Azure Assignment-4

Name: Priyeshwar

Mail: priyesh2664@gmail.com

Databases vs. Data Warehouses vs. Data Lakes

1. Database

A database is a structured collection of data designed to support Online Transaction Processing (OLTP) — fast inserts, updates, and queries for operational systems.

Characteristics

- Stores structured and sometimes semi-structured data.
- Supports ACID transactions for data integrity.
- Optimized for small, frequent read/write operations.
- Can be relational (SQL) or non-relational (NoSQL).

Common Uses

- Application backends (e.g., e-commerce carts, banking systems).
- Storing operational records (e.g., customers, orders, inventory).

Examples

- Relational: MySQL, PostgreSQL, Oracle, SQL Server.
- NoSQL: MongoDB, Cassandra, Redis.

---

2. Data Warehouse

A data warehouse is a centralized repository optimized for Online Analytical Processing (OLAP) — querying and analyzing large volumes of structured, historical data from multiple sources.

Characteristics

- Stores structured and some semi-structured data.
- Requires a predefined schema (*schema-on-write*).
- Data loaded via ETL (Extract-Transform-Load) or ELT processes.
- Optimized for analytical queries, aggregations, and reporting.
- Data may be updated periodically (not real-time).

Common Uses

- Business Intelligence (BI) dashboards.
- Historical trend analysis.
- KPI reporting.

Examples

- Snowflake, Amazon Redshift, Google BigQuery, Azure Synapse.

---

3. Data Lake

A data lake is a large-scale storage repository for raw data in its original format, supporting a wide range of analytics and machine learning use cases.

Characteristics

- Stores structured, semi-structured, and unstructured data (e.g., JSON, logs, images, videos).
- No schema required at ingestion (*schema-on-read*).
- Highly scalable and cost-effective.
- Can integrate with multiple processing frameworks (e.g., Spark, Presto, Hive).
- Suitable for both batch and streaming data.

Common Uses

- Data science and machine learning workloads.
- Exploratory analytics.
- Long-term, low-cost data archiving.

Examples

- Storage: AWS S3, Azure Data Lake Storage, Google Cloud Storage.
- Query Engines: Databricks, AWS Athena, Presto.

Comparison Table

Feature	Database	Data Warehouse	Data Lake
Main Workload	Operational (OLTP)	Analytical (OLAP)	Analytical / ML
Data Type	Structured / Semi-structured	Structured / Semi-structured	All types (structured, semi, unstructured)
Schema	Fixed or flexible	Fixed, predefined (*schema-on-write*)	None until read (*schema-on-read*)
Data Freshness	Real-time	Batch updates	Variable (real-time to batch)
Primary Users	Developers	BI analysts, data scientists	Data scientists, engineers
Strengths	Fast transactions	High-performance analytics	Flexible, cheap, scalable

Feature	Database	Data Warehouse	Data Lake
Limitations	Limited analytics	Rigid schema, ETL required	Requires prep before analysis

---

Create a table

Create a Delta table

Merge & Upsert to a table

To merge a set of updates and insertions into an existing Delta table, you use the DeltaTable.merge method for Python and Scala, and the MERGE INTO statement for SQL. For example, the following example takes data from the source table and merges it into the target Delta table. When there is a matching row in both tables, Delta Lake updates the data column using the given expression. When there is no matching row, Delta Lake adds a new row. This operation is known as an upsert.

```python
%python
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DateType
from datetime import date

schema = StructType([
  StructField("id", IntegerType(), True),
  StructField("firstName", StringType(), True),
  StructField("middleName", StringType(), True),
  StructField("lastName", StringType(), True),
  StructField("gender", StringType(), True),
  StructField("birthDate", DateType(), True),
  StructField("ssn", StringType(), True),
  StructField("salary", IntegerType(), True)
])

data = [
  (9999998, 'Billy', 'Tommie', 'Luppitt', 'M', date.fromisoformat('1992-09-17'), '953-38-9452', 55250),
  (9999999, 'Elias', 'Cyril', 'Leadbetter', 'M', date.fromisoformat('1984-05-22'), '906-51-2137', 48500),
  (10000000, 'Joshua', 'Chas', 'Broggio', 'M', date.fromisoformat('1968-07-22'), '988-61-6247', 90000),
  (20000001, 'John', '', 'Doe', 'M', date.fromisoformat('1978-01-14'), '345-67-8901', 55500),
  (20000002, 'Mary', '', 'Smith', 'F', date.fromisoformat('1982-10-29'), '456-78-9012', 98250),
  (20000003, 'Jane', '', 'Doe', 'F', date.fromisoformat('1981-06-25'), '567-89-0123', 89900)
]

people_10m_updates = spark.createDataFrame(data, schema)
people_10m_updates.createOrReplaceTempView("people_10m_updates")

# ...
```

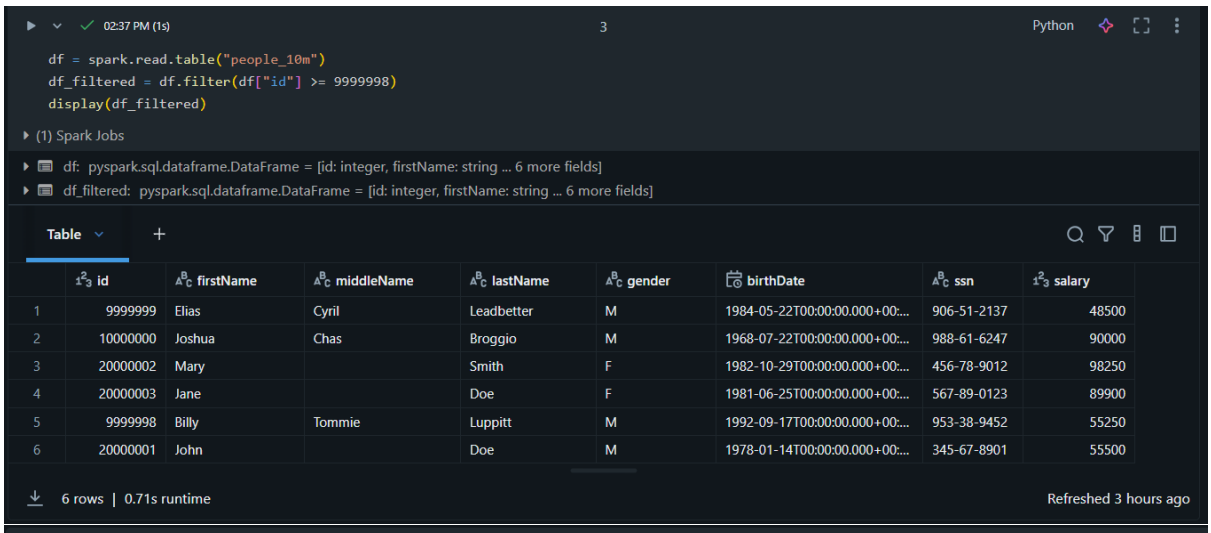
```python
from delta.tables import DeltaTable

deltaTable = DeltaTable.forName(spark, 'people_10m')

(deltaTable.alias("people_10m")
  .merge(
    people_10m_updates.alias("people_10m_updates"),
    "people_10m.id = people_10m_updates.id")
  .whenMatchedUpdateAll()
  .whenNotMatchedInsertAll()
  .execute()
)
```

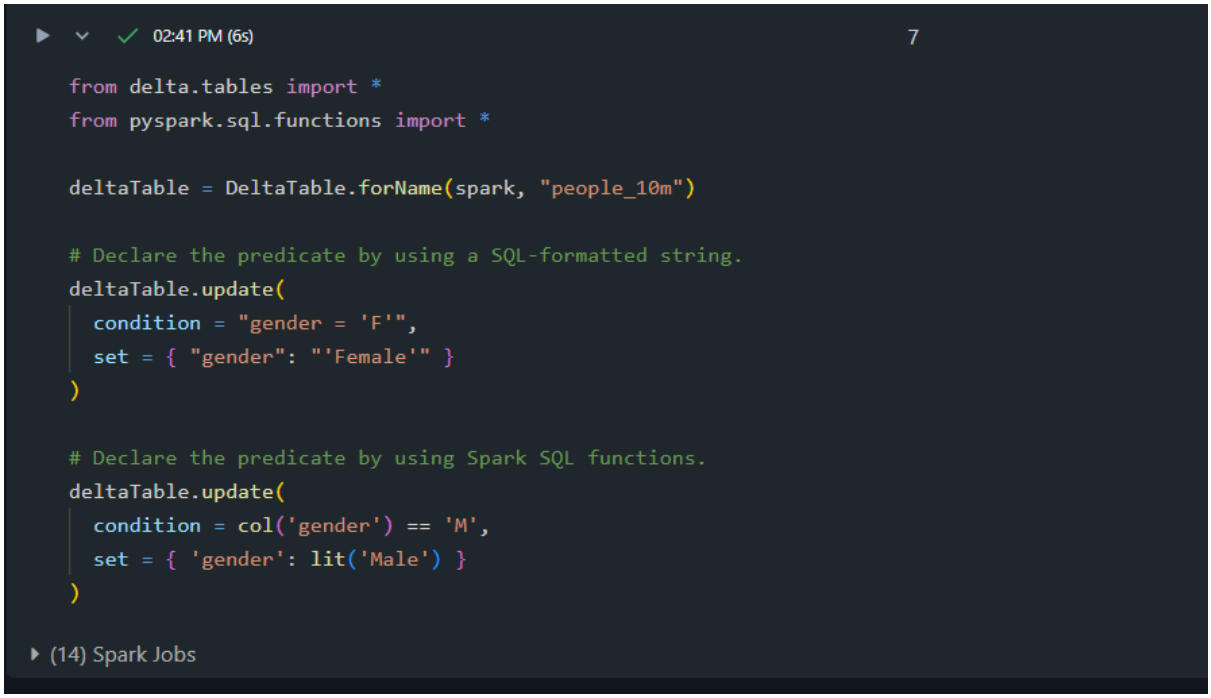

02:37 PM (1s)
3
df = spark.read.table("people_10m")
df_filtered = df.filter(df["id"] >= 9999998)
display(df_filtered)
(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]
df_filtered: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

Read a table

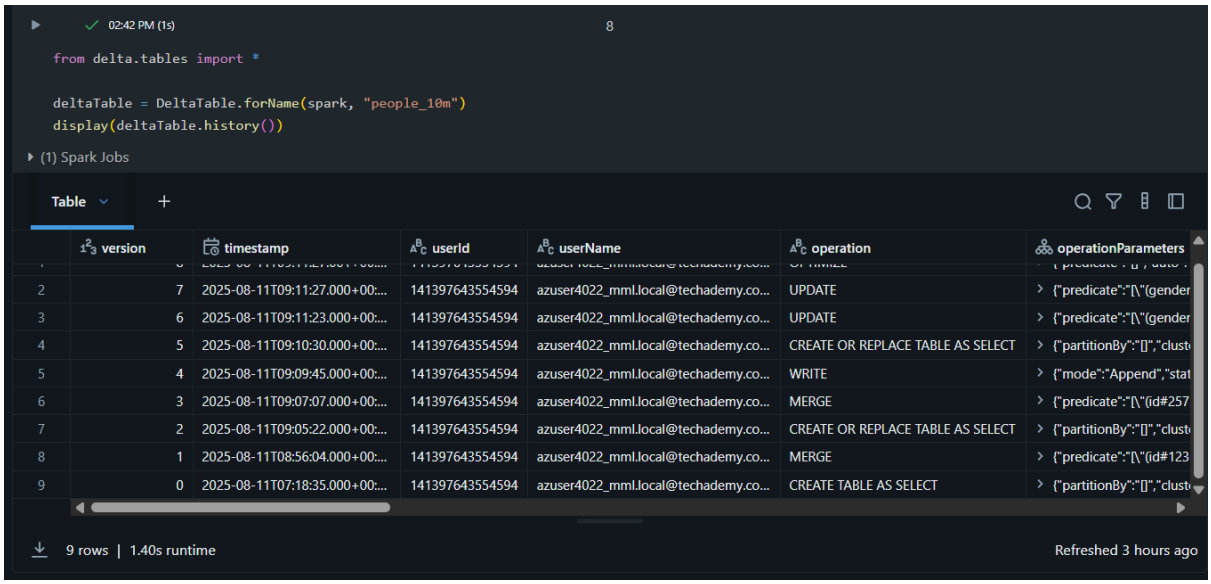

02:37 PM (1s)
3
Python
df = spark.read.table("people_10m")
df_filtered = df.filter(df["id"] >= 9999998)
display(df_filtered)
(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]
df_filtered: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]
Table
id | firstName | middleName | lastName | gender | birthDate | ssn | salary
1 | 9999999 | Elias | Cyril | Leadbetter | M | 1984-05-22T00:00:00.000+00:... | 906-51-2137 | 48500
2 | 10000000 | Joshua | Chas | Broggio | M | 1968-07-22T00:00:00.000+00:... | 988-61-6247 | 90000
3 | 20000002 | Mary | | Smith | F | 1982-10-29T00:00:00.000+00:... | 456-78-9012 | 98250
4 | 20000003 | Jane | | Doe | F | 1981-06-25T00:00:00.000+00:... | 567-89-0123 | 89900
5 | 9999998 | Billy | Tommie | Luppitt | M | 1992-09-17T00:00:00.000+00:... | 953-38-9452 | 55250
6 | 20000001 | John | | Doe | M | 1978-01-14T00:00:00.000+00:... | 345-67-8901 | 55500
6 rows | 0.71s runtime
Refreshed 3 hours ago
02:38 PM (1s)
4
Python
people_df = spark.read.table("people_10m")
display(people_df)
(2) Spark Jobs
people_df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]
Table
id | firstName | middleName | lastName | gender | birthDate | ssn | salary
1 | 1 | Pennie | Carry | Hirschmann | F | 1955-07-02T04:00:00.000+00:... | 981-43-9345 | 56172
2 | 2 | An | Amira | Cowper | F | 1992-02-08T05:00:00.000+00:... | 978-97-8086 | 40203
3 | 3 | Quyen | Marlen | Dome | F | 1970-10-11T04:00:00.000+00:... | 957-57-8246 | 53417
4 | 4 | Coralie | Antonina | Marshal | F | 1990-04-11T04:00:00.000+00:... | 963-39-4885 | 94727
5 | 5 | Terrie | Wava | Bonar | F | 1980-01-16T05:00:00.000+00:... | 964-49-8051 | 79908
6 | 6 | Chassidy | Concepcion | Bourthouloume | F | 1990-11-24T05:00:00.000+00:... | 954-59-9172 | 64652
7 | 7 | Geri | Tambra | Mosby | F | 1970-12-19T05:00:00.000+00:... | 968-16-4020 | 38195
8 | 8 | Patria | Nancy | Arstall | F | 1985-01-02T05:00:00.000+00:... | 984-76-3770 | 102053
9 | 9 | Terese | Alfredia | Tocque | F | 1967-11-17T05:00:00.000+00:... | 967-48-7309 | 91294
10 | 10 | Wava | Lyndsey | Jeandon | F | 1963-12-30T05:00:00.000+00:... | 997-82-2946 | 56521
11 | 11 | Sophie | Emerita | Hearn | F | 1979-09-17T04:00:00.000+00:... | 977-66-4483 | 90920
12 | 12 | Jodie | Tabetha | Laneham | F | 1959-01-31T05:00:00.000+00:... | 923-24-9769 | 90634
13 | 13 | Marietta | Mandi | Yansons | F | 1974-02-19T04:00:00.000+00:... | 900-34-8083 | 93162

Write to a table

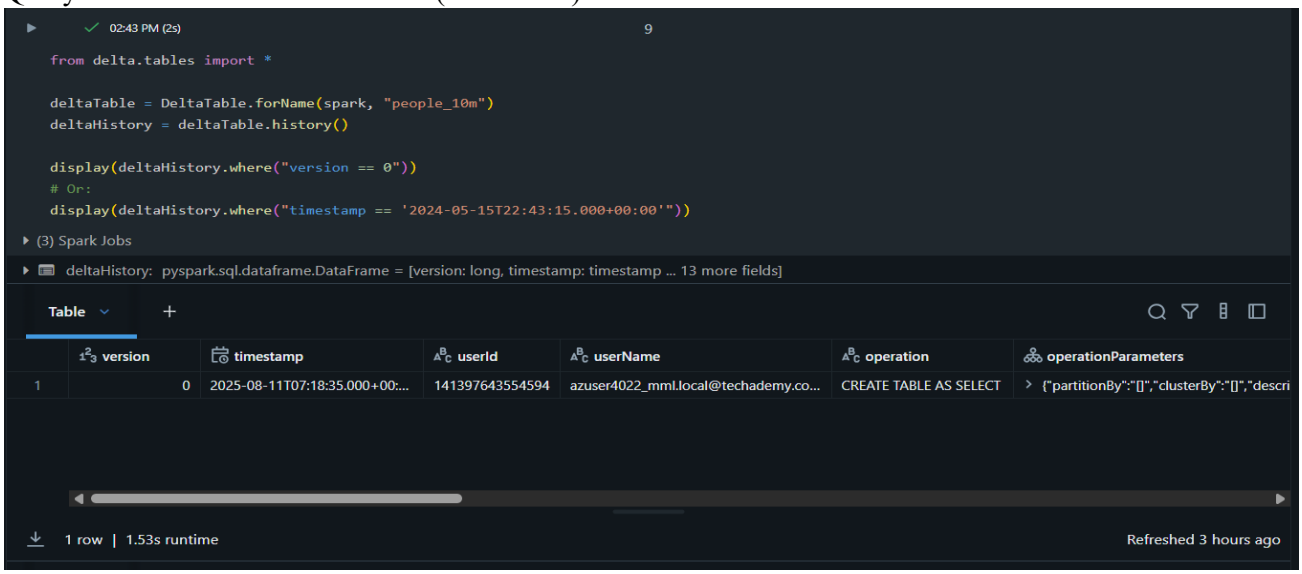
Update a table

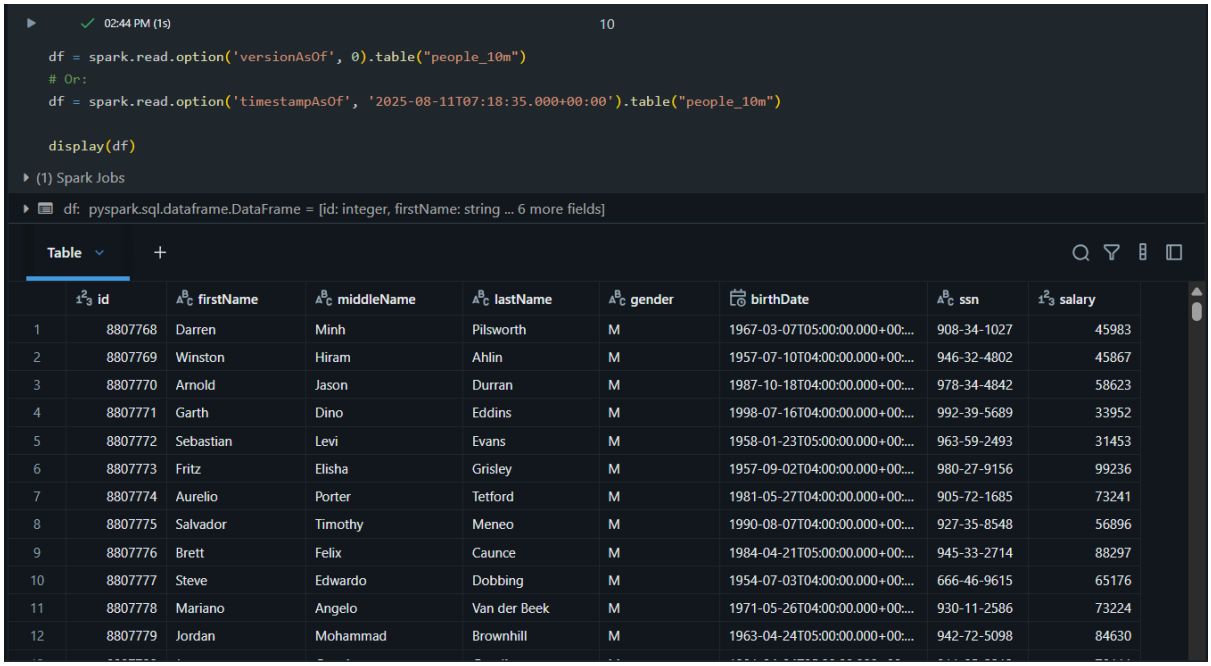


Display table history



Query an earlier version of the table (time travel)





Optimize a table

After you have performed multiple changes to a table, you might have a lot of small files. To improve the speed of read queries, you can use the optimize operation to collapse small files into larger ones:

Z-order by columns

To improve read performance further, you can collocate related information in the same set of files by z-ordering. Delta Lake data-skipping algorithms use this collocation to dramatically reduce the

amount of data that needs to be read. To z-order data, you specify the columns to order on in the z-order by operation. For example, to collocate by gender, run:

Clean up snapshots with VACUUM