DAWD 02-2 - Demo - Ingesting Data

The two fields below are used to customize queries used in this course. Enter your schema (database) name and username, and press "Enter" to populate necessary information in the queries on this page.

Schema Name:	hive_metastore.class_013_odg7_da_dawd
Username:	class+013@databricks.com

Lesson Objective

At the end of this lesson, you will be able to:

- Describe how to ingest data

Ingest Data Using Databricks SQL

We can ingest data into Databricks SQL in a variety of ways. In this first example, we are going to start with a .csv file that exists in an object store and finish with a Delta table that contains all the data from that .csv file.

1. Run the code below.

There are several things to note in these commands:

- Since the file we are ingesting is a .csv file, we state `USING csv`
- We are setting three options:
    - path - the path to the object store
    - header - whether or not the .csv file contains a header row
    - inferSchema - whether or not Databricks should infer the schema from the contents of the file
- We are creating a second table, with default settings from the contents of the first table, using a CTAS statement.

The reason we are creating a second table that is a copy of the first table is because we want the resulting table to be in Delta format, which gives us the most options. Because the second table uses default options, it will be a Delta table.

We can see this in the Data Explorer

2. In the sidebar menu, click "Data"
3. If needed, select your schema in the dropdown
4. Select `web_events_csv` and note it is a CSV table
5. Select `web_events` and note it is a Delta table

Once we are finished with the `web_events_csv` table, we can drop it from the schema.

```
USE hive_metastore.class_013_odg7_da_dawd;
DROP TABLE IF EXISTS web_events_csv;
CREATE TABLE web_events_csv
    USING csv
    OPTIONS (
        path='wasb://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/web_events/web_events.csv',
        header="true",
        inferSchema="true"
    );
DROP TABLE IF EXISTS web_events;
CREATE OR REPLACE TABLE web_events AS
    SELECT * FROM web_events_csv;
```

Copy

The LOCATION Keyword

For those of you taking the full course, we talked about the `LOCATION` keyword in the last module. This section is a review, and you can safely skip it. If your taking the short course, or if you need to review the use of `LOCATION`, keep reading.

We can ingest data in-place using the `LOCATION` keyword to create an external/unmanaged table. There is a dataset, called Sales, that is currently in an object store. Running the command below creates an external table that is associated with this dataset.

6. Run the code below.

The table's data is stored in the external location, but the table itself is registered in the metastore. We can query the data just like any other table in the schema.

```
USE hive_metastore.class_013_odg7_da_dawd;
DROP TABLE IF EXISTS external_table;
CREATE TABLE external_table
    LOCATION 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-databricks/v02/retail-org/sales/sales_delta';
SELECT * FROM external_table;
```

Copy

COPY INTO

We use `COPY INTO` to load data from a file location into a Delta table. `COPY INTO` is a re-triable and idempotent operation, so files in the source location that have already been loaded are skipped.

7. Run the code below.

The first command creates an empty Delta table. Note that you must specify a schema when creating an empty Delta table.

The second command copies data from an object store location into the `gym_logs` table. Note that the file type for the files in the object store location is specified as "JSON". The last part of the `COPY INTO` command is a file name, a list of file names, or a directory of files.

```
USE hive_metastore.class_013_odg7_da_dawd;
DROP TABLE IF EXISTS gym_logs;
CREATE TABLE gym_logs (first_timestamp DOUBLE, gym Long, last_timestamp DOUBLE, mac STRING);
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
    FILES = ('20191201_2.json');
```

Copy

We can run the same `COPY INTO` command as above and add a second one. The first one will be skipped because the file has already been loaded.

8. Run the code below.

```
USE hive_metastore.class_013_odg7_da_dawd;
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
    FILES = ('20191201_2.json');
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
    FILES = ('20191201_3.json');
```

Copy

Do a count on the table you just populated using COPY INTO. Then, rerun the COPY INTO code above, and do another count. Notice that 'COPY INTO' will not input data from files that have already been ingested using COPY INTO. The count stays the same.

9. Run the code below.

```
USE hive_metastore.class_013_odg7_da_dawd;
SELECT count(*) FROM gym_logs;
```

Copy

Next, let's add another `COPY INTO` command, but this time, we will use the `PATTERN` option. This allows us to load any file that fits a specific pattern.

10. Run the code below.

```
USE hive_metastore.class_013_odg7_da_dawd;
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
    FILES = ('20191201_2.json');
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
    FILES = ('20191201_3.json');
COPY INTO gym_logs
    FROM 'wasbs://courseware@dbacademy.blob.core.windows.net/data-analysis-with-
databricks/v02/gym_logs/gym_logs_json'
    FILEFORMAT = JSON
```

Copy

Privileges

Data object owners and Databricks administrators can grant and revoke a variety of privileges on securable objects. These objects include functions, files, tables, views, and more. These privileges can be granted using SQL or using the Data Explorer.

11. Run the code below.

This statement returns the privileges granted on this schema.

```
SHOW GRANT ON SCHEMA hive_metastore.class_013_odg7_da_dawd;
```

Copy

GRANT

If we wish to grant privileges to a user, we can run `GRANT commands for the specific privileges we wish to grant. The privileges available include USAGE, SELECT, CREATE, READ FILES, and more.`

12. Run the code below.

We need to start by granting `USAGE to a user. We can then grant other privileges. If we want to grant all privileges, we can use GRANT ALL PRIVILEGES.`

```
GRANT USAGE ON SCHEMA hive_metastore.class_013_odg7_da_dawd TO `class+013@databricks.com`;
GRANT SELECT ON SCHEMA hive_metastore.class_013_odg7_da_dawd TO `users`;
```

Copy

REVOKE

We can revoke privileges on securable objects in the exact same way.

We don't want to actually run the command because we don't want to try to revoke our own privileges, but here is the command:

```
REVOKE ALL PRIVILEGES ON SCHEMA `schema_name` from `user_name`;
```

Data Explorer

While we can certainly grant and revoke privileges using SQL, we can also use the Data Explorer.

13. Click "Data" in the sidebar menu.
14. If needed, select your schema in the dropdown
15. Select the table, "gym_logs" from the list
16. Click "Permissions"
17. Use the "Grant" and "Revoke" buttons to change permission settings.

Note that students can view and change permissions on this table because, since they created the table, they are the owner. Only owners and admins can view and change permissions.

18. Select the "flight_delays" table.
19. Click "Permissions"

    Note that, since the student is not the owner of the table, they can't view and change permissions.
