databricks 5.1-Lab-Incremental-Updates-with-Structured-Streaming-and-Delta-Lake

(https://databricks.com)

Processing Incremental Updates with Structured Streaming and Delta Lake

In this lab you'll apply your knowledge of structured streaming and Auto Loader to implement a simple multi-hop architecture.

1.0. Import Shared Utilities and Data Files

Run the following cell to setup necessary variables and clear out past runs of this notebook. Note that re-executing this cell will allow you to start the lab over.

```
%run ./Includes/5.1-Lab-setup
```

```
Dropping the database "dbacademy_baw3hg_virginia_edu_dewd_5_1"
Removing the working directory "dbfs:/user/baw3hg@virginia.edu/dbacademy/dewd/5.1"

Creating the database "dbacademy_baw3hg_virginia_edu_dewd_5_1"

Predefined Paths:
  DA.paths.working_dir: dbfs:/user/baw3hg@virginia.edu/dbacademy/dewd/5.1
  DA.paths.user_db:     dbfs:/user/baw3hg@virginia.edu/dbacademy/dewd/5.1/5_1.db
  DA.paths.checkpoints: dbfs:/user/baw3hg@virginia.edu/dbacademy/dewd/5.1/_checkpoints

Predefined tables in dbacademy_baw3hg_virginia_edu_dewd_5_1:
  -none-

Setup completed in 4 seconds
```

2.0. Bronze Table: Ingest data

This lab uses a collection of customer-related CSV data from DBFS found in */databricks-datasets/retail-org/customers/*. Read this data using Auto Loader using its schema inference (use `customersCheckpointPath` to store the schema info). Stream the raw data to a Delta table called `bronze`.

```
# TODO:
source_data  = '/databricks-datasets/retail-org/customers/'
customers_to_checkpoint_path = f"{DA.paths.checkpoints}/customers"

query = (spark.readStream
              .format("cloudFiles")
              .option("cloudFiles.format", "csv")
              .option("cloudFiles.schemaLocation", customers_to_checkpoint_path)
              .load("/databricks-datasets/retail-org/customers/")
              .writeStream
              .format("delta")
              .option("checkpointLocation", customers_to_checkpoint_path)
              .outputMode("append")
              .table("bronze"))
```

Stream stopped...

```
DA.block_until_stream_is_ready(query)
```

```
The stream has processed 6 batchs
```

2.1. Create a Streaming Temporary View

Create a streaming temporary view into the bronze table so that we can perform transformations using SQL.

```
(spark
  .readStream
  .table("bronze")
  .createOrReplaceTempView("bronze_temp"))
```

2.2. Clean and Enhance the Data

Use the CTAS syntax to define a new streaming view called `bronze_enhanced_temp` that does the following:

- Skips records with a null `postcode` (set to zero)
- Inserts a column called `receipt_time` containing a current timestamp
- Inserts a column called `source_file` containing the input filename

```
%sql
-- TODO:
CREATE OR REPLACE TEMPORARY VIEW bronze_enhanced_temp AS
SELECT *, current_timestamp() receipt_time, input_file_name() source_file
FROM bronze_temp
WHERE postcode > 0
```

OK

3.0. Silver Table

Stream the data from `bronze_enhanced_temp` to a table called `silver`.

```
# TODO:
silver_checkpoint_path = f"{DA.paths.checkpoints}/silver"

query = (spark.table("bronze_enhanced_temp")
             .writeStream
             .format("delta")
             .option("checkpointLocation", silver_checkpoint_path)
             .outputMode("append")
             .table("silver"))
```

▸ a83daed4-ec02-4d36-8940-872ef8faa7b2 *Last updated: 36 minutes ago*

```
DA.block_until_stream_is_ready(query)
```

```
The stream has processed 10 batchs
```

3.1. Create a Streaming Temporary View

Create another streaming temporary view for the silver table so that we can perform business-level queries using SQL.

```
(spark
  .readStream
  .table("silver")
  .createOrReplaceTempView("silver_temp"))
```

4.0. Gold Table

```
%sql
-- TODO:
CREATE OR REPLACE TEMPORARY VIEW customer_count_by_state_temp AS
SELECT state, count(state) AS customer_count
FROM silver_temp
GROUP BY state
```

OK

Finally, stream the data from the `customer_count_by_state_temp` view to a Delta table called `gold_customer_count_by_state`.

```
# TODO:
customers_count_checkpoint_path = f"{DA.paths.checkpoints}/customers_counts"

query = (spark.table("customer_count_by_state_temp")
         .writeStream
         .format("delta")
         .option("checkpointLocation", customers_count_checkpoint_path)
         .outputMode("complete")
         .table("gold_customer_count_by_state"))
```

▸ d08f6ee5-baf5-4ae2-aeca-f78fcab86d46 *Last updated: 27 minutes ago*

```
DA.block_until_stream_is_ready(query)
```

```
The stream has processed 2 batchs
```

5.0. Query the Results

Query the `gold_customer_count_by_state` table (this will not be a streaming query). Plot the results as a bar graph and also using the map plot.

```
%sql
SELECT * FROM gold_customer_count_by_state
```

Table

	state	customer_count
1	MT	203
2	TX	564
3	NV	40
4	AR	11
5	NH	2
6	WI	938

51 rows

6.0. Clean Up

Run the following cell to remove the database and all data associated with this lab.

```
DA.cleanup()
```

```
Stopping the stream "None"
Stopping the stream "None"
Stopping the stream "None"
```

```
Dropping the database "dbacademy_baw3hg_virginia_edu_dewd_5_1"
Removing the working directory "dbfs:/user/baw3hg@virginia.edu/dbacademy/dewd/5.1"
```

By completing this lab, you should now feel comfortable:

- Using PySpark to configure Auto Loader for incremental data ingestion
- Using Spark SQL to aggregate streaming data
- Streaming data to a Delta table