databricks Bronze Table Creation

(https://databricks.com)

```
databaseName = "group4_finalproject"
print('databaseName: ' + databaseName)

spark.sql(f"CREATE DATABASE IF NOT EXISTS {databaseName}")
spark.sql(f"use {databaseName}")
```

```
databaseName: group4_finalproject
DataFrame[]
```

```
%fs
ls /mnt/data/2023-kaggle-final/energy-prediction/
```

Table

	path	name
1	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/client.csv	client.csv
2	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/county_id_to_name_map.json	county_id_to_name_map.json
3	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/electricity_prices.csv	electricity_prices.csv
4	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/enefit/	enefit/
5	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/example_test_files/	example_test_files/
6	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/forecast_weather.csv	forecast_weather.csv
7	dbfs:/mnt/data/2023-kaggle-final/energy-prediction/gas_prices.csv	gas_prices.csv

11 rows

Train Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/train.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
root
 |-- county: integer (nullable = true)
 |-- is_business: integer (nullable = true)
 |-- product_type: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- is_consumption: integer (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- data_block_id: integer (nullable = true)
 |-- row_id: integer (nullable = true)
 |-- prediction_unit_id: integer (nullable = true)
```

```
source_df.createOrReplaceTempView("temp_table")
merge_column = "row_id"

non_merge_columns = [col for col in source_df.columns if col != f"{merge_column}"]
update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

merge_sql = f"""
MERGE INTO train_bronze AS destination
     USING temp_table AS source
        ON destination.{merge_column} = source.{merge_column}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""


if not spark._jsparkSession.catalog().tableExists("train_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("train_bronze")
else:
    spark.sql(merge_sql)
```

Client Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/client.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
root
 |-- product_type: integer (nullable = true)
 |-- county: integer (nullable = true)
 |-- eic_count: integer (nullable = true)
 |-- installed_capacity: double (nullable = true)
 |-- is_business: integer (nullable = true)
 |-- date: date (nullable = true)
 |-- data_block_id: integer (nullable = true)
```

```
source_df.createOrReplaceTempView("temp_table")
merge_columns = ["product_type", "county", "is_business", "date"]

non_merge_columns = [col for col in source_df.columns if col not in merge_columns]

update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

on_clause = " AND ".join([f"destination.{col} = source.{col}" for col in merge_columns])

merge_sql = f"""
MERGE INTO client_bronze AS destination
     USING temp_table AS source
        ON {on_clause}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""

if not spark._jsparkSession.catalog().tableExists("client_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("client_bronze")
else:
    spark.sql(merge_sql)
```

```
AnalysisException: Cannot resolve destination.date in search condition given columns destination.county, destination.is
_business, destination.product_type, destination.target, destination.is_consumption, destination.datetime, destination.da
ta_block_id, destination.row_id, destination.prediction_unit_id, source.product_type, source.county, source.eic_count, so
urce.installed_capacity, source.is_business, source.date, source.data_block_id.; line 4 pos 142
```

County ID To Name Map Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/county_id_to_name_map.json"
source_df = spark.read.json(file_path)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
bronze_table = "county_id_to_name_map_bronze"
if not spark._jsparkSession.catalog().tableExists(bronze_table):
    source_df.write.format("delta").saveAsTable(bronze_table)
```

Electricity Prices Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/electricity_prices.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
source_df.createOrReplaceTempView("temp_table")
merge_column = "forecast_date"

non_merge_columns = [col for col in source_df.columns if col != f"{merge_column}"]
update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

merge_sql = f"""
MERGE INTO electricity_prices_bronze AS destination
     USING temp_table AS source
        ON destination.{merge_column} = source.{merge_column}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""

if not spark._jsparkSession.catalog().tableExists("electricity_prices_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("electricity_prices_bronze")
else:
    spark.sql(merge_sql)
```

Forecast Weather Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/forecast_weather.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
root
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- origin_datetime: timestamp (nullable = true)
 |-- hours_ahead: integer (nullable = true)
 |-- temperature: double (nullable = true)
 |-- dewpoint: double (nullable = true)
 |-- cloudcover_high: double (nullable = true)
```

```
 |-- cloudcover_low: double (nullable = true)
 |-- cloudcover_mid: double (nullable = true)
 |-- cloudcover_total: double (nullable = true)
 |-- 10_metre_u_wind_component: double (nullable = true)
 |-- 10_metre_v_wind_component: double (nullable = true)
 |-- data_block_id: integer (nullable = true)
 |-- forecast_datetime: timestamp (nullable = true)
 |-- direct_solar_radiation: double (nullable = true)
 |-- surface_solar_radiation_downwards: double (nullable = true)
 |-- snowfall: double (nullable = true)
 |-- total_precipitation: double (nullable = true)
```

```
source_df.createOrReplaceTempView("temp_table")
merge_columns = ["latitude", "longitude", "origin_datetime", "forecast_datetime", "hours_ahead"]

non_merge_columns = [col for col in source_df.columns if col not in merge_columns]

update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

on_clause = " AND ".join([f"destination.{col} = source.{col}" for col in merge_columns])

merge_sql = f"""
MERGE INTO forecast_weather_bronze AS destination
     USING temp_table AS source
        ON {on_clause}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""

if not spark._jsparkSession.catalog().tableExists("forecast_weather_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("forecast_weather_bronze")
else:
    spark.sql(merge_sql)
```

Gas Prices Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/gas_prices.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
root
 |-- forecast_date: date (nullable = true)
 |-- lowest_price_per_mwh: double (nullable = true)
 |-- highest_price_per_mwh: double (nullable = true)
 |-- origin_date: date (nullable = true)
```

```
 |-- data_block_id: integer (nullable = true)
```

```
source_df.createOrReplaceTempView("temp_table")
merge_column = "forecast_date"

non_merge_columns = [col for col in source_df.columns if col != f"{merge_column}"]
update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

merge_sql = f"""
MERGE INTO gas_prices_bronze AS destination
     USING temp_table AS source
        ON destination.{merge_column} = source.{merge_column}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""

if not spark._jsparkSession.catalog().tableExists("gas_prices_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("gas_prices_bronze")
else:
    spark.sql(merge_sql)
```

Historical Weather Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/historical_weather.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```

```
root
 |-- datetime: timestamp (nullable = true)
 |-- temperature: double (nullable = true)
 |-- dewpoint: double (nullable = true)
 |-- rain: double (nullable = true)
 |-- snowfall: double (nullable = true)
 |-- surface_pressure: double (nullable = true)
 |-- cloudcover_total: integer (nullable = true)
 |-- cloudcover_low: integer (nullable = true)
 |-- cloudcover_mid: integer (nullable = true)
 |-- cloudcover_high: integer (nullable = true)
 |-- windspeed_10m: double (nullable = true)
 |-- winddirection_10m: integer (nullable = true)
 |-- shortwave_radiation: double (nullable = true)
 |-- direct_solar_radiation: double (nullable = true)
 |-- diffuse_radiation: double (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
```

```
|-- data_block_id: integer (nullable = true)
```

```
source_df.createOrReplaceTempView("temp_table")
merge_columns = ["latitude", "longitude", "datetime"]

non_merge_columns = [col for col in source_df.columns if col not in merge_columns]

update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])

on_clause = " AND ".join([f"destination.{col} = source.{col}" for col in merge_columns])

merge_sql = f"""
MERGE INTO historical_weather_bronze AS destination
     USING temp_table AS source
        ON {on_clause}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET THEN
           INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE
"""

if not spark._jsparkSession.catalog().tableExists("historical_weather_bronze"):
    source_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("historical_weather_bronze")
else:
    spark.sql(merge_sql)
```

Weather Station to County Mapping Bronze Table

```
file_path = "dbfs:/mnt/data/2023-kaggle-final/energy-prediction/weather_station_to_county_mapping.csv"
source_df = spark.read.csv(file_path, header=True, inferSchema=True)
```

```
display(source_df)
```

```
source_df.printSchema()
```