Cloudera Data Warehouse Private Cloud 1.5.0

Querying Data in Unified Analytics

Date published: 2023-01-24
Date modified: 2023-01-25

CLOUDERA

Legal Notice

Contents

Unified Analytics basics

A brief description of Unified Analytics engines and query editors prepares you to run a query. You learn how to select the SQL engine that best suits your needs.

In Cloudera Data Warehouse (CDW), Unified Analytics integrates essential warehouse software:

- SQL engines: Hive and Impala
- Query editors: Hue and the Impala shell

As DW Admin, you use a one- or two-step procedure to create a Virtual Warehouse (VW) for querying your data, as shown in the following diagram:
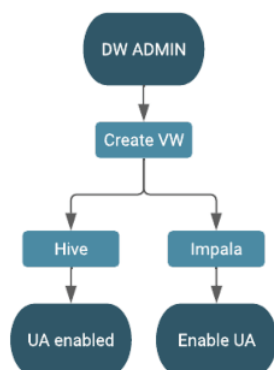
When you create a Hive Virtual Warehouse, Unified Analytics is enabled. When you create an Impala Virtual Warehouse, you need to perform an extra step to enable Unified Analytics.

Related Information

CDP CLI: Create a Database Catalog create-db

Setting up an Impala Virtual Warehouse

You learn how to set up a Unified Analytics Virtual Warehouse for Business Intelligence (BI), or ad-hoc querying. An Impala Virtual Warehouse can read full ACID tables in the Hive metastore, and can write insert-only ACID tables because you select the Impala engine type.

About this task

First you create a Virtual Warehouse of SQL engine type Impala, and then you enable Unified Analytics.

Before you begin

- Required role: DW Admin

Procedure

1. Create a Virtual Warehouse as described in Adding a new Virtual Warehouse, selecting the Impala engine type.

New Virtual Warehouse

Name *

impala

Type *

HIVE   IMPALA

2. Select a database catalog, or accept the default.

3. Select a size for the Virtual Warehouse.

4. Select Enable Unified Analytics.

5. Deselect Disable AutoSuspend if you intend to use Impala coordinator shutdown, which you enable in the next step.

   The Impala coordinator does not automatically shutdown unless the Impala executors are suspended.

6. Configure Impala coordinator shutdown by turning on Allow Shutdown Of Coordinator and accept the default value Enabled Active-Passive in High Availability (HA).

7. Customize other options as necessary.

8. Click Create.

Related Information

Adding a new Virtual Warehouse

CDP CLI: Create a Virtual Warehouse create-vw

Setting up a Hive Virtual Warehouse

You learn how to set up a Hive Unified Analytics Virtual Warehouse. If you have extract, transform, and load (ETL) jobs, you need to learn how to enable query isolation mode.

About this task

When you create a Hive Virtual Warehouse, Unified Analytics is automatically enabled. This type of Virtual Warehouse can read and write full ACID tables in the Hive metastore. ETL jobs, identified by their size, run in query isolation mode if enabled.

Before you begin

• Required role: DW Admin

Procedure

1. Create a Virtual Warehouse, selecting Hive as the type of SQL engine.

2. Select a database catalog, or accept the default.

3. Select a size for the Virtual Warehouse.

4. Click Query Isolation.

   Note: Query isolation feature for Unified Analytics is in Technical Preview in 1.5.0 and not recommended for use in production deployments. Cloudera recommends that you try this feature in test or development environments.

5. Click Create.

Running a query in Cloudera Data Warehouse

You simply open Hue and submit your query. You do not need to manually start beeline or any other shell.

About this task

As a DW User, you open Hue from a Virtual Warehouse that you set up, and run the query. The SQL engine reads from and writes to the same metastore, regardless of the type Virtual Warehouse.
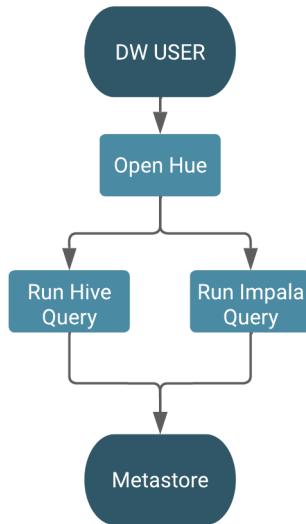
Before you begin

• Required role: DW User
• You obtained permission to run SQL queries from the Env Admin, who added you to a Hadoop SQL policy.

Procedure

1.
     On the **Overview** page under Virtual Warehouses, click options ⋮ , and select Open Hue.

2. Select a database.

For example, select database airline_ontime_parquet.
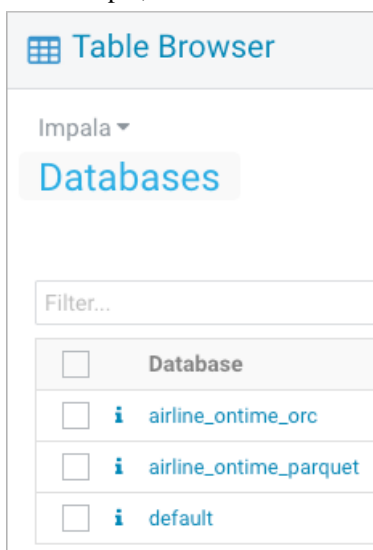
3. In Hue, enter a query.

```
SELECT dest, origin
FROM flights
GROUP BY dest, origin;
```

4. Click ▶ to run the query.

Related Information

Submitting queries with Hue

Granting permissions to run SQL queries

You must grant Hadoop SQL Ranger permissions to your users so they can query tables.

Procedure

1. Log in to the CDP Management Console as an Administrator.

2. Go to Environments and click on your environment name.

3. Click Ranger on the Data Lake tab.

This opens the Ranger service in Cloudera Manager.

4. Click Ranger Admin Web UI to open Ranger.

5. Click Hadoop SQL on the **Service Manager** page.

6. Click Add New Policy.

7. Specify the databases, tables, and columns on which you want to grant permissions to the users.

8. Under the **Allow Conditions** section, specify users or groups to whom you want to grant permissions.

you can specify {USER} to grant permissions to all users.

9. Click Add Permissions and select permissions such as Create, Alter, Drop, Select, and so on.

10. Click Add to save the settings.

Using DDL statements

Unified Analytics supports DDL statements to create a database schema and to define the type and structure of the data added to a database.

The following list contains the DDL statements that are currently supported through the Unified Analytics engine.

- Create/Drop/Alter/Use Database
- Create/Drop/Truncate Table
- Create Table Like Parquet
- Alter Table/Partition/Column
- Create/Drop/Alter View
- Create/Drop/Alter Materialized View
- Create/Drop/Alter Scheduled Query
- Create/Drop/Grant/Revoke Roles and Privileges
- Show
- Describe
- Analyze Compute / Drop Statistics
- Refresh Table

> **Note:** For more information about these statements, refer to Hive documentation. Be sure to use Hive syntax when you run these statements on either Unified Analytics SQL engines.

Escaping an invalid identifier

When you need to use reserved words, special characters, or a space in a column or partition name, enclose it in backticks (`).

About this task

An identifier in SQL is a sequence of alphanumeric and underscore (_) characters enclosed in backtick characters. In Hive, these identifiers are called quoted identifiers and are case-insensitive. You can use the identifier instead of a column or table partition name.

Before you begin

You have set the following parameter to column in the hive-site.xml file to enable quoted identifiers:

Set the hive.support.quoted.identifiers configuration parameter to column in the hive-site.xml file to enable quoted identifiers in column names. Valid values are none and column. For example, in Hive execute the following command: SET   hive.support.quoted.identifiers = column.

Procedure

1. Create a table named test that has two columns of strings specified by quoted identifiers:
   CREATE TABLE test (`x+y` String, `a?b` String);

2. Create a table that defines a partition using a quoted identifier and a region number:
   CREATE TABLE partition_date-1 (key string, value string) PARTITIONED BY (`dt+x` date, region int);

3. Create a table that defines clustering using a quoted identifier:
   CREATE TABLE bucket_test(`key?1` string, value string) CLUSTERED BY (`key?1`) into 5 buckets;

Client access to Unified Analytics

You learn which interfaces Unified Analytics supports for connecting a client. Procedures for using the JDBC and Impala shell interfaces include step-by-step instructions.

To connect to Unified Analytics you can download the JDBC driver from the Unified Analytics to give to your client. Also, provide the Unified Analytics endpoint to your client to establish the connection.

You can connect and submit requests using the following interfaces:

- JDBC/ODBC
- Beeline
- Impala shell
- Impyla Python client
- Hue web-based user interface

Connecting a JDBC client

You see how to establish the connection between your JDBC client and Unified Analytics.

About this task

In this task, you connect to a Unified Analytics Virtual Warehouse. First, you download the JDBC driver to give to your client. Next, you obtain the Unified Analytics endpoint for the connection. Finally, you provide necessary information to your client.

Procedure

1. Log in to the CDP web interface and navigate to the Data Warehouse service.

2. Click Data Warehouse Overview .

3. Click ⋮ on the Virtual Warehouse tile.

4. Click Download JDBC Jar or Download Unified Analytics JDBC Driver, depending on your Virtual Warehouse SQL engine type.
   This action downloads the JAR (driver) file to your local system.

5. Provide the JAR file you downloaded to your JDBC client. For most of the clients, the JAR file should be added under the Libraries folder. Refer to your client documentation for information about the correct location.

6. Click the options ⋮ again.

7. Click Copy JDBC URL.

8. Paste the copied JDBC URL into a text file. It looks something like this:

```
jdbc:hive2://<your-virtual-warehouse>.<your-environment>.<your-domain.co
m>/default;transportMode=http;httpPath=cliservice;ssl=true;retries=3
```

9. Provide the Unified Analytics JDBC URL to your client.

```
<your-virtual-warehouse>.<your-environment>.<domain.company.com>
```

10. On the JDBC client end, check that the user can set the authentication credentials:

   Authentication: Username and Password

   Username: Username you use to connect to the CDP Data Warehouse service.

   Password: Password you use to connect to the CDP Data Warehouse service.

Connecting an Impala shell client

You need to provide the commands to your client users for installing and launching the Impala shell to connect to your Unified Analytics. Client users can then query your tables. You learn how to obtain the command for installing the Impala shell on a client and other information to provide to clients.

About this task

To access Unified Analytics from the Impala Shell, clients need to connect to HiveServer (HS2), which is available in the same CDP cluster as Impala. The JDBC URL for the HS2 endpoint uses the strict HS2 protocol option to access Unified Analytics from the Impala Shell. Windows clients are not supported.

Before you begin

• Obtain the DWUser role.

Procedure

1. On the Unified Analytics Impala Virtual Warehouse tile, click ⋮ , and select Copy Impala shell Download command.

2. Click ⋮ again, and select Copy Impala shell command.

   This action copies the command that launches the Impala shell and includes the connection string your client needs to connect to Unified Analytics in your Virtual Warehouse. For example:

```
impala-shell --protocol='hs2-http' --ssl -i 'coordinator-vw-impala.dw-dwx-
rzs556.xcu2-8y8x.dev.cldr.work:443' -u client_max -l
```

3. Provide the commands you copied to your client user.

4. Provide the instructions to your client user to use as described in the next topic.

Using the Impala shell client

You obtain the commands from the Impala Virtual Warehouse owner for installing and launching the Impala shell to connect to the Impala Virtual Warehouse. You learn how to install and launch the Impala shell that connects to the Impala Virtual Warehouse UI. You then query tables in the Impala Virtual Warehouse, assuming you have authentication credentials. Windows clients are not supported.

Before you begin

- You must have the latest stable version of Python 2.7.
- You must obtain the pip installer compatible with the Python version.

Procedure

1. In a terminal window, run the command provided by the Impala Virtual Warehouse owner to update impyla to the version compatible with the Impala Virtual Warehouse.
   For example:

   ```
   pip install impyla==0.18a2
   ```

   Installing/updating to impyla 0.18a2, for example, is required before you install the Impala shell in the next step.

2. Run the Impala shell Download command provided by the Impala Virtual Warehouse owner to install the Impala shell.

   The command looks something like this:

   ```
   pip install impala-shell==4.1.0a1
   ```

   The Impala shell is installed.

3. Run the Help command to confirm a successful installation:

   ```
   impala-shell --help
   ```

4. Run the provided Impala shell command to launch the Impala shell.

   The command looks something like this:

   ```
   impala-shell --protocol='hs2-http' --ssl -i 'coordinator-vw-impala.dw-dwx-
   rzs556.xcu2-8y8x.dev.cldr.work:443' -u client_max -l
   ```

   The Impala shell is launched and you connect to the Impala Virtual Warehouse.

5. Run a SQL command to confirm that you are connected to the Impala Virtual Warehouse instance.

   ```
   SHOW DATABASES;
   ```

   The query returns a list of databases in the Impala Virtual Warehouse.

Materialized views

A materialized view is a Hive-managed database object that can be queried by Hive or Impala users. The materialized view holds a query result you can use to speed up the execution of a query workload. If your queries are repetitive, you can reduce latency and resource consumption by using materialized views. You create materialized views to optimize your queries automatically.

Using a materialized view, the optimizer can compare old and new tables, rewrite queries to accelerate processing, and manage maintenance of the materialized view when data updates occur. The optimizer can use a materialized view to fully or partially rewrite projections, filters, joins, and aggregations.

When you enable BI mode to use Apache DataSketches approximations, a materialized view automatically calls DataSketches for some types of operations.

You can perform the following materialized view operations:

- Create a materialized view of queries or subqueries
- Drop a materialized view
- Show materialized views
- Describe a materialized view
- Enable or disable query rewriting based on a materialized view
- Globally enable or disable rewriting based on any materialized view
- Use partitioning to improve the performance of materialized views

Creating and using a materialized view

You can create a materialized view of a query to calculate and store results of an expensive operation, such as a particular join, on a managed, ACID table that you repeatedly run. When you issue queries specified by that materialized view, the optimizer rewrites the query based on it. This action saves reprocessing. Query performance improves.

About this task

In the tasks that follow, first you create, or use an existing, Hive Virtual Warehouse. You create and populate example tables. The tables are managed tables. You cannot create a materialized view of an external table. You create a materialized view of a join of the tables. Subsequently, you run a query to join the tables, and the query plan takes advantage of the precomputed join to accelerate processing. These over-simplified tasks show the syntax and output of a materialized view, and do not demonstrate accelerated processing that occurs in a real-world task, processing a large amount of data.

Before you begin

- You have access to an existing Hive Virtual Warehouse, or you created a new one.
- You have a client connection, such as Beeline, to Hive in CDP.

Creating the tables and view

You see how to create simple tables, insert the data, and join the tables using a materialized view. You run the query, and the optimizer takes advantage of the precomputation performed by the materialized view to speed response time.

Procedure

1. Click ⋮ on the Virtual Warehouse tile and select Open Hue.

2. Select a database, for example default.

3. Create a managed table.

For example, create this emps table in the Hue editor:

```
CREATE TABLE emps (
empid INT,
deptno INT,
name VARCHAR(256),
salary FLOAT,
hire_date TIMESTAMP);
```

4. Create another table.

For example, create this depts table:
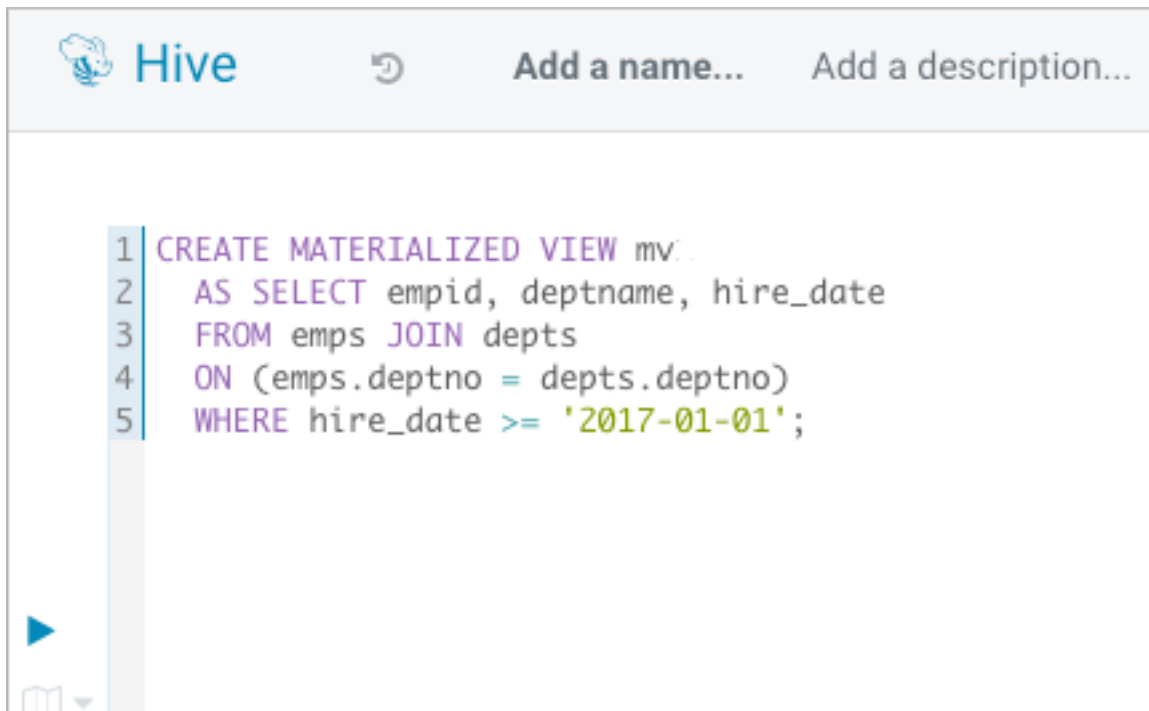
```
CREATE TABLE depts (
deptno INT,
deptname VARCHAR(256),
locationid INT);
```

5. Insert data into the tables by copy/pasting the following statements one-by-one and executing them one-by-one.

```
INSERT INTO TABLE emps VALUES (10001,101,CAST('jane doe' as VARCHAR(256)
),250000,'2018-01-10');
INSERT INTO TABLE emps VALUES (10002,100,CAST('somporn klailee' as VARCH
AR(256)),210000,'2017-12-25');
INSERT INTO TABLE emps VALUES (10003,200,CAST('jeiranan thongnopneua' as
VARCHAR(256)),175000,'2018-05-05');
INSERT INTO TABLE depts VALUES (100,CAST('HR' as VARCHAR(256)),10);
INSERT INTO TABLE depts VALUES (101,CAST('Eng' as VARCHAR(256)),11);
INSERT INTO TABLE depts VALUES (200,CAST('Sup' as VARCHAR(256)),20);
```

6. Create a materialized view to join the tables:

```
CREATE MATERIALIZED VIEW mv
AS SELECT empid, deptname, hire_date
FROM emps JOIN depts
ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2017-01-01';
```


Verifying the query rewrite

You can check that a materialized view is used as a query rewrite.

Procedure

1. In Hue, verify that the query rewrite used the materialized view by running an extended EXPLAIN statement:

```
EXPLAIN EXTENDED SELECT empid, deptname
FROM emps
JOIN depts
ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2017-01-01'
AND hire_date <= '2019-01-01';
```

2. Export results.
   The output shows the alias default.mv1 for the materialized view in the Scan section of the plan.

```
Explain
OPTIMIZED SQL: SELECT `empid`, `deptname`
FROM `default`.`mv`
WHERE '2019-01-01' >= `hire_date`
STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-0 is a root stage
STAGE PLANS:
Stage: Stage-1
Impala
Impala Plan:
F01:PLAN FRAGMENT [UNPARTITIONED] hosts=1 instances=1
|  Per-Instance Resources: mem-estimate=4.02MB mem-reservation=4.00MB threa
d-reservation=1
PLAN-ROOT SINK
|  output exprs: default.mv.empid, default.mv.deptname
|  mem-estimate=4.00MB mem-reservation=4.00MB spill-buffer=2.00MB thread-
reservation=0
|
01:EXCHANGE [UNPARTITIONED]
|  mem-estimate=16.00KB mem-reservation=0B thread-reservation=0
|  tuple-ids=0 row-size=32B cardinality=15
|  in pipelines: 00(GETNEXT)
|
F00:PLAN FRAGMENT [RANDOM] hosts=1 instances=1
Per-Instance Resources: mem-estimate=16.00MB mem-reservation=24.00KB thre
ad-reservation=1
00:SCAN S3 [default.mv default.mv1, RANDOM]
S3 partitions=1/1 files=1 size=964B
predicates: casttotimestamp('2019-01-01') >= default.mv1.hire_date
stored statistics:
table: rows=unavailable size=964B
columns: unavailable
extrapolated-rows=disabled max-scan-range-rows=unavailable
parquet dictionary predicates: casttotimestamp('2019-01-01') >= default
.mv1.hire_date
mem-estimate=16.00MB mem-reservation=24.00KB thread-reservation=0
tuple-ids=0 row-size=32B cardinality=15
in pipelines: 00(GETNEXT)
Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink
```

Query rewrite example

You examine a materialized view definition and see how an example query that matches a materialized view is rewritten transparently.

The following materialized view definition uses dynamic partitioning by year:

```
CREATE MATERIALIZED VIEW mat_view
PARTITIONED ON (d_year)
AS SELECT d_year, d_moy, d_dom, SUM(ss_sales_price) AS sum_sales
FROM store_sales, date_dim
WHERE ss_sold_date_sk = d_date_sk AND d_year > 2017
GROUP BY d_year, d_moy, d_dom;
```

With this materialized view in place, when the following query runs, transparent rewriting to speed up the execution occurs.

```
SELECT SUM(ss_sales_price) AS sum_sales
FROM store_sales, date_dim
WHERE ss_sold_date_sk = d_date_sk AND d_year = 2018 AND d_mon IN (1,2,3);
```

The query matches most of the constructs, but not the filter, in the materialized view. Calcite recognizes this query as a full, mapped query, minus the predicate. Calcite rewrites the query as a select from the materialized view. Calcite also recognizes that the materialized view is partitioned, and does partition elimination:

```
SELECT SUM(sum_sales) AS sum_sales
FROM mat_view
WHERE d_year = 2018 AND d_mon IN (1,2,3);
```

The final rewrite selects from the partition, and applies the predicate to the eliminated partition.

```
SELECT SUM(sum_sales) AS sum_sales
 FROM mat_view(d_year = 2018)
 WHERE d_mon IN (1,2,3);
```

Managing query rewrites

After changes to base tables, the data in a materialized view is stale. You need to know how to prevent the SQL optimizer from rewriting queries in this situation. If you want a query executed without regard to a materialized view, for example to measure the execution time difference, you can disable rewriting and then enable it again.

About this task

As administrator, you can globally enable or disable all query rewrites based on materialized views. By default, the optimizer rewrites a query based on a materialized view.

Procedure

1. Disable rewriting of a query based on a materialized view named mv1 in the default database.

```
ALTER MATERIALIZED VIEW default.mv1 DISABLE REWRITE;
```

2. Enable rewriting of a query based on materialized view mv1.

```
ALTER MATERIALIZED VIEW default.mv1 ENABLE REWRITE;
```

3. Globally enable rewriting of queries based on materialized views by setting a global property.

```
SET hive.materializedview.rewriting=true;
```

Using optimizations from a subquery

You can create a query having a subquery that the optimizer rewrites based on a materialized view. You create a materialized view, and then run a query that uses the materialized view.

About this task

In this task, you create a materialized view and use it in a subquery to return the number of destination-origin pairs. Suppose the data resides in a table named flights_data that has the following columns:

c_id	dest	origin
1	Chicago	Hyderabad
2	London	Moscow
...		

Procedure

1. Create a table schema definition named flights_data for destination and origin data.

```
CREATE TABLE flights_data(
   c_id INT,
   dest VARCHAR(256),
   origin VARCHAR(256));
```

2. Create a materialized view that counts destinations and origins.

```
CREATE MATERIALIZED VIEW mv1
AS
   SELECT dest, origin, count(*)
   FROM flights_data
   GROUP BY dest, origin;
```

3. Take advantage of the materialized view to speed your queries when you have to count destinations and origins again.

For example, use a subquery to select the number of destination-origin pairs like the materialized view.

```
SELECT count(*)/2
FROM(
   SELECT dest, origin, count(*)
   FROM flights_data
   GROUP BY dest, origin
) AS t;
```

Transparently, the SQL engine uses the work already in place since creation of the materialized view instead of reprocessing.

Dropping a materialized view

You must understand when to drop a materialized view to successfully drop related tables.

About this task

Drop a materialized view before performing a DROP TABLE operation on a related table. You cannot drop a table that has a relationship with a materialized view.

In this task, you drop a materialized view named mv1 from the database named default.

Procedure

Drop a materialized view in my_database named mv1 .
DROP MATERIALIZED VIEW default.mv;

Showing materialized views

You can list all materialized views in the current database or in another database. You can filter a list of materialized views in a specified database using regular expression wildcards.

About this task

You can use regular expression wildcards to filter the list of materialized views you want to see. The following wildcards are supported:

- Asterisk (*)

  Represents one or more characters.
- Pipe symbol (|)

  Represents a choice.

For example, mv_q* and *mv|q1* match the materialized view mv_q1. Finding no match does not cause an error.

Procedure

1. List materialized views in the current database.
   SHOW MATERIALIZED VIEWS;
2. List materialized views in a particular database.
   SHOW MATERIALIZED VIEWS IN another_database;

Describing a materialized view

You can get summary, detailed, and formatted information about a materialized view.

About this task

This task builds on the task that creates a materialized view named mv1.

Procedure

1. Get summary information about the materialized view named mv1.

   ```
   DESCRIBE mv1;
   ```


2. Get detailed information about the materialized view named mv1.

   ```
   DESCRIBE EXTENDED mv1;
   ```

3. Click Export Results to copy the information.

4. Get formatting details about the materialized view named mv1.

```
DESCRIBE FORMATTED mv1;
```

For example, 14 of the 38 actual results are shown below:

		col_name	data_type
		Query History **Saved Queries**	**Results (38)**
	1	dest	varchar(256)
	2	origin	varchar(256)
	3	_c2	bigint
	4		NULL
	5	# Detailed Table Information	NULL
	6	Database:	default
	7	OwnerType:	USER
	8	Owner:	csso_khahn
	9	CreateTime:	Sat May 22 23:57:47 UTC 2021
	10	LastAccessTime:	UNKNOWN
	11	Retention:	0
	12	Location:	s3a://qe-s3-bucket-weekly-xtx4--
	13	Table Type:	MATERIALIZED_VIEW
	14	Table Parameters:	NULL

Periodically rebuilding a materialized view

Using materialized views can enhance query performance. You need to update materialized view contents when new data is added to the underlying table. Instead of rebuilding the materialized view manually, you can schedule this task. Automatic rebuilding then occurs periodically and transparently to users.

About this task

This task assumes you created the following schemas for storing employee and departmental information:

```
CREATE TABLE emps (
empid INTEGER,
deptno INTEGER,
name VARCHAR(256),
salary FLOAT,
hire_date TIMESTAMP);

CREATE TABLE depts (
deptno INTEGER,
deptname VARCHAR(256),
locationid INTEGER);
```

Imagine that you add data for a number of employees to the table. Assume many users of your database issue queries to access to data about the employees hired during last year including the department they belong to.

You perform the steps below to create a materialized view of the table to address these queries. Imagine new employees are hired and you add their records to the table. These changes render the materialized view contents outdated. You need to update its contents. You create a scheduled query to perform this task. The scheduled rebuilding will not occur unless there are changes to the input tables. You test the scheduled query by bypassing the schedule and executing the schedule immediately. Finally, you change the schedule to rebuild less often.

Procedure

1. To handle many queries to access recently hired employee and departmental data, create a materialized view.

```
CREATE MATERIALIZED VIEW mv_recently_hired AS
   SELECT empid, name, deptname, hire_date FROM emps
   JOIN depts ON (emps.deptno = depts.deptno)
   WHERE hire_date >= '2020-01-01 00:00:00';
```

2. Use the materialized view by querying the employee data.

```
SELECT empid, name FROM emps
JOIN depts  ON (emps.deptno = depts.deptno)
WHERE hire_date >= '2020-03-01 00:00:00' AND deptname = 'finance';
```

3. Assuming new hiring occurred and you added new records to the emps table, rebuild the materialized view.

```
ALTER MATERIALIZED VIEW mv_recently_hired REBUILD;
```

The rebuilding updates the contents of the materialized view.

4. Create a scheduled query to invoke the rebuild statement every 10 minutes.

```
CREATE SCHEDULED QUERY scheduled_rebuild
EVERY 10 MINUTES AS
ALTER MATERIALIZED VIEW mv_recently_hired REBUILD;
```

A rebuild executes every 10 minutes, assuming changes to the emp table occur within that period. If a materialized view can be rebuilt incrementally, the scheduled rebuild does not occur unless there are changes to the input tables.

5. To test the schedule, run a scheduled query immediately.

```
ALTER SCHEDULED QUERY scheduled_rebuild EXECUTE;
```

6. Change the frequency of the rebuilding.

```
ALTER SCHEDULED QUERY scheduled_rebuild EVERY 20 MINUTES;
```

Incremental rebuilds

The materialized view is a physical manifestation and summary of the original data in your table. If the source table, or multiple tables in the case of joins, changes, you likely want updates to the materialized view.

An update can occur as an incremental rebuild, or a full rebuild, of the materialized view. A full rebuild can be expensive. An incremental rebuild updates only the affected parts of the materialized view. An incremental rebuild improves the efficiency of maintaining the materialized view by decreasing rebuild step execution time. The incremental rebuild also relies on existing ACID guarantees of view tables.

The following types of incremental rebuilds occur automatically:

- Record-based

  The delta is generated based on a snapshot of source tables taken at view creation and at rebuild time. Source tables must be full ACID (transactional) tables, which only Hive can write.

- Partition-based

  The partitions that need rebuilding are identified, and then only those partitions are updated by an insert-overwrite. Source tables can be insert-only ACID (tranactional) tables, which Hive and Impala can write.

The materialized view must be transactional to qualify for an incremental rebuild. If the materialized view is not transactional, a full rebuild occurs.

Conditions for a record-based incremental rebuild

A record-based incremental rebuild occurs under the following conditions:

- The materialized view includes a count, sum, min, or max aggregation, and one of the source tables has an additional insert after the view is created.
- The materialized view includes a sum aggregation, and one of the source tables has an additional insert or delete after the view is created.
- You are using a Unified Analytics Virtual Warehouse with a Hive engine, which is the only Virtual Warehouse type that supports the incremental, records-based rebuild.
- The materialized view is a full ACID table.

Conditions for a partition-based incremental rebuild

If compaction did not run on the source table after the last rebuild, a partitions-based incremental rebuild occurs under the following conditions:

- The source table is an insert-only transactional tables.
- The materialized view is an insert-only transactional table.

  If the materialized view is a full acid, a partition-based incremental rebuild is not attempted. Instead, a record-based incremental rebuild is attempted.
- The materialized view definition includes any supported aggregation.
- You are using a Unified Analytics Virtual Warehouse with the Hive engine.

Purposely using a stale materialized view

When materialized view data becomes stale, or when the optimizer cannot determine the data freshness, which is the case if you use external tables, you might want to use the materialized view anyway:

About this task

You can use the materialized view by performing these tasks:

- Schedule the materialized view for rebuilding. For example, schedule a rebuild to occur every x minutes.
- Adjust the rewriting time window to use stale or possibly stale data for a period of time. For example, schedule the window within which to use stale data for x + y minutes.

Procedure

1. Create a scheduled query to invoke the rebuild statement every 10 minutes.

```
CREATE SCHEDULED QUERY scheduled_rebuild
EVERY 10 MINUTES AS
ALTER MATERIALIZED VIEW mv_recently_hired REBUILD;
```

2. Define the window of time for using stale data, 13 minutes.

```
SET hive.materializedview.rewriting.time.window=13min;
```

Creating and using a partitioned materialized view

When creating a materialized view, you can partition selected columns to improve performance. Partitioning separates the view of a table into parts, which often improves query rewrites of partition-wise joins of materialized views with tables or other materialized views.

About this task

This task assumes you created a materialized view of the emps and depts tables and assumes you created these tables. The emps table contains the following data:

empid	deptno	name	salary	hire_date
10001	101	jane doe	250000	2018-01-10
10005	100	somporn klailee	210000	2017-12-25
10006	200	jeiranan thongnopneua	175000	2018-05-05

The depts table contains the following data:

deptno	deptname	locationid
100	HR	10
101	Eng	11
200	Sup	20

In this task, you create two materialized views: one partitions data on department; the other partitions data on hire date. You select data, filtered by department,from the original table, not from either one of the materialized views. The explain plan shows that Hive rewrites your query for efficiency to select data from the materialized view that partitions data by department. In this task, you also see the effects of rebuilding a materialized view.

Procedure

1. Create a materialized view of the emps table that partitions data into departments.

```
CREATE MATERIALIZED VIEW partition_mv_1 PARTITIONED ON (deptno)
AS SELECT hire_date, deptno FROM emps WHERE deptno > 100 AND deptno < 200;
```

2. Create a second materialized view that partitions the data on the hire date instead of the department number.

```
CREATE MATERIALIZED VIEW partition_mv_2 PARTITIONED ON (hire_date)
  AS SELECT deptno, hire_date FROM emps where deptno > 100 AND deptno < 2
00;
```

3. Generate an extended explain plan by selecting data for department 101 directly from the emps table without using the materialized view.

```
EXPLAIN EXTENDED SELECT deptno, hire_date FROM emps_a where deptno = 101;
```

The explain plan shows that Hive rewrites your query for efficiency, using the better of the two materialized views for the job: partition_mv_1.

```
+------------------------------------------------------+
|                      Explain                         |
+------------------------------------------------------+
| OPTIMIZED SQL: SELECT CAST(101 AS INTEGER) AS `deptno`, `hire_date` |
| FROM `default`.`partition_mv_1`                      |
| WHERE 101 = `deptno`                                 |
| STAGE DEPENDENCIES:                                  |
|    Stage-0 is a root stage                           |
...
```

4. Correct Jane Doe's hire date to February 12, 2018, rebuild one of the materialized views, but not the other, and compare contents of both materialized views.

```
INSERT INTO emps VALUES (10001,101,'jane doe',250000,'2018-02-12');
ALTER MATERIALIZED VIEW partition_mv_1 REBUILD;
SELECT * FROM partition_mv_1 where deptno = 101;
```

```
SELECT * FROM partition_mv_2 where deptno = 101;
```

The output of selecting the rebuilt partition_mv_1 includes the original row and newly inserted row because INSERT does not perform in-place updates (overwrites).

```
+--------------------------+--------------------------+
| partition_mv_1.hire_date | partition_mv_1.deptno    |
+--------------------------+--------------------------+
| 2018-01-10 00:00:00.0    | 101                      |
| 2018-02-12 00:00:00.0    | 101                      |
+--------------------------+--------------------------+
```

The output from the other partition is stale because you did not rebuild it:

```
+-----------------------+---------------------------+
| partition_mv_2.deptno | partition_mv_2.hire_date  |
+-----------------------+---------------------------+
| 101                   | 2018-01-10 00:00:00.0     |
+-----------------------+---------------------------+
```

5. Create a second employees table and a materialized view of the tables joined on the department number.

```
CREATE TABLE emps2 TBLPROPERTIES AS SELECT * FROM emps;
CREATE MATERIALIZED VIEW partition_mv_3 PARTITIONED ON (deptno) AS
   SELECT emps.hire_date, emps.deptno FROM emps, emps2
   WHERE emps.deptno = emps2.deptno
   AND emps.deptno > 100 AND emps.deptno < 200;
```

6. Generate an explain plan that joins tables emps and emps2 on department number using a query that omits the partitioned materialized view.

```
EXPLAIN EXTENDED SELECT emps.hire_date, emps.deptno FROM emps, emps2
   WHERE emps.deptno = emps2.deptno
   AND emps.deptno > 100 AND emps.deptno < 200;
```

The output shows that Hive rewrites the query to use the partitioned materialized view partition_mv_3 even though your query omitted the materialized view.

7. Verify that the partition_mv_3 sets up the partition for deptno=101 for partition_mv_3.

```
SHOW PARTITIONS partition_mv_3;
```

Output is:

```
+--------------+
|  partition   |
+--------------+
| deptno=101   |
+--------------+
```

Materialized view commands

The syntax and examples of materialized view commands provide details for using materialized views in Unified Analytics.

ALTER MATERIALIZED VIEW REBUILD

You must rebuild the materialized view to keep it up-to-date when changes to the data occur.

Syntax

```
ALTER MATERIALIZED VIEW [db_name.]materialized_view_name REBUILD;
```

db_name.materialized_view_name

> The database name followed by the name of the materialized view in dot notation.

Description

A rewrite of a query based on a stale materialized view does not occur automatically. If you want a rewrite of a stale or possibly stale materialized view, you can force a rewrite. For example, you might want to use the contents of a materialized view of a non-transactional table because the freshness of such a table is unknown. To enable rewriting of a query based on a stale materialized view, you can run the rebuild operation periodically and set the following property: hive.materializedview.rewriting.time.window. For example, SET   hive.materializedview.rewriting.time.window=10min;

Example

ALTER MATERIALIZED VIEW mydb.mv1 REBUILD;

ALTER MATERIALIZED VIEW REWRITE

You can enable or disable the rewriting of queries based on a particular materialized view.

Syntax

```
ALTER MATERIALIZED VIEW [db_name.]materialized_view_name ENABLE|DISABLE REWR
ITE;
```

db_name.materialized_view_name

> The database name followed by the name for the materialized view in dot notation.

Description

To optimize performance, by default, Hive rewrites a query based on materialized views. You can change this behavior to manage query planning and execution manually. By setting the hive.materializedview.rewriting global property, you can manage query rewriting based on materialized views for all queries.

Example

ALTER MATERIALIZED VIEW mydb.mv1 DISABLE REWRITE;

CREATE MATERIALIZED VIEW

If you are familiar with the CREATE TABLE AS SELECT (CTAS) statement, you can quickly master how to create a materialized view.

Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] [db_name.]materialized_view_name
    [DISABLE REWRITE]
    [COMMENT materialized_view_comment]
    [PARTITIONED ON (column_name, ...)]
    [
      [ROW FORMAT row_format]
      [STORED AS file_format]
        | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (ser
de_property_name=serde_property_value, ...)]
    ]
```

```
    [LOCATION file_path]
    [TBLPROPERTIES (tbl_property_name=tbl_property_value, ...)]
    AS
    <query>;
```

Required Parameters

query

> The query to run for results that populate the contents of the materialized view.

Optional Parameters

db_name.materialized_view_name

> The database name followed by a name, unique among materialized view names, for the materialized view. The name must be a valid a table name, including case-insensitive alphanumeric and underscore characters.

materialized_view_comment

> A string literal enclosed in single quotation marks.

column_name

> A key that determines how to do the partitioning, which divides the view of the table into parts.

'storage.handler.class.name'

> The name of a storage handler, such as org.apache.hadoop.hive.druid.DruidStorageHandler, that conforms to the Apache Hive specifications for storage handlers in a table definition that uses the STORED BY clause. The default is hive.materializedview.fileformat.

serde_property_name

> A property supported by SERDEPROPERTIES that you specify as part of the STORED BY clause. The property is passed to the serde provided by the storage handler. When not specified, Hive uses the default hive.materializedview.serde.

serde_property_value

> A value of the SERDEPROPERTIES property.

file_path

> The location on the file system for storing the materialized view.

tbl_property_name

> A key that conforms to the Apache Hive specification for TBLPROPERTIES keys in a table.

tbl_property_value

> The value of a TBLPROPERTIES key.

Usage

The materialized view creation statement meets the criteria of being atomic: it does not return incomplete results. By default, the optimizer uses materialized views to rewrite the query. You can store a materialized view in an external storage system using the STORED AS clause followed by a valid storage handler class name. You can set the DISABLE REWRITE option to alter automatic rewriting of the query at materialized view creation time. The table on which you base the materialized view, src in the example below, must be an ACID, managed table.

Examples

```
CREATE MATERIALIZED VIEW druid_t
  STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
  AS SELECT a, b, c
```

```
   FROM src;
```

```
CREATE MATERIALIZED VIEW mv4
   LOCATION '/user/csso_max'
   AS SELECT empid, deptname, hire_date
   FROM emps JOIN depts
   ON (emps.deptno = depts.deptno)
   WHERE hire_date >= '2017-01-01';
```

DESCRIBE EXTENDED and DESCRIBE FORMATTED

You can get extensive formatted and unformatted information about a materialized view.

Syntax

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]materialized_view_name;
```

db_name

>   The database name.

materialized_view_name

>   The name of the materialized view.

Examples

Get summary, details, and formatted information about the materialized view in the default database and its partitions.

```
DESCRIBE FORMATTED default.partition_mv_1;
```

Example output is:

col_name	data_type	comment
# col_name	data_type	comment
name	varchar(256)	
	NULL	NULL
# Partition Information	NULL	NULL
# col_name	data_type	comment
deptno	int	
	NULL	NULL
# Detailed Table Information	NULL	NULL
Database:	default	NULL
OwnerType:	USER	NULL
Owner:	hive	NULL
CreateTime:	Wed Aug 24 19:46:08 UTC 2022	NULL
LastAccessTime:	UNKNOWN	NULL
Retention:	0	NULL
Location:	hdfs://myserver:8020/warehouse/ tables pace/managed/hive/partition_mv_1	NULL
Table Type:	MATERIALIZED_VIEW	NULL
Table Parameters:	NULL	NULL

col_name	data_type	comment
	COLUMN_STATS_ACCURATE	{\"BASIC_STATS\":\"true\"}
	bucketing_version	2
	numFiles	2
	numPartitions	2
	numRows	4
	rawDataSize	380
	totalSize	585
	transient_lastDdlTime	1534967168
	NULL	NULL
# Storage Information	NULL	NULL
SerDe Library:	org.apache.hadoop.hive.ql.io.orc.OrcSerde	NULL
InputFormat:	org.apache.hadoop.hive.ql.io.orc.OrcInputFormat	NULL
OutputFormat:	org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat	NULL
Compressed:	No	NULL
Num Buckets:	-1	NULL
Bucket Columns:	[]	NULL
Sort Columns:	[]	NULL
	NULL	NULL
# Materialized View Information	NULL	NULL
Original Query:	SELECT hire_date, deptno FROM emps WHERE deptno > 100 AND deptno < 200	NULL
Expanded Query:	SELECT \`hire_date\`, \`deptno\` FROM (SELECT \`emps\`.\`hire_date\`, \`emps\`.\`deptno\` FROM \`default\`.\`emps\` WHERE \`emps\`.\`deptno\` > 100 AND \`emps\`.\`deptno\` < 200) \`default.partition_mv_1\`	NULL
Rewrite Enabled:	Yes	NULL
Outdated for Rewriting:	No	NULL

DROP MATERIALIZED VIEW

You can avoid making a table name unusable by dropping a dependent materialized view before dropping a table.

Syntax

```
DROP MATERIALIZED VIEW [db_name.]materialized_view_name;
```

db_name.materialized_view_name

The database name followed by a name for the materialized view in dot notation.

Description

Dropping a table that is used by a materialized view is not allowed and prevents you from creating another table of the same name. You must drop the materialized view before dropping the tables.

Example

DROP MATERIALIZED VIEW mydb.mv1;

SHOW MATERIALIZED VIEWS

You can list all materialized views in the current database or in another database. You can filter a list of materialized views in a specified database using regular expression wildcards.

Syntax

```
SHOW MATERIALIZED VIEWS [IN db_name];
```

db_name

  The database name.

'identifier_with_wildcards'

  The name of the materialized view or a regular expression consisting of part of the name plus
  wildcards. The asterisk and pipe (* and |) wildcards are supported. Use single quotation marks to
  enclose the identifier.

Examples

SHOW MATERIALIZED VIEWS;

```
+-----------------+--------------------+-----------------+
|     mv_name     |   rewrite_enabled  |       mode      |
+-----------------+--------------------+-----------------+
| # MV Name       | Rewriting Enabled  | Mode            |
| partition_mv_1  | Yes                | Manual refresh  |
| partition_mv_2  | Yes                | Manual refresh  |
| partition_mv_3  | Yes                | Manual refresh  |
+-----------------+--------------------+-----------------+
```

SHOW MATERIALIZED VIEWS '*1';

```
+-----------------+--------------------+-----------------+
|     mv_name     |   rewrite_enabled  |       mode      |
+-----------------+--------------------+-----------------+
| # MV Name       | Rewriting Enabled  | Mode            |
| partition_mv_1  | Yes                | Manual refresh  |
|                 | NULL               | NULL            |
+-----------------+--------------------+-----------------+
```

Materialized view recommendations introduction

Materialized views can improve query performance greatly, but deciding which query results to materialize is difficult. You can use the materialized view recommender command-line interface to help you decide which query results to materialize.

The recommender design is based on Agrawal et al. - Automated Selection of Materialized Views and Indexes for SQL Databases (VLDB 2000). The recommender takes a set of Hive queries as input and creates materialized views as output by exploring many table subsets and syntactical variants for each view candidate.

Typically, instead of considering a single, or just a few queries for materializing, you are considering an entire workload. Even the most experienced database experts have difficulty. There are many factors to consider, such as view maintenance and effectiveness. Making a bad decision about what to materialize can make performance worse.

You provide a set of queries or explain plans to the recommender, and then run the Workload Insights (wi) script to get recommendations. You can get recommendations online or offline, depending on the type of input you have:

- Online: input = queries only
- Offline: input = explain plans

If you have explain plans to provide as input to the Workload Insights (wi) script, you can run the script offline; otherwise, you have to provide the HiveServer endpoint for example, jdbc:hive2://myhost:10000, to the script. In lieu of explain plans, the recommender needs to gather information about the query, such as the number of rows of each query, how long the query ran, and other statistics, from HiveServer.

The following diagram shows the input and output of the recommender:
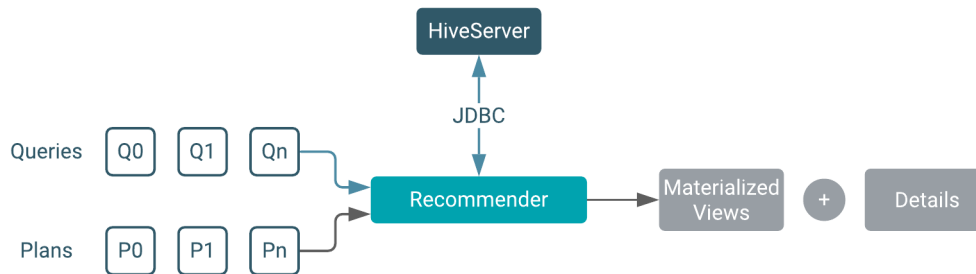
How much detail you receive in a recommendation depends on your configuration. Details include the number of recommendations and other information, such as why a particular view is recommended and how to exploit the view.

Workload Insights input requirements

You must have one of the following inputs to the Workload Insights (wi) script:

- One or more directories containing one or more select and aggregation queries in .sql files and multiple queries in files, separated by a semicolon (;).

  A .sql file name suffix is required.
- One or more directories containing an explain plan in .json files. A .json file name suffix is required.

Speeding up queries using BI mode

You can use BI mode to automatically rewrite incoming queries to be answered approximately using Apache DataSketches. BI mode can be useful when using data visualization BI tools such as Tableau.

About this task

Queries of very large data sets with a number of distinct values often take too long to return results. If you can accept approximate results, using DataSketch algorithms can save significant time. BI mode integrated with DataSketches accelerates query execution while decreasing resource utilization. BI mode can rewrite a wide range of SQL analytical queries, including COUNT(DISTINCT), CUME_DIST, RANK, and NTILE.

When you enable BI mode to use Apache DataSketches approximations, a materialized view automatically calls DataSketches for some types of operations. You can pre-compute data sketches using materialized views, and then rely on Unified Analytics BI mode to rewrite algorithms to answer queries directly from those materialized views. This accelerates query execution by orders of magnitude without any change to your original queries.
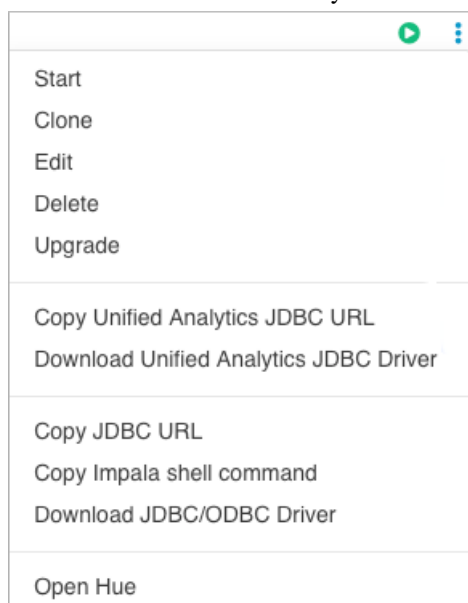
Before you begin

- Choose an authorization model.
- Configure authenticated users for querying Hive through JDBC or ODBC driver. For example, set up a Ranger policy.

Get the JDBC driver

To query, analyze, and visualize data stored in CDW Unified Analytics, you must download the Unified Analytics JDBC driver to your cluster and provide it to your client. You can use the latest version of drivers provided by Cloudera to connect Unified Analytics to Business Intelligence (BI) tools.

Procedure

1. Click ⋮ on the Impala Virtual Warehouse tile.

2. Click Download Unified Analytics JDBC Driver.

Connect to Unified Analytics with a BI tool

After you downloaded the JDBC driver, you must set up a BI tool such as Tableau to connect with the Unified Analytics engine.

Before you begin

Before you can use Tableau with an Impala/Hive Virtual Warehouses, you must have created a Database Catalog that is populated with data. You have the option to populate your Database Catalog with sample data when you create it. You must also create a Impala/Hive Virtual Warehouse and enable Unified Analytics, which is configured to connect to the Database Catalog that is populated with data.

Procedure

1. Click ⋮ on the Impala Virtual Warehouse tile.

2. Select Copy Unified Analytics JDBC URL.

3. Paste the copied JDBC URL into a text file. It should look similar to the following:

```
jdbc:hive2://<your-virtual-warehouse>.<your-environment>.<dwx.company.co
m>/default;transportMode=http;httpPath=cliservice;ssl=true;retries=3
```

4. From the text file where you just pasted the URL, copy the host name from the JDBC URL to your system's clipboard. For example, in the URL shown in Step 3, the host name is:

```
<your-virtual-warehouse>.<your-environment>.<yourdomain.com>
```
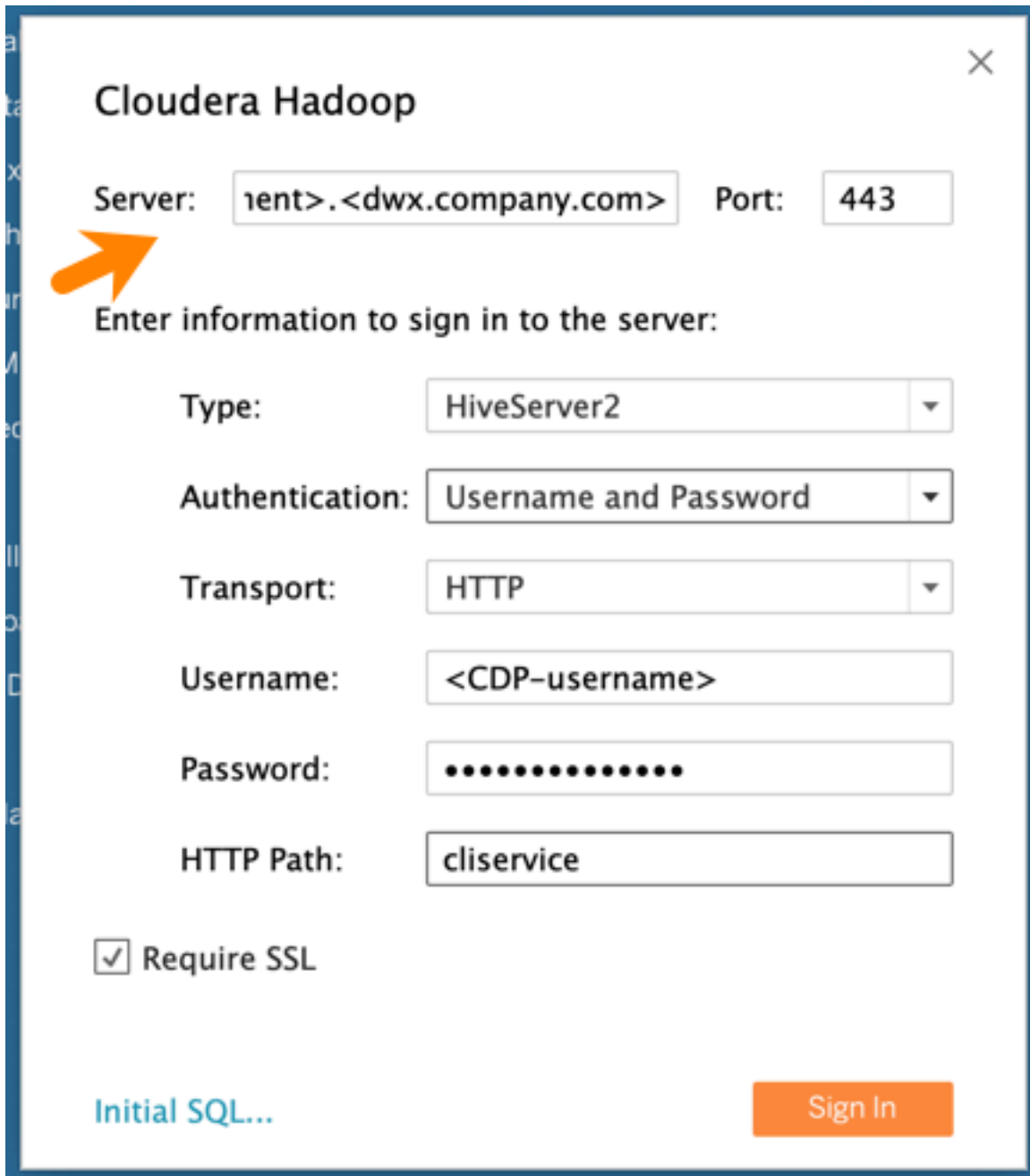
5. Start Tableau and navigate to ConnectMore…Cloudera Hadoop :
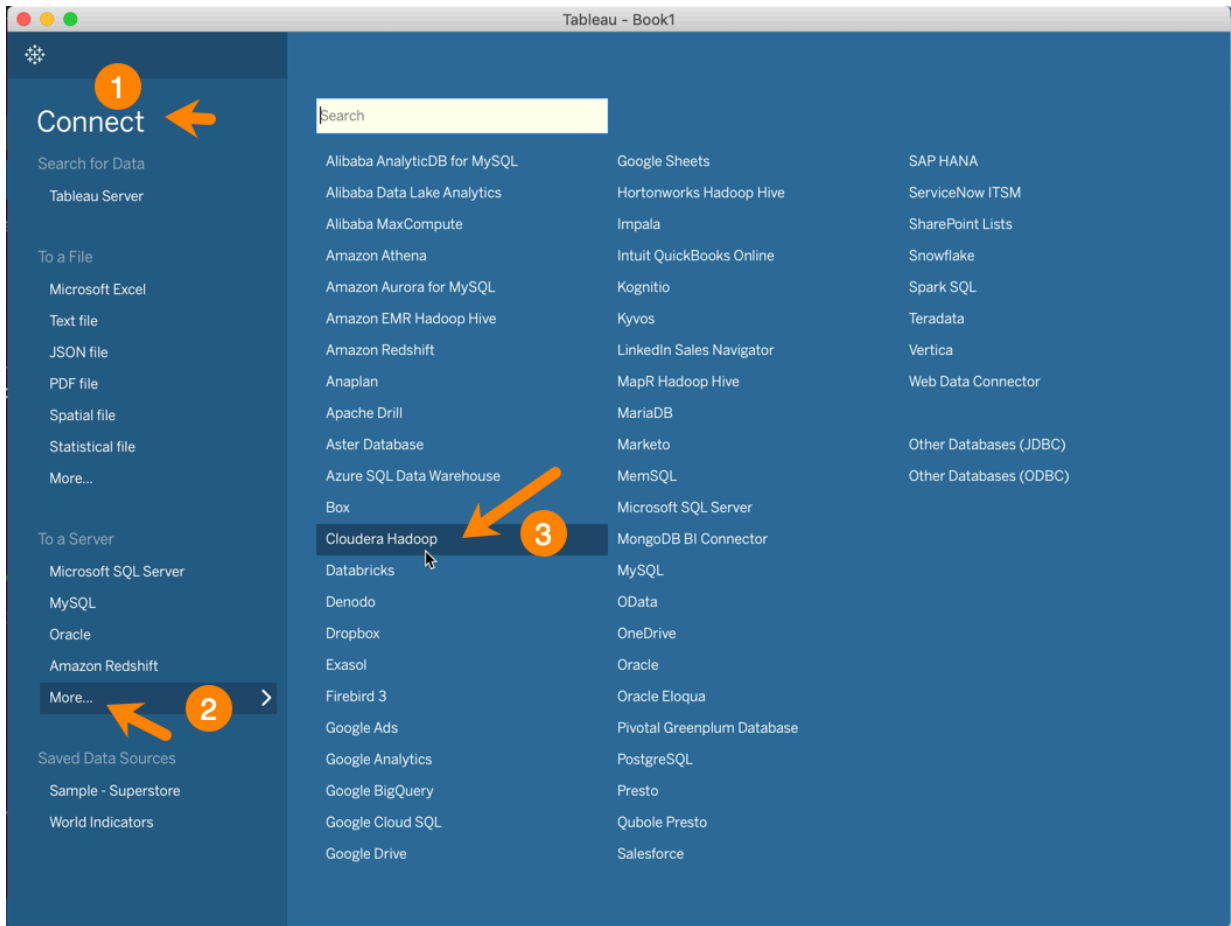
This launches the Cloudera Hadoop dialog box.

6. In the Tableau Cloudera Hadoop dialog box, paste the host name you copied to your clipboard in Step 4 into the Server field:

7. Then in the Tableau Cloudera Hadoop dialog box, set the following other options:

- Port: 443
- Type: HiveServer2
- Authentication: Username and Password
- Transport: HTTP
- Username: Username you use to connect to the CDP Data Warehouse service.
- Password: Password you use to connect to the CDP Data Warehouse service.
- HTTP Path: cliservice
- Require SSL: Make sure this is checked.

8. Click Sign In.

Enable BI mode to rewrite queries automatically

You must enable BI mode to combine the BI capability with the power of transparent materialized view rewritings.

About this task

If you want your incoming queries, at different granularity levels, to be answered directly from the materialized view rather than being computed from the source tables, you must enable BI mode. BI mode is disabled by default.

Procedure

1. To enable BI mode, execute the following statement from Hue or your preferred client. set hive.optimize.bi.ena bled=true;

2. Create a materialized view that stores one of the supported DataSketches (for example, HLL or KLL) on the columns for which the aggregations will be computed during query processing.
   The following example shows creating a materialized view for computing the number of unique visitors per region for a given website.

```
CREATE MATERIALIZED VIEW AS
SELECT country, state, city, ds_hll_sketch(userid)
FROM visitors
GROUP BY country, state, city;
```

Results

With the BI mode enabled, any incoming queries will be sliced and diced at different granular levels to be answered directly from the materialized view.

```
SELECT country, count(distinct userid)
FROM visitors
GROUP BY country;

SELECT country, state, count(distinct userid)
FROM visitors
GROUP BY country, state;

SELECT country, state, count(distinct userid)
FROM visitors
WHERE country = 'USA'
 GROUP BY country, state;
```

Using roll ups and grouping sets

If your work involves heavy OLAP queries, you need to know about grouping sets and roll ups. Unified Analytics supports grouping sets with rollups and grouping functions.

Before you begin
You created a Unified Analytics Virtual Warehouse based on a Database Catalog that loads the sample data to run queries of airline data.

Procedure

1. Click   Open Hue on the Virtual Warehouse tile.
2. Select a database.

3. In Hue, select either the Impala or Hive editor.

4. Run a query that determines the number of flights per country and state.

```
SELECT
  a.country,
  a.state,
  grouping(a.country)
  grouping(a.state)
  SUM(f.Distance),
  COUNT(*)
FROM
  flights f,
  airports a
WHERE
  f.origin = a.iata
GROUP BY
  a.country,
  a.state
WITH ROLLUP;
```

5. Run a query that defines custom grouping sets: country and country/state.

```
SELECT
  a.country,
  a.state,
  grouping(a.country)
  grouping(a.state)
  SUM(f.Distance),
  COUNT(*)
FROM
  flights f,
  airports a
WHERE
  f.origin = a.iata
GROUP BY
  a.country,
  a.state
GROUPING SETS (
  (a.country),
  (a.country, a.state)
);
```

Using set operations

When you want to combine separate query results into a single result, you need to know about the INTERSECT and EXCEPT clauses in addition to UNION.

About this task

In this task, you run a query to find the number of Boeing planes leaving from DFW in 2005. The query filters data using an intersect on the tail number, tailnum, with the unique identification of a plane, planes.manufacturer.

Before you begin
You created a Unified Analytics VW that loads the sample data to run queries of airline data.

Procedure

1. Click ⋮ Open Hue on the Virtual Warehouse tile.

2. Select a database.

For example, select TBD.

3. In Hue, select either the Impala or Hive editor.

4. Enter a query that intersects the tail number and type of plane.

```
SELECT COUNT(*)
FROM
  SELECT tailnum
  FROM flights
  WHERE
    flights.origin = 'DFW' AND flights.`year`= 2005
  INTERSECT
  SELECT tailnum
  FROM planes
  WHERE
    planes.manufacturer = 'BOEING'
) subq;
```

5. Run a query that returns the planes not manufactured by Boeing leaving from DFW.

```
SELECT COUNT(*)
FROM
  SELECT tailnum
  FROM flights
  WHERE
    flights.origin = 'DFW' AND flights.`year`= 2005
  EXCEPT
  SELECT tailnum
  FROM planes
  WHERE
    planes.manufacturer = 'BOEING'
) subq;
```

Using anti joins

Unified Analytics supports anti join queries to find the rows in a table that do not match rows in another table. The performance of queries involving NULL filters, "not exists" clauses, and "not in" clauses" are improved using the anti join optimization.

About this task

In this task, run an anti join query. For example:

```
 SELECT * FROM left_table t1 LEFT JOIN right_table t2 ON t1.id = t2.id WHERE
 t2.id IS NULL;
```

Procedure

1. In the Data Warehouse service, click Overview in the left navigation pane.

2. In Overview, i Virtual Warehouses, select your Virtual Warehouse, click ⋮ , and select Edit.

3. In Configurations, select hive-site, add the hive.auto.convert.anti.join property, and set it to true.

4. Open Hue, and create two tables, one having more rows than the other.

```
create table tt1 (order_num bigint, price decimal(5, 2));

create table tt2 (order_num bigint);
```

5. Insert several, say five, rows of values (order number, price) in the first table and fewer, say 2, rows of values (order number only) in the second table.

```
insert into tt1 values (42, 300.50), (141, 250.25), (150, 85.00), (150,
500.44), (166, 125.65);
insert into tt2 values (150), (166);
```

6. Take a look at the content of the first table.

```
select * from tt1;
```

```
tt1.order_num tt1.price
42               300.50
141               250.25
150                85.00
150               500.44
166               125.65
```

7. Take a look at the content of the second table.

```
select * from tt2;
```

```
tt2.order_num
150
166
```

8. Perform an equality left join on the order numbers in the tables.

```
select * from tt1 t1 LEFT JOIN tt2 t2 ON t1.order_num=t2.order_num;
```

```
t1.order_num t1.price t2.order_num
42               300.50    NULL
141               250.25    NULL
150                85.00    150
150               500.44    150
166               125.65    166
```

9. Add a null filter to the join query to anti join the tables.

```
select * from tt1 t1 LEFT JOIN tt2 t2 ON t1.order_num=t2.order_num where
 t2.order_num is NULL;
```

The anti join returns the rows having NULL in the price column.

```
t1.order_num t1.price t2.order_num
42               300.50    NULL
141               250.25    NULL
```

Creating a table from Parquet data

In Unified Analytics, you can base a new table on a schema in a Parquet file using the CREATE TABLE LIKE FILE PARQUET statement. You are likely familiar with this statement if you have been an Impala user. The Impala counterpart CREATE TABLE LIKE PARQUET lacks the keyword LIKE, but is functionally similar.

About this task

The column definitions in the new table are inferred from the Parquet data file when you create a table like Parquet in Unified Analytics. Set the following table property for creating the table:

```
hive.parquet.infer.binary.as = <value>
```

Where <value> is binary (the default) or string.

This property determines the interpretation of the unannotated Parquet binary type. Some systems expect binary to be interpreted as string.

Use the following syntax for creating a table like Parquet.

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name LIKE PARQUET FI
LE 'object_storage_path_of_parquet_file'
[PARTITIONED BY (col_name data_type [COMMENT 'col_comment'], ...)]
[SORT BY ([column [, column ...]])] [COMMENT 'table_comment'] [ROW FORMAT
row_format]
[WITH SERDEPROPERTIES ('key1'='value1', 'key2'='value2', ...)] [STORED AS f
ile_format]
[LOCATION 'object_storage_path']
[TBLPROPERTIES ('key1'='value1', 'key2'='value2', ...)]
```

In this task, you create a table like parquet that uses the default value binary for hive.parquet.infer.binary.as.

Procedure

Create a table based on a parquet table.

```
CREATE TABLE ua_table LIKE FILE PARQUET 's3a://testbucket/files/schema.p
arq';
```

Querying correlated data

You can query one table relative to the data in another table.

About this task
A correlated query contains a query predicate with the equals (=) operator. One side of the operator must reference at least one column from the parent query and the other side must reference at least one column from the subquery. An uncorrelated query does not reference any columns in the parent query.

Procedure

Select all state and net_payments values from the transfer_payments table for years during which the value of the state column in the transfer_payments table matches the value of the state column in the us_census table.

```
SELECT state, net_payments
FROM transfer_payments
```

```
WHERE EXISTS
    (SELECT year
    FROM us_census
    WHERE transfer_payments.state = us_census.state);
```

This query is correlated because one side of the equals predicate operator in the subquery references the state column in the transfer_payments table in the parent query and the other side of the operator references the state column in the us_census table.

This statement includes a conjunct in the WHERE clause.

A conjunct is equivalent to the AND condition, while a disjunct is the equivalent of the OR condition The following subquery contains a conjunct:

... WHERE transfer_payments.year = "2018" AND us_census.state =   "california"

The following subquery contains a disjunct:

... WHERE transfer_payments.year = "2018" OR us_census.state =   "california"

Subquery restrictions

To construct queries efficiently, you must understand the restrictions of subqueries in WHERE clauses.

- Subqueries must appear on the right side of an expression.
- Nested subqueries are not supported.
- Subquery predicates must appear as top-level conjuncts.
- Subqueries support four logical operators in query predicates: IN, NOT IN, EXISTS, and NOT EXISTS.
- The IN and NOT IN logical operators may select only one column in a WHERE clause subquery.
- The EXISTS and NOT EXISTS operators must have at least one correlated predicate.
- The left side of a subquery must qualify all references to table columns.
- References to columns in the parent query are allowed only in the WHERE clause of the subquery.
- Subquery predicates that reference a column in a parent query must use the equals (=) predicate operator.
- Subquery predicates may refer only to columns in the parent query.
- Correlated subqueries with an implied GROUP BY statement may return only one row.
- All unqualified references to columns in a subquery must resolve to tables in the subquery.
- Correlated subqueries cannot contain windowing clauses.

Comparing tables using ANY/SOME/ALL

You learn how to use quantified comparison predicates (ANY/SOME/ALL) in non-correlated subqueries according to the SQL standard. SOME is any alias for ANY.

About this task
You can use one of the following operators with a comparison predicate:

- >
- <
- >=
- <=
- <>
- =

ALL:

- If the table is empty, or the comparison is true for every row in subquery table, the predicate is true for that predicand.

- If the comparison is false for at least one row, the predicate is false.

SOME or ANY:

- If the comparison is true for at least one row in the subquery table, the predicate is true for that predicand.
- If the table is empty or the comparison is false for each row in subquery table, the predicate is false.

If the comparison is neither true nor false, the result is undefined.

For example, you run the following query to match any value in c2 of tbl equal to any value in c1 from the same tbl:

```
select c1 from tbl where c1 = ANY (select c2 from tbl);
```

You run the following query to match all values in c1 of tbl not equal to any value in c2 from the same tbl.

```
select c1 from tbl where c1 <> ALL (select c2 from tbl);
```

Handling ETL jobs

Unified Analytics provides DML support for extract, transform, and load tasks. ETL jobs, identified by their size, run in query isolation mode if enabled.

About this task

Unified Analytics supports insert, update, merge, and delete statements for performing ETL tasks.

> **Note:** Query isolation feature for Unified Analytics is in Technical Preview in 1.5.0 and not recommended for use in production deployments. Cloudera recommends that you try this feature in test or development environments.

Procedure

1. Create a Unified Analytics Virtual Warehouse (Hive), turning on the Query Isolation option.

2. Click Create.

What to do next
Insert, update, and delete data as described in subsequent topics.

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 an ACID table, use the Optimized Row Columnar (ORC) storage format. To insert data into a non-ACID table, you can use other supported formats. You can specify partitioning as shown in the following syntax:

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

```
CREATE TABLE pageviews (userid VARCHAR(64), link STRING, origin STRING)
PARTITIONED BY (datestamp STRING) CLUSTERED BY (userid) INTO 256 BUCKETS;
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.

Updating data in a table

The syntax describes the UPDATE statement you use 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 statement. Partition columns cannot be updated.

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;

Merging data in tables

A sample statement shows how you can conditionally insert existing data in Hive tables using the ACID MERGE statement. Additional merge operations are mentioned.

About this task
The MERGE statement is based on ANSI-standard SQL.

Procedure

1. Construct a query to update the customers' names and states in customer target table to match the names and states of customers having the same IDs in the new_customer_stage source table.

2. Enhance the query to insert data from new_customer_stage table into the customer table if none already exists.

   Update or delete data using MERGE in a similar manner.

   ```
   MERGE INTO customer USING (SELECT * FROM new_customer_stage) AS sub ON s
   ub.id = customer.id
   WHEN MATCHED THEN UPDATE SET name = sub.name, state = sub.state
   WHEN NOT MATCHED THEN INSERT VALUES (sub.id, sub.name, sub.state);
   ```

   Note: You can map specific columns in the INSERT clause of the query instead of passing values (including null) for columns in the target table that do not have any data to insert. The unspecified columns in the INSERT clause are either mapped to null or use default constraints, if any.

   For example, you can construct the INSERT clause as WHEN NOT MATCHED THEN INSERT VAL UES (customer.id=sub.id,   customer.name=sub.name, customer.state=sub.state) instead of WHEN NOT MATCHED THEN INSERT VALUES (sub.id, sub.name, 'null',   sub.state).

Deleting data from a table

You use the DELETE statement to delete data already written to an ACID table.

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