
(partition_column1=value1, partition_column2=value2, ...);
```

- **QUERY DATA FROM A STATIC PARTITIONED TABLE**

```
SELECT column1, column2, ..., columnN FROM table_name WHERE partition_column1=value1 AND
partition_column2=value2 AND ...;
```

- **DESCRIBE THE PARTITIONS OF A STATIC PARTITIONED TABLE**

```
SHOW PARTITIONS table_name;
```

- **ADD A NEW PARTITION TO A STATIC PARTITIONED TABLE**

```
ALTER TABLE table_name ADD PARTITION (partition_column1=value1,
partition_column2=value2, ...);
```

- **DROP A PARTITION FROM A STATIC PARTITIONED TABLE**

```
ALTER TABLE table_name DROP PARTITION (partition_column1=value1,
partition_column2=value2, ...);
```

DYNAMIC PARTITIONING

- **ENABLE DYNAMIC PARTITIONING**

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

- **DISABLE DYNAMIC PARTITIONING**

```
SET hive.exec.dynamic.partition=false;
```

Cost-based optimization

- **SET THE COST-BASED OPTIMIZER TO ON**

```
SET hive.cbo.enable=true;
```

- **SET THE COST-BASED OPTIMIZER TO OFF**

```
SET hive.cbo.enable=false;
```

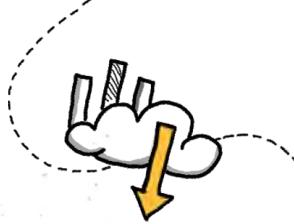
- **SET THE COST MODEL VERSION TO USE FOR THE OPTIMIZER**

```
SET hive.stats.dbclass=<class_name>;
```

- **COLLECT STATISTICS FOR A TABLE OR PARTITION**

```
ANALYZE TABLE <table_name> [PARTITION(<partition_spec>)] COMPUTE STATISTICS [noscan];
```





- DISPLAY STATISTICS FOR A TABLE OR PARTITION

```
ANALYZE TABLE <table_name> [PARTITION(<partition_spec>)] COMPUTE STATISTICS noscan;
```

- DROP STATISTICS FOR A TABLE OR PARTITION

```
ANALYZE TABLE <table_name> [PARTITION(<partition_spec>)] DROP STATISTICS;
```

- SHOW THE COST-BASED OPTIMIZER CONFIGURATION

```
SET hive.cbo.showcase.compilation.rule.name;
```

- SHOW THE OPTIMIZED LOGICAL PLAN FOR A QUERY

```
EXPLAIN LOGICAL <query>;
```

- SHOW THE OPTIMIZED PHYSICAL PLAN FOR A QUERY

```
EXPLAIN <query>;
```

- SHOW THE COST OF A QUERY PLAN

```
EXPLAIN COST <query>;
```

Bucketed Map Joins

- SET THE REQUIRED PROPERTIES FOR BUCKET MAP JOIN

```
set hive.optimize.bucketmapjoin = true;  
set hive.enforce.bucketing = true;
```

- CREATE THE BUCKETED TABLES

- TABLE1

```
CREATE TABLE IF NOT EXISTS table1 (  
    emp_id int,  
    emp_name string,  
    emp_city string,  
    gender string  
) CLUSTERED BY (emp_id) INTO 8 BUCKETS  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

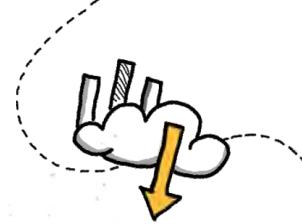
- TABLE2

```
CREATE TABLE IF NOT EXISTS table2 (  
    emp_id int,  
    emp_dept string,  
    emp_salary int  
) CLUSTERED BY (emp_id) INTO 8 BUCKETS  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

- PERFORM THE BUCKET MAP JOIN

```
SELECT /*+ MAPJOIN(table2) */ table1.* , table2.emp_dept, table2.emp_salary  
FROM table1  
JOIN table2 ON table1.emp_id = table2.emp_id;
```





Hive Storage Formats

- **ENABLE TEZ EXECUTION ENGINE**

```
CREATE TABLE table_name (column1 data_type, column2 data_type, ...)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

- **SEQUENCE FILE FORMAT**

```
CREATE TABLE table_name (column1 data_type, column2 data_type, ...)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.SequenceFileSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.SequenceFileInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat';
```

- **RCFILE FORMAT**

```
CREATE TABLE table_name (column1 data_type, column2 data_type, ...) STORED AS RCFILE;
```

- **ORC FILE FORMAT**

```
CREATE TABLE table_name (column1 data_type, column2 data_type, ...) STORED AS ORC;
```

- **PARQUET FILE FORMAT**

```
CREATE TABLE table_name (column1 data_type, column2 data_type, ...) STORED AS PARQUET;
```

- **AVRO FILE FORMAT**

```
CREATE TABLE table_name
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat';
```

Hive Security

AUTHENTICATION

- **HIVE AUTHENTICATION WITH LDAP**

1. To enable LDAP authentication in Hive, set the following properties in `hive-site.xml`:

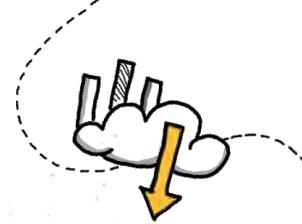
- `hive.server2.authentication.ldap.url`: LDAP server URL.
- `hive.server2.authentication.ldap.Domain`: Domain name to be used.
- `hive.server2.authentication.ldap.groupDNPattern`: DN pattern for group membership
- `hive.server2.authentication.ldap.groupMembershipKey`: Group membership attribute

2. Start HiveServer2 with the LDAP authentication option:

- `hive --service hiveserver2 --hiveconf hive.server2.authentication=LDAP`

3. Users can then authenticate using their LDAP username and password.





• HIVE AUTHENTICATION WITH KERBEROS

1. To enable Kerberos authentication in Hive, set the following properties in `hive-site.xml`:

- `hive.server2.authentication.kerberos.principal`: Kerberos principal for HiveServer2.
- `hive.server2.authentication.kerberos.keytab`: Path to the keytab file for HiveServer2.
- `hive.server2.authentication.kerberos.name.rules`: Kerberos name rules.

2. Start HiveServer2 with the Kerberos authentication option:

```
hive --service hiveserver2 --hiveconf hive.server2.authentication=KERBEROS
```

3. Users can then authenticate using their Kerberos credentials.

• For both LDAP and Kerberos authentication, make sure that all the required dependencies are installed and properly configured.

AUTHORIZATION

• HIVE AUTHORIZATION SQL STANDARD-BASED

• **CREATE ROLE**: Creates a new role in Hive

```
CREATE ROLE role_name;
```

• **DROP ROLE**: Drops an existing role in Hive

```
DROP ROLE role_name;
```

• **GRANT**: Grants privileges on an object to a role or user

```
GRANT [privilege_type] ON [object_type] [object_name] TO [role_name/user_name];
```

Example:

```
GRANT SELECT ON TABLE table_name TO role_name;
```

```
GRANT INSERT, UPDATE ON DATABASE db_name TO user_name;
```

• **REVOKE**: Revokes privileges on an object from a role or user

```
REVOKE [privilege_type] ON [object_type] [object_name] FROM [role_name/user_name];
```

Example:

```
REVOKE SELECT ON TABLE table_name FROM role_name;
```

```
REVOKE INSERT, UPDATE ON DATABASE db_name FROM user_name;
```

• **SHOW GRANT**: Displays the privileges granted to a role or user on an object

```
SHOW GRANT [privilege_type] ON [object_type] [object_name] TO [role_name/user_name];
```

Example:

```
SHOW GRANT SELECT ON TABLE table_name TO role_name;
```

```
SHOW GRANT INSERT, UPDATE ON DATABASE db_name TO user_name;
```





How to use ChatGPT Effectively for Hive Workflow

Python Program to Integrate ChatGPT and HIVE

- CONNECT TO HIVE

```
from pyhive import hive
conn = hive.Connection(host="YOUR_HIVE_HOST", port=PORT, username="YOUR_USERNAME",
password="YOUR_PASSWORD")
```

- SET UP OPENAI API FOR CHATGPT

```
import openai
openai.api_key = 'YOUR_OPENAI_KEY'
```

- DEFINE HELPER FUNCTION TO EXECUTE HIVE QUERIES

```
def execute_hive_query(sql):
    cursor = conn.cursor()
    cursor.execute(sql)
    return cursor.fetchall()
```

- DEFINE A FUNCTION TO GENERATE SQL QUERIES USING CHATGPT

```
def generate_sql(prompt):
    response = openai.Completion.create(
        engine="text-davinci-002",
        prompt=prompt + "\nSQL:",
        temperature=0.5,
        max_tokens=60
    )
    return response.choices[0].text.strip()
```

- USE THESE FUNCTIONS TOGETHER TO PERFORM DATA ENGINEERING TASKS IN HIVE

```
#Use ChatGPT to generate a SQL query
prompt = "Find the total number of employees in each department."
sql = generate_sql(prompt)
results = execute_hive_query(sql) #Execute the query in Hive
#Process the results...
```

- EXAMPLE USE CASE: DATA CLEANING IN HIVE WITH CHATGPT

```
#Use ChatGPT to generate a SQL query to calculate the average age
prompt = "Calculate the average value of the column 'age' in the table 'employees'."
sql = generate_sql(prompt)
average_age_results = execute_hive_query(sql) #Execute the query in Hive to get the avg age
average_age = average_age_results[0][0] #Assuming the result is a single row with one column
#Use ChatGPT to generate a SQL query to fill missing
```

