CDP One

Running SQL Queries

Date published: 2022-06-03
Date modified: 2022-08-15

Legal Notice

Contents

Overview

CDP One provides two SQL engines for querying your data: Hive and Impala. For running simple queries, these SQL engines are very similar. You can quickly master using either engine if you are familiar with SQL. You see the minor differences in query examples, and can easily choose the one that suits your needs.

You can start Hive or Impala from the command line of the cluster or Hue to run queries.

SQL tables in CDP

You need to understand the relationship of table types to ACID properties to run SQL queries on your data. The location of a table depends on the table type. You might choose a table type based on its supported storage format.

You can create ACID (atomic, consistent, isolated, and durable) tables for unlimited transactions or for insert-only transactions. These tables are managed tables. Alternatively, you can create an external table for non-transactional use. Because control of the external table is weak, the external table is not ACID compliant.

The following matrix includes the types of tables you can create, SQL engine support, ACID property support, required storage format, and key SQL operations (INSERT, UPDATE, and DELETE).

Table Type	Engine	ACID	File Format	INSERT	UPDATE/DELETE
Managed: CRUD transactional	Hive	Yes	ORC	Yes	Yes
Managed: Insert-only transactional	Hive, Impala	Yes	Any	Yes	No
External	Hive, Impala	No	Any	Yes	No

Although you cannot use the SQL UPDATE or DELETE statements to delete data in some types of tables, you can use DROP PARTITION on any table type to delete the data.

Transactional tables

Transactional tables are ACID tables that reside in the Hive warehouse. To achieve ACID compliance, Hive has to manage the table, including access to the table data. Only through Hive and Impala can you access and change the data in managed tables. Because the engine has full control of managed tables, the engine can optimize these tables extensively.

CDP is designed to support a relatively low rate of transactions, as opposed to serving as an online analytical processing (OLAP) system. You can use the SHOW TRANSACTIONS command to list open and aborted transactions.

Transactional tables are on a par with non-ACID tables performance-wise. No bucketing or sorting is required. Bucketing does not affect performance. These tables are compatible with native cloud storage.

Hive supports one statement per transaction, which can include any number of rows, partitions, or tables.

External tables

External table data created from Hive or Impala is not owned or controlled by Hive. You typically use an external table when you want to create the table based on data located in the object store. You can create an external table based on a PostgreSQL table, and use the table from Hive or Impala. You can create an external table based on a text file, such as CSV (comma separated values). You can use Hue to put the Postgres table or CSV file on S3.

The following capabilities are not supported for external tables:

- Query cache

- Materialized views, except in a limited way
- Automatic runtime filtering
- File merging after insert

When you run DROP TABLE on an external table, by default only the metadata (schema) is dropped. If you want the DROP TABLE command to also remove the actual data in the external table, as DROP TABLE does on a managed table, you need to set the external.table.purge property to true.

ORC vs Parquet formats

The differences between Optimized Row Columnar (ORC) file format for storing data in SQL engines are important to understand. Query performance improves when you use the appropriate format for your application.

The following table compares SQL engine support for ORC and Parquet.

Table 1:

Capability	ORC	Parquet	SQL Engine
Read external data	#	#	Hive, Impala
Write Full ACID tables	#		Hive
Read Full ACID tables	#		Hive, Impala
Read Insert-only managed tables	#	#	Impala
Column index	#	#	Hive
Column index		#	Impala
CBO uses column metadata	#		Hive
Recommended format	#		Hive
Recommended format		#	Impala
Read complex types	#	#	Impala
Read/write complex types	#	#	Hive

The major differences between Hive and Impala are as follows:

- Default file format for table creation

  Hive=ORC; Impala=Parquet
- Support for CRUD ACID tables

  Hive only, ORC storage format only
- Support for insert-only ACID tables

  Hive and Impala, in all storage formats

In Hive, if you accept the default, or explicitly specify ORC storage, you get an ACID table with insert, update, and delete (CRUD) capabilities. If you specify any other storage type for a managed table, such as text, CSV, AVRO, or JSON, you get an insert-only ACID table. You cannot update or delete columns in the insert-only table.

SQL table locations

You need a little information about the location of your SQL tables in CDP. The location depends on the table type that you can determine by running a query.

Hive metastore properties hive.metastore.warehouse.dir and hive.metastore.warehouse.external.dir set the storage locations for Hive and Impala tables. For example:

- hive.metastore.warehouse.external.dir = s3a://bucketName/warehouse/tablespace/external/hive
- hive.metastore.warehouse.dir=s3a://bucketName/warehouse/tablespace/managed/hive

Managed tables reside in the managed tablespace, which only Hive can access. By default, external tables reside in the external tablespace.

To determine the managed or external table type, you can run the DESCRIBE EXTENDED table_name command.

Creating a CRUD transactional table in Hive

Hive supports creating a CRUD (Create, Read, Update, Delete) transactional table having ACID (atomic, consistent, isolated, and durable) properties. A CRUD table is a managed table that you can update, delete, and merge. You learn by example how to determine the table type.

About this task

In this task, you create a CRUD transactional table. You cannot sort this type of table. To create a CRUD transactional table, you must accept the default ORC format by not specifying any storage during table creation, or by specifying ORC storage explicitly.

Impala does not support creating a CRUD transactional table.

Procedure

1. Start Hive.
2. Enter your user name and password.
   The Hive 3 connection message, followed by the Hive prompt for entering SQL queries on the command line, appears.
3. Create a CRUD transactional table named T having two integer columns, a and b:
   Hive example:

   ```
   CREATE TABLE T(a int, b int);
   ```
4. Confirm that you created a managed, ACID table.
   Hive example:

   ```
   DESCRIBE FORMATTED T;
   ```

   The output looks something like this:

   ```
   +-------------------------------+------------------------------------
   -------------+----------------------------------------------------+
   |           col_name            |                     data_type
              |                    comment                        |
   +-------------------------------+------------------------------------
   -------------+----------------------------------------------------+
   | a                             | int
              |                                                    |
   | b                             | int
              |                                                    |
   |                               | NULL
              | NULL                                               |
   | # Detailed Table Information  | NULL
              | NULL                                               |
   | Database:                     | default
              | NULL                                               |
   | OwnerType:                    | USER
              | NULL                                               |
   ```

```
| Owner:                          | max
          | NULL                                               |
| CreateTime:                     | Fri Jul 22 22:04:34 UTC 2022
            | NULL                                             |
| LastAccessTime:                 | UNKNOWN
            | NULL                                             |
| Retention:                      | 0
            | NULL                                             |
| Location:                       | s3a://cdpsaasdemo-cdp-private-default-
1ogvplm/warehouse/tablespace/managed/hive/t | NULL                 |
| Table Type:                     | MANAGED_TABLE
            | NULL                                             |
...
|                                 | totalSize
            | 0                                                |
|                                 | transactional
            | true
```

The table type says MANAGED_TABLE and transactional = true.

Creating an insert-only transactional table

You can create a transactional table using any storage format if you do not require update and delete capability. This type of table has ACID properties, is a managed table, and accepts insert operations only.

About this task

In this task, you create an insert-only transactional table for storing data in the default storage format of your SQL engine.. In Hive, setting 'transactional'='true','transactional_properties'='insert_only' in table properties when creating the table is required; otherwise, a CRUD table results. In Impala, these table properties are set by default..

In Hive, the default storage format is ORC (Optimized Row Columnar). In Impala, the default storage format is Parquet. Use the STORED AS clause is optional if you want to specify a different format.

Procedure

1. Start Hive or Impala.
2. Enter your user name and password.
3. Create an insert-only transactional table in the default storage file format having two integer columns, a and b: Hive example:

```
CREATE TABLE T2(a int, b int)
  TBLPROPERTIES ('transactional'='true','transactional_properties'='inser
t_only');
```

Impala example:

```
CREATE TABLE T3 (a int, b int);
```

By default, Impala creates an insert-only table when you run a CREATE TABLE statement to create a managed table. Specifying the 'transactional'='true' 'transactional_properties'='insert_only' table property is optional.

Creating an external table

You see how to put your source data on the S3 object store using Hue. From the Hue file browser, you copy the location of the data to create the LOCATION clause in the CREATE EXTERNAL TABLE statement. From Hive or Impala, you run the SQL query to create an external table from Hive or Impala from source data.

About this task

In this task, you create a CSV file of data and upload data to the CDP One object store S3. You run the CREATE TABLE statement. The metadata is stored in the metastore in the warehouse. After uploading a file to S3, delete it when you are finished to reduce cost.

Before you begin

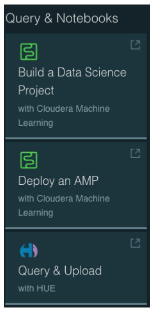
Set up Hadoop SQL policies in Ranger to include S3 URLs.

Procedure

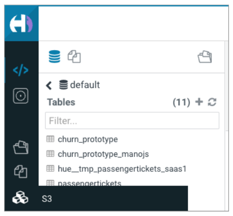
1. Create a text file that contains CSV (comma-separated values) data.
For example, create a file called students.csv that has the following lines of text:

```
1,jane,doe,senior,mathematics
2,john,smith,junior,engineering
```

2. Log into CDP One, and click All Services Query & Notebooks Upload .

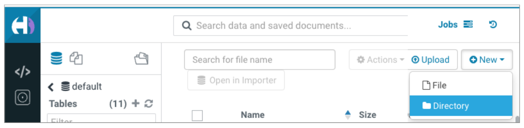

Query & Notebooks
Build a Data Science Project
with Cloudera Machine Learning
Deploy an AMP
with Cloudera Machine Learning
Query & Upload
with HUE

3. In the Hue Editor, click S3.



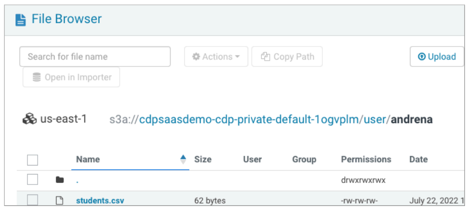
4. In the Hue File Browser, click Upload, browse for, and select the CSV file you created.

5. Click New Directory , and create a directory named andrena.



6. Drag the students.csv file to the andrena directory.

7. Click the andrena directory, check that the CSV is in place, and copy the path.
   For example, copy s3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena.



s3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena

Name	Size	User	Group	Permissions	Date
.				drwxrwxrwx	
students.csv	62 bytes			-rw-rw-rw-	July 22, 2022 1

8. Create an external table in the default storage format of the SQL engine you are using.
   Hive or Impala example:

```
CREATE EXTERNAL TABLE IF NOT EXISTS names_text(
student_ID INT,
FirstName STRING,
LastName STRING,
year STRING,
Major STRING)
COMMENT 'Student Names' ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION 's3a://cdpsaasdemo-cdp-private-default-1ogvplm
/user/andrena';
```

   From Hive, data is stored in ORC (Optimized Row Columnar) format by default. From Impala, data is stored in Parquet format.

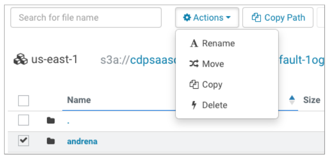
9. Query the table to see the data.

```
SELECT * FROM names_text;
```

   Output looks something like this:

```
+------------------------+-----------------------+---------------------
+------------------+-------------------+
| names_text.student_id  | names_text.firstname  | names_text.lastname  |
 names_text.year  | names_text.major  |
+------------------------+-----------------------+---------------------+-
-----------------+------------------+
| 1                      | jane                  | doe                  |
 senior           | mathematics       |
| 2                      | john                  | smith                |
 junior           | engineering       |
+------------------------+-----------------------+---------------------
+------------------+-------------------+
```

10. In the Hue File Browser, select the file you uploaded, and click Actions Delete



Dropping an external table and data

When you run DROP TABLE on an external table, by default Hive and Impala drop only the metadata (schema). If you want the DROP TABLE command to also remove the actual data in the external table, as DROP TABLE does on a managed table, you need to configure the table properties accordingly.

Before you begin

Set up Hadoop SQL policies in Ranger to include S3 URLs.

Procedure

1. Create a CSV file of data you want to query.
2. Start Hive or Impala.
3. Create an external table to store the CSV data, configuring the table so you can drop it along with the data.
   Hive and Impala example:

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS names_text2(
     a INT, b STRING)
     ROW FORMAT DELIMITED
     FIELDS TERMINATED BY ','
     STORED AS TEXTFILE
     LOCATION 's3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena'
     TBLPROPERTIES ('external.table.purge'='true');
   ```
4. Run DROP TABLE on the external table.
   Hive and Impala example:

   ```
   DROP TABLE names_text2;
   ```

   The table is removed from Hive Metastore and the data stored externally is also removed. For example, names_text is removed from the Hive Metastore and the CSV file that stored the data is also deleted from the file system.
5. Prevent data in the external table from being deleted by a DROP TABLE statement.
   Hive and Impala example:

   ```
   ALTER TABLE addresses_text SET TBLPROPERTIES ('external.table.purge'='fa
   lse');
   ```

Partitioned tables

Partitioning organizes table data and can improve query performance of low volume data. You can query slices of the data instead of scanning the entire table.

You should avoid creating many small partitions. You can partition managed and external tables. You use the PARTITIONED BY clause to create a partitioned table and follow step-by-step instructions to insert data into the partitions. You can put files, such as CSV (comma-separated-values) files, that contain the data in directories that represent partitions and create external tables based on the CSV data.

Under certain conditions, you must manually repair metadata about Hive or Impala partitions that resides in the metastore to keep changes to partitions in sync with the metadata. You learn when and how to do this.

Creating and loading a managed, partitioned table

You use the PARTITIONED BY clause to create a partitioned table. To insert data into particular partitions in the table, you use a familiar ANSI SQL statement that indentifies the data for each partition. A simple example shows you have to accomplish this basic task.

Procedure

1. **Create a partitioned table.**
   Hive and Impala example:

   ```
   CREATE TABLE IF NOT EXISTS pageviews (userid VARCHAR(64), link STRING, o
   rigin STRING) PARTITIONED BY (datestamp STRING);
   ```

2. **Insert data into the table.**

   Assign null values to columns you do not want to assign a value.

   Hive example:

   ```
   INSERT INTO TABLE pageviews PARTITION (datestamp = '2022-09-23') VALUES
   ('jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);
   ```

   Impala example:

   ```
   INSERT INTO TABLE pageviews PARTITION (datestamp = '2022-09-23') VALUES
   (CAST ('jsmith' AS VARCHAR(64)), 'mail.com', 'sports.com'), (CAST ('jdoe'
    AS VARCHAR(64)), 'mail.com', null);
   ```

   Cast strings to VARCHAR(64) when you insert data of type STRING into the Impala table.

3. **Check that the 2022-09-23 partition was created.**
   Hive and Impala example:

   ```
   SHOW PARTITIONS pageviews;
   ```

   Hive output looks something like this:

   ```
   +------------------------+
   |       partition        |
   +------------------------+
   | datestamp=2022-09-23   |
   +------------------------+
   ```

   Impala output looks something like this:

   ```
   +------------+-------+--------+------+--------------+-------------------
   +---------+-------------------+-----------------------------------------
   -----------------------------------------------------------------------+
   | datestamp  | #Rows | #Files | Size | Bytes Cached | Cache Replication |
    Format  | Incremental stats | Location
                                                                       |
   +------------+-------+--------+------+--------------+-------------------+-
   --------+-------------------+-------------------------------------------
   ---------------------------------------------------------------------+
   | 2022-09-23 | -1    | 1      | 938B | NOT CACHED   | NOT CACHED        |
    PARQUET | false             | s3a://cdpsaasdemo-cdp-private-default-1og
   vplm/warehouse/tablespace/managed/hive/pageviews/datestamp=2022-09-23 |
   | Total      | -1    | 1      | 938B | 0B           |                   |
           |                   |
                                                                      |
   +------------+-------+--------+------+--------------+------------------
   -+---------+-------------------+----------------------------------------
   ------------------------------------------------------------------------+
   ```

Creating an Impala external partitioned table

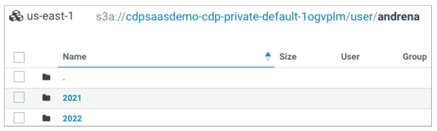
You specify a partition column using PARTITIONED BY clause when you create an Impala table. The actual data for the table can reside in a file, for example a CSV (comma-separated values) in the object store. Alternatively, the data for the external table can reside in another table that you select in a subquery.

Before you begin

- Set up Hadoop SQL policies in Ranger to include S3 URLs.

Procedure

1. Create subdirectories named 2021 and 2022 in the andrena directory on S3.



us-east-1 s3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena

Name	Size	User	Group
.			
2021			
2022			

2. Create CSV files named students_2021 and students_2022 that have the following text:
   students_2021:

```
2,john,smith,junior,engineering,2021
```

   students_2022:

```
1,jane,doe,senior,mathematics,2022
3,somporn,lee,senior,humanities,2022
```

3. Upload the CSV files students_2021.csv and students_2022.csv to the 2021 and 2022 directories you created on S3.
4. Run the queries to create the external partitioned table and alter the table to add partition directories.
   Impala example:

```
CREATE EXTERNAL TABLE IF NOT EXISTS names_partitioned(
  student_ID INT,
  FirstName STRING,
  LastName STRING,
  level STRING,
  Major STRING)
  PARTITIONED BY (grad INT)
  COMMENT 'Student Names' ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  STORED AS TEXTFILE LOCATION 's3a://cdpsaasdemo-cdp-private-default-1o
gvplm/user/andrena';
```

   Output indicates the table has been created. If you query the table, you see there is no data in the table yet.
5. Add partitions and load the data in the CSV files from the partition directories.

```
ALTER TABLE names_partitioned ADD PARTITION (grad=2021) LOCATION 's3a://
cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena/2021';
ALTER TABLE names_partitioned ADD PARTITION (grad=2022) LOCATION 's3a://
cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena/2022';
```

6. Query the partitioned table for 2022 graduates.

```
select * from names_partitioned where grad=2022;
```

Output looks something like this:

```
+------------+-----------+----------+--------+-------------+------+
| student_id | firstname | lastname | level  | major       | grad |
+------------+-----------+----------+--------+-------------+------+
| 1          | jane      | doe      | senior | mathematics | 2022 |
| 3          | somporn    | lee      | senior | humanities   | 2022 |
+------------+-----------+----------+--------+-------------+------+
```

```
 select * from names_partitioned where grad=2021;
```

7. Find the students in the table who graduated in 2021.

```
SELECT * FROM names_partitioned WHERE grad=2021;
```

Output looks something like this:

```

+------------+-----------+----------+--------+-------------+------+
| student_id | firstname | lastname | level  | major       | grad |
+------------+-----------+----------+--------+-------------+------+
| 2          | john      | smith    | junior | engineering | 2021 |
+------------+-----------+----------+--------+-------------+------+
```

Creating a Hive external partitioned table

You can configure Hive to create partitions dynamically and then run a query that creates the related directories on the object store. Hive also separates the data into the directories.

About this task

This example assumes you have the following CSV file named employees.csv to use as the data source:

```
1,jane doe,engineer,service
2,john smith,sales rep,sales
3,naoko murai,service rep,service
4,somporn thong,ceo,sales
5,xi singh,cfo,finance
```

Procedure

1. Create an andrena directory on S3.
2. Upload the CSV file to the andrena directory..
3. In the Hive shell, create an unpartitioned table that holds all the data, substituting the S3 path displayed in the Hue Web Browser.

```
CREATE EXTERNAL TABLE employees (eid int, name string, position string,
dept string)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  STORED AS TEXTFILE
  LOCATION 's3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena';
```

For example, specify location s3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena.

4. Check that the data loaded into the employees table.

```
SELECT * FROM employees;
```

The output, formatted to fit this publication, appears:

```

+------+---------------+-------------+-------+---------+
| eid  |    name       | position    | dept  |         |
+------+---------------+-------------+-------+---------|
| 1    | jane doe      | engineer    | service         |
| 2    | john smith    | sales rep   | sales           |
| 3    | naoko murai   | service rep | service         |
| 4    | somporn thong | ceo         | sales           |
| 5    | xi singh      | cfo         | finance         |
+------+---------------+-------------+-----------------+
```

5. Create a partitioned table.

```
CREATE EXTERNAL TABLE EMP_PART (eid int, name string, position string)
  PARTITIONED BY (dept string);
```

6. Set dynamic partition mode (nonstrict) to create partitioned directories of data dynamically when data is inserted:

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

7. Insert data from the unpartitioned table (all the data) into the partitioned table , dynamically creating the partitions.

```
INSERT INTO TABLE EMP_PART PARTITION (DEPT)
  SELECT eid,name,position,dept FROM employees;
```

Partitions are created dynamically.

8. Check that the partitions were created.

```
SHOW PARTITIONS emp_part;
```

```
+----------------+
|   partition    |
+----------------+
| dept=finance   |
| dept=sales     |
| dept=service   |
+----------------+
```

9. Select the data in the emp_part table;

```
SELECT * from emp_part;
```

Output looks something like this:

```
+---------------+----------------+--------------------+----------------+
| emp_part.eid  | emp_part.name  | emp_part.position  | emp_part.dept  |
+---------------+----------------+--------------------+----------------+
| 5             | xi singh       | cfo                | finance        |
| 2             | john smith     | sales rep          | sales          |
| 4             | somporn thong  | ceo                | sales          |
| 1             | jane doe       | engineer           | service        |
| 3               | naoko murai     | service rep          | service          |
+-----------------+-----------------+----------------------+------------------+
```

10. Query the data to return only rows in the sales department partition.

```
SELECT * FROM emp_part WHERE emp_part.dept="sales";
```

```
+---------------+---------------+--------------------+----------------+
| emp_part.eid  | emp_part.name  | emp_part.position  | emp_part.dept  |
+---------------+---------------+--------------------+----------------+
| 2             | john smith     | sales rep          | sales          |
| 4             | somporn thong  | ceo                | sales          |
+---------------+---------------+--------------------+----------------+
```

Repairing Hive or Impala partitions

Under certain conditions, you must manually repair metadata about Hive or Impala partitions. You learn when and how to do this.

About this task

Partition information in the metadata gets stale under the following circumstances:

- Hive: The dynamic partition refresh is disabled.
- Impala: Data files are removed by a non-Impala mechanism, and the table metadata is not updated

You must manually refresh metastore partition information when these conditions exist.

This task assumes you created a partitioned external table named students_part that stores partitions outside the warehouse. You remove one of the partition directories on the object store. This action renders the metastore inconsistent with S3. You repair the discrepancy manually to synchronize the metastore with S3 as follows;

- Hive

  To repair partition metadata, you run the MSCK (metastore consistency check) Hive command to manually add partitions that are added to or removed from the object store, but are not present in the Hive metastore.
- Impala

  To repair partition metadata, you run ALTER TABLE with the RECOVER PARTITIONS clause to to find any new partition directories and the data files.

Procedure

1. Remove a partition directory, called engineering for example, from S3.

2. In Hive or Impala, look at the students_part table partitions.

```
SHOW PARTITIONS students_part;
```

The list of partitions is stale in the metadata because it still includes the major=engineering partition, so you must run MSCK to repair the metadata.

```
+--------------------+
|       major        |
+--------------------+
| major=engineering  |
| major=humanities   |
| major=mathematics  |
+--------------------+
```

3. Repair the partition.
   Hive example:

   ```
   MSCK REPAIR TABLE students_part DROP PARTITIONS;
   ```

   Impala example:

   ```
   ALTER TABLE students_part RECOVER PARTITIONS;
   ```

4. Show that the repair succeeded.

   ```
   SHOW PARTITIONS students_part;
   ```

   The repair was successful. The sales partition no longer appears.

   ```
   +--------------------+
   |       major        |
   +--------------------+
   | major=humanities   |
   | major=mathematics  |
   +--------------------+
   ```

Using your schema in PostgreSQL

You follow an example of how to create an external table in PostgreSQL using your own schema.

Procedure

1. Using Postgres, create external tables based on a user-defined schema.

   ```
   CREATE SCHEMA bob;
   CREATE TABLE bob.country
   (
       id   int,
       name varchar(20)
   );
   insert into bob.country
   values (1, 'India');
   insert into bob.country
   values (2, 'Russia');
   insert into bob.country
   values (3, 'USA');

   CREATE SCHEMA alice;
   CREATE TABLE alice.country
   (
       id   int,
       name varchar(20)
   );
   insert into alice.country
   values (4, 'Italy');
   insert into alice.country
   values (5, 'Greece');
   insert into alice.country
   values (6, 'China');
   insert into alice.country
   values (7, 'Japan');
   ```

2. Create a user and associate them with a default schema <=> search_path.

```
CREATE ROLE greg WITH LOGIN PASSWORD 'GregPass123!$';
ALTER ROLE greg SET search_path TO bob;
```

3. Grant the necessary permissions to be able to access the schema.

```
GRANT USAGE ON SCHEMA bob TO greg;
GRANT SELECT ON ALL TABLES IN SCHEMA bob TO greg;
```

Determining the table type

You can determine the type of a SQL table, whether it has ACID properties, its storage format, such as ORC or Parquet, and other information. Knowing the table type is important for understanding how to store data in the table or how to completely remove data from the cluster.

About this task

Procedure

1. Get an extended description of the table.
   Hive example:

```
DESCRIBE EXTENDED `mydatabase`.`mytable`;
```

   Impala example:

```
DESCRIBE EXTENDED mydatabase.mytable;
```

2. Scroll through the command output to see the table type.

   The output says the table type is managed. The transaction=true indicates that the table has ACID properties:

```
+-------------------------------+-------------------------------------
-----------------
| name                          | type
      | comment
+-------------------------------+--------------------------------------
----------------
...
| Location:                     | s3a://cdpsaasdemo-cdp-private-default-
1ogvplm/ | NULL
                                          warehouse/tablespace/managed/
hive/t3 | NULL
| Table Type:                   | MANAGED_TABLE
      | NULL
| Table Parameters:             | NULL
      | NULL
|                               | OBJCAPABILITIES
       | NULL
|                               | transactional
      | true
|                               | transactional_properties
       | insert_only
...
```

Using database.table in queries

In Hive, table references using dot notation require backticks around the database and table names.

About this task

Hive rejects `db.table` in SQL queries. The dot (.) is not allowed in table names. To reference the database and table in a Hive table name, enclosed both in backticks as follows: `db`.`table` .

Impala table references do not require backticks.

Procedure

Create a table using dot notation to refer to the database and table.
Hive example:

```
CREATE TABLE `math`.`students` (name VARCHAR(64), age INT, gpa DECIMAL(3
,2));
```

Impala example:

```
CREATE TABLE math.students (name VARCHAR(64), age INT, gpa DECIMAL(3,2));
```

Inserting data into a table

To insert data into a table you use a familiar ANSI SQL statement. A simple example shows you have to accomplish this basic task.

About this task

To insert data into a full ACID table, use Hive and store data in Optimized Row Columnar (ORC). To insert data into a non-ACID table or insert-only ACID table, you can use Hive or Impala and store data in other supported formats. You can specify partitioning as shown in the following syntax:

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

where

values_row is (value [, value]).

A value can be NULL or any SQL literal.

Procedure

1. Create an ACID table to contain student information.
   CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3,2));
2. Insert name, age, and gpa values for a few students into the table.
   INSERT INTO TABLE students VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32, 2.32);
3. Create a table called pageviews and assign null values to columns you do not want to assign a value.
   Hive and Impala example

   ```
   CREATE TABLE pageviews (userid VARCHAR(64), link STRING, from STRING) PA
   RTITIONED BY (datestamp STRING);
   ```

```
INSERT INTO TABLE pageviews PARTITION (datestamp = '2014-09-23') VALUES ('
jsmith', 'mail.com', 'sports.com'), ('jdoe', 'mail.com', null);
INSERT INTO TABLE pageviews PARTITION (datestamp) VALUES ('tjohnson', '
sports.com', 'finance.com', '2014-09-23'), ('tlee', 'finance.com', null,
 '2014-09-21');
```

The ACID data resides in the warehouse. From Hive, a full ACID table results. From Impala, an insert-only ACID table results.

Updating data in a table

You use the UPDATE statement to modify data already stored in a table. An example shows how to apply the syntax.

About this task

You construct an UPDATE statement using the following syntax:

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

Depending on the condition specified in the optional WHERE clause, an UPDATE statement might affect every row in a table. The expression in the WHERE clause must be an expression supported by a SELECT clause. Subqueries are not allowed on the right side of the SET clause. Partition columns cannot be updated.

Impala allows an optional FROM clause after the SET clause to restrict the updates to only the rows in the specified table that are part of the result set for a join query.

Before you begin

You must have SELECT and UPDATE privileges to use the UPDATE statement.

Procedure

Create a statement that changes the values in the name column of all rows where the gpa column has the value of 1.0.

```
UPDATE students SET name = null WHERE gpa <= 1.0;
```

Deleting data from a table

You use the DELETE statement to delete data already written to an ACID table from Hive. Impala does not support deletions of ACID data.

About this task

Use the following syntax to delete data from a Hive table.

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

Procedure

Delete any rows of data from the students table if the gpa column has a value of 1 or 0.

Hive example:

```
DELETE FROM students WHERE gpa <= 1,0;
```

Using a subquery

Hive and Impala supports subqueries in FROM clauses and WHERE clauses that you can use for many operations, such as filtering data from one table based on contents of another table.

About this task

A subquery is a SQL expression in an inner query that returns a result set to the outer query. From the result set, the outer query is evaluated. The outer query is the main query that contains the inner subquery. A subquery in a WHERE clause includes a query predicate and predicate operator. A predicate is a condition that evaluates to a Boolean value. The predicate in a subquery must also contain a predicate operator. The predicate operator specifies the relationship tested in a predicate query.

Before you begin

You ran the following queries in Hive or Impala to create the tables for this task.

```
CREATE TABLE us_census(year int);
INSERT INTO us_census VALUES(2020),(2021),(2022);
CREATE TABLE transfer_payments(state STRING, net_payments FLOAT, year INT);
INSERT INTO transfer_payments VALUES('CA',12000.75,2010),('CA',45789.95,2020
),('WA',78987.52,2022);
```

Procedure

Select all the state and net_payments values from the transfer_payments table if the value of the year column in the table matches a year in the us_census table.
Hive and Impala example:

```
SELECT state, net_payments
FROM transfer_payments
WHERE transfer_payments.year IN (SELECT year FROM us_census);
```

The predicate starts with the first WHERE keyword. The predicate operator is the IN keyword.

The predicate returns true for a row in the transfer_payments table if the year value in at least one row of the us_census table matches a year value in the transfer_payments table.

```
+--------+---------------+
| state  | net_payments  |
+--------+---------------+
| CA     | 45789.95      |
| WA     | 78987.52      |
+--------+---------------+
```

Aggregating and grouping data

You use functions, such as AVG, SUM, or MAX, to aggregate data. The GROUP BY clause groups data query results in one or more columns.

About this task

The GROUP BY clause explicitly groups data. Implicit grouping occurs when aggregating the table.

Before you begin

- You upload a CSV file to S3 having the following data:

```
1,jane doe,engineer,service,2022,375000
2,john smith,sales rep,sales,2021,375000
3,naoko murai,service rep,service,2022,200000
4,somporn thong,ceo,sales,2020,500000
5,xi singh,cfo,finance,2023,222000
6,mary brown,sales rep,sales,2021,475000
7,rashida kumar,sales rep,sales,2021,299000
```

- You create a table having the following data in Hive or Impala as shown in the examples below:

```
+-----+---------------+-------------+---------+------+----------+
| eid | name          | job         | dept    | year | salary   |
+-----+---------------+-------------+---------+------+----------+
| 1   | jane doe      | engineer    | service | 2022 | 375000.0 |
| 2   | john smith    | sales rep   | sales   | 2021 | 375000.0 |
| 3   | naoko murai   | service rep | service | 2022 | 200000.0 |
| 4   | somporn thong | ceo         | sales   | 2020 | 500000.0 |
| 5   | xi singh      | cfo         | finance | 2023 | 222000.0 |
| 6   | mary brown    | sales rep   | sales   | 2021 | 475000.0 |
| 7   | rashida kumar | sales rep   | sales   | 2021 | 299000.0 |
+-----+---------------+-------------+---------+------+----------+
```

Hive and Impala example:

```
CREATE EXTERNAL TABLE employees (eid int, name string, job string, dept
string, year int, salary float)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 's3a://cdpsaasdemo-cdp-private-default-1ogvplm/user/andrena';
```

Procedure

1. Run a query that returns the average salary of all employees in the engineering department grouped by year.
Hive and Impala example:

```
SELECT year, AVG(salary) as average_pay
FROM employees
WHERE dept = 'sales' GROUP BY year;
```

Hive output:

```
+-------+-------------+
| year  | average_pay  |
+-------+-------------+
| 2020  | 500000.0     |
| 2021  | 383000.0     |
+-------+-------------+
```

Impala output:

```
+------+-------------+
| year | average_pay |
+------+-------------+
| 2021 | 383000.0    |
| 2020 | 500000.0    |
+------+-------------+
```

2. **2.** Run an implicit grouping query to get the highest paid employee and the average pay.
   Hive and Impala example:

   ```
   ELECT MAX(salary) as highest_pay,
   AVG(salary) as average_pay
   FROM employees
   WHERE dept = 'sales';
   ```

   Hive and Impala output:

   ```
   +--------------+--------------+
   | highest_pay  | average_pay  |
   +--------------+--------------+
   | 500000.0     | 412250.0     |
   +--------------+--------------+
   ```

3. **3.** Delete the employees.csv from S3.