databricks Academy

# DAWD 02-5-1 - Demo - Delta Commands in Databricks SQL

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 write commands for working with Delta tables in Databricks SQL

### SELECT on Delta Tables

So far, the SQL commands we have used are generic to most flavors of SQL. In the next few queries, we are going to look at commands that are specific to using `SELECT` on Delta tables.

Delta tables keep a log of changes that we can view by running the command below.

1. Run the code below.

    After running `DESCRIBE HISTORY`, we can see that we are on version number 0 and we can see a timestamp of when this change was made.

```
USE hive_metastore.class_013_odg7_da_dawd;
DESCRIBE HISTORY customers;
```

Copy

---

## SELECT on Delta Tables -- Updating the Table

We are going to make a change to the table.

2. Run the code below.

    The code uses an `UPDATE` statement to make a change to the table. We will be discussing `UPDATE` later on. For now, we just need to understand that a change was made to the table. We also reran our `DESCRIBE HISTORY` command, and note that we have a new version in the log, with a new timestamp.

```
USE hive_metastore.class_013_odg7_da_dawd;
UPDATE customers SET loyalty_segment = 10 WHERE loyalty_segment = 0;
DESCRIBE HISTORY customers;
```

Copy

---

## SELECT on Delta Tables -- Updating the Table back to where it was

We are going to make a change to the table.

3. Run the code below.

    The code uses an `UPDATE` statement to update the loyalty_segment back to its original value. We also reran our `DESCRIBE HISTORY` command, and note that we have a new version in the log, with a new timestamp.

```
USE hive_metastore.class_013_odg7_da_dawd;
UPDATE customers SET loyalty_segment = 0 WHERE loyalty_segment = 10;
DESCRIBE HISTORY customers;
```

Copy

---

## SELECT on Delta Tables -- VERSION AS OF

We can now use a special predicate for use with Delta tables: `VERSION AS OF`

4. Run the code below.

By using `VERSION AS OF`, we can `SELECT` from specific versions of the table. This feature of Delta tables is called "Time Travel," and is very powerful.

We can also use `TIMESTAMP AS OF` to `SELECT` based on a table's state on a specific date, and you can find more information in the documentation.

```
USE hive_metastore.class_013_odg7_da_dawd;
SELECT loyalty_segment FROM customers VERSION AS OF 1;
```

Copy

---

## MERGE INTO

Certainly, there are times when we want to insert new data but ensure we don't re-insert matched data. This is where we use `MERGE INTO`. `MERGE INTO` will merge two tables together, but you specify in which column to look for matched data and what to do when a match is found. Let's run the code and examine the command in more detail.

5. Run the code below.

We are merging the `source_suppliers` table into the `suppliers` table. After `ON` keyword, we provide the columns on which we want to look for matches. We then state what the command should do when a match is found. In this example, we are inserting the row when the two columns are not matched. Thus, if the columns match, the row is ignored. Notice that the count is the exact same as the original table. This is because the two tables have the exact same data, since we overwrote the `suppliers` table with the `source_suppliers` table earlier in the lesson.

```
USE hive_metastore.class_013_odg7_da_dawd;
MERGE INTO suppliers
    USING source_suppliers
    ON suppliers.SUPPLIER_ID = source_suppliers.SUPPLIER_ID
    WHEN NOT MATCHED THEN INSERT *;
SELECT count(*) FROM suppliers;
```

Copy