
Task 1: Vehicle Maintenance Data Ingestion

- Use the following CSV data representing vehicle maintenance records:

  ```
  VehicleID,Date,ServiceType,ServiceCost,Mileage
  V001,2024-04-01,Oil Change,50.00,15000
  V002,2024-04-05,Tire Replacement,400.00,30000
  V003,2024-04-10,Battery Replacement,120.00,25000
  V004,2024-04-15,Brake Inspection,200.00,40000
  V005,2024-04-20,Oil Change,50.00,18000
  ```

- Ingest this CSV data into a Delta table in Databricks.
- Add error handling for cases where the file is missing or contains incorrect data, and log any such issues.

Task 2: Data Cleaning

- Clean the vehicle maintenance data:
  - Ensure that the `ServiceCost` and `Mileage` columns contain valid positive values.
  - Remove any duplicate records based on `VehicleID` and `Date`.
  - Save the cleaned data to a new Delta table.

Task 3: Vehicle Maintenance Analysis

- Create a notebook to analyze the vehicle maintenance data:
  - Calculate the total maintenance cost for each vehicle.
  - Identify vehicles that have exceeded a certain mileage threshold (e.g., 30,000 miles) and might need additional services.
  - Save the analysis results to a Delta table.

Task 5: Data Governance with Delta Lake

- Enable Delta Lake's data governance features:
  - Use `VACUUM` to clean up old data from the Delta table.
  - Use `DESCRIBE HISTORY` to check the history of updates to the maintenance records.

Task 1: Movie Ratings Data Ingestion

- Use the following CSV data to represent movie ratings by users:

  ```
  UserID,MovieID,Rating,Timestamp
  U001,M001,4,2024-05-01 14:30:00
  U002,M002,5,2024-05-01 16:00:00
  U003,M001,3,2024-05-02 10:15:00
  U001,M003,2,2024-05-02 13:45:00
  U004,M002,4,2024-05-03 18:30:00
  ```

- Ingest this CSV data into a Delta table in Databricks.
- Ensure proper error handling for missing or inconsistent data, and log errors accordingly.

Task 2: Data Cleaning

- Clean the movie ratings data:
    - Ensure that the `Rating` column contains values between 1 and 5.
    - Remove any duplicate entries (same `UserID` and `MovieID`).
    - Save the cleaned data to a new Delta table.

Task 3: Movie Rating Analysis

- Create a notebook to analyze the movie ratings:
    - Calculate the average rating for each movie.
    - Identify the movies with the highest and lowest average ratings.
    - Save the analysis results to a Delta table.

Task 4: Time Travel and Delta Lake History

- Implement Delta Lake's time travel feature:
    - Perform an update to the movie ratings data (e.g., change a few ratings).
    - Roll back to a previous version of the Delta table to retrieve the original ratings.
    - Use `DESCRIBE HISTORY` to view the history of changes to the Delta table.

Task 5: Optimize Delta Table

- Apply optimizations to the Delta table:
    - Implement `Z-ordering` on the `MovieID` column to improve query performance.
    - Use the `OPTIMIZE` command to compact the data and improve performance.
    - Use `VACUUM` to clean up older versions of the table.

Task 1: Data Ingestion - Reading Data from Various Formats

1. **Ingest data from different formats** (CSV, JSON, Parquet, Delta table):
    - **CSV Data**: Use the following CSV data to represent student information:

      ```
      StudentID,Name,Class,Score
      S001,Anil Kumar,10,85
      S002,Neha Sharma,12,92
      S003,Rajesh Gupta,11,78
      ```

    - **JSON Data**: Use the following JSON data to represent city information:

      ```json
      [
        {"CityID": "C001", "CityName": "Mumbai", "Population": 20411000},
        {"CityID": "C002", "CityName": "Delhi", "Population": 16787941},
        {"CityID": "C003", "CityName": "Bangalore", "Population": 8443675}
      ]
      ```

    - **Parquet Data**: Use a dataset containing data about hospitals stored in Parquet format. Write code to load this data into a DataFrame.
    - **Delta Table**: Load a Delta table containing hospital records, ensuring you include proper error handling in case the table does not exist.

Task 2: Writing Data to Various Formats

1. **Write data from the following DataFrames to different formats**:

    - **CSV**: Write the student data (from Task 1) to a CSV file.
    - **JSON**: Write the city data (from Task 1) to a JSON file.
    - **Parquet**: Write the hospital data (from Task 1) to a Parquet file.
    - **Delta Table**: Write the hospital data to a Delta table.

Task 3: Running One Notebook from Another

1. **Create two notebooks**:

    - Notebook A: Ingest data from a CSV file, clean the data (remove duplicates, handle missing values), and save it as a Delta table.
    - Notebook B: Perform analysis on the Delta table created in Notebook A (e.g., calculate the average score of students) and write the results to a new Delta table.

2. **Run Notebook B from Notebook A**:

    - Implement the logic to call and run Notebook B from within Notebook A.

Task 4: Databricks Ingestion

1. **Read data from the following sources**:

    - CSV file from Azure Data Lake.
    - JSON file stored on Databricks FileStore.
    - Parquet file from an external data source (e.g., AWS S3).
    - Delta table stored in a Databricks-managed database.

2. **Write the cleaned data** to each of the formats listed above (CSV, JSON, Parquet, and Delta) after performing some basic transformations (e.g., filtering rows, calculating totals).

Additional Tasks:

- **Optimization Task**: Once the data is written to a Delta table, optimize it using Delta Lake's `OPTIMIZE` command.
- **Z-ordering Task**: Apply Z-ordering on the `CityName` or `Class` columns for faster querying.
- **Vacuum Task**: Use the `VACUUM` command to clean up old versions of the Delta table.

Exercise 1: Creating a Complete ETL Pipeline using Delta Live Tables (DLT)

Objective:

Learn how to create an end-to-end ETL pipeline using Delta Live Tables.

Tasks:

1. **Create Delta Live Table (DLT) Pipeline**:

    - Set up a DLT pipeline for processing transactional data. Use sample data representing daily customer transactions.

```
TransactionID,TransactionDate,CustomerID,Product,Quantity,Price
1,2024-09-01,C001,Laptop,1,1200
2,2024-09-02,C002,Tablet,2,300
3,2024-09-03,C001,Headphones,5,50
4,2024-09-04,C003,Smartphone,1,800
5,2024-09-05,C004,Smartwatch,3,200
```

- Define the pipeline steps:
  - **Step 1**: Ingest raw data from CSV files.
  - **Step 2**: Apply transformations (e.g., calculate total transaction amount).
  - **Step 3**: Write the final data into a Delta table.

2. **Write DLT in Python**:

   - Implement the pipeline using **DLT in Python**. Define the following tables:
     - **Raw Transactions Table**: Read data from the CSV file.
     - **Transformed Transactions Table**: Apply transformations (e.g., calculate total amount: `Quantity * Price`).

3. **Write DLT in SQL**:

   - Implement the same pipeline using **DLT in SQL**. Use SQL syntax to define tables, transformations, and outputs.

4. **Monitor the Pipeline**:

   - Use Databricks' DLT UI to monitor the pipeline and check the status of each step.

Exercise 2: Delta Lake Operations - Read, Write, Update, Delete, Merge

Objective:

Work with Delta Lake to perform read, write, update, delete, and merge operations using both PySpark and SQL.

Tasks:

1. **Read Data from Delta Lake**:

   - Read the transactional data from the Delta table you created in the first exercise using PySpark and SQL.
   - Verify the contents of the table by displaying the first 5 rows.

2. **Write Data to Delta Lake**:

   - Append new transactions to the Delta table using PySpark.
   - Example new transactions:

     ```
     6,2024-09-06,C005,Keyboard,4,100
     7,2024-09-07,C006,Mouse,10,20
     ```

3. **Update Data in Delta Lake**:

   - Update the `Price` of `Product = 'Laptop'` to `1300`.

- Use PySpark or SQL to perform the update and verify the results.

4. **Delete Data from Delta Lake**:

   - Delete all transactions where the `Quantity` is less than 3.
   - Use both PySpark and SQL to perform this deletion.

5. **Merge Data into Delta Lake**:

   - Create a new set of data representing updates to the existing transactions. Merge the following new data into the Delta table:

     ```
     TransactionID,TransactionDate,CustomerID,Product,Quantity,Price
     1,2024-09-01,C001,Laptop,1,1250  -- Updated Price
     8,2024-09-08,C007,Charger,2,30   -- New Transaction
     ```

   - Use the Delta Lake **merge** operation to insert the new data and update the existing records.

Exercise 3: Delta Lake - History, Time Travel, and Vacuum

Objective:

Understand how to use Delta Lake features such as versioning, time travel, and data cleanup with vacuum.

Tasks:

1. **View Delta Table History**:

   - Query the **history** of the Delta table to see all changes (inserts, updates, deletes) made in the previous exercises.
   - Use both PySpark and SQL to view the history.

2. **Perform Time Travel**:

   - Retrieve the state of the Delta table as it was **5 versions ago**.
   - Verify that the table reflects the data before some of the updates and deletions made earlier.
   - Perform a query to get the transactions from a specific timestamp (e.g., just before an update).

3. **Vacuum the Delta Table**:

   - Clean up old data using the **VACUUM** command.
   - Set a retention period of 7 days and vacuum the Delta table.
   - Verify that old versions are removed, but the current table state is intact.

4. **Converting Parquet Files to Delta Files**:

   - Create a new Parquet-based table from the raw transactions CSV file.
   - Convert this Parquet table to a Delta table using Delta Lake functionality.

Exercise 4: Implementing Incremental Load Pattern using Delta Lake

Objective:

Learn how to implement incremental data loading with Delta Lake to avoid reprocessing old data.

Tasks:

1. **Set Up Initial Data**:
   - Use the same transactions data from previous exercises, but load only transactions from the first three days (`2024-09-01` to `2024-09-03`) into the Delta table.
2. **Set Up Incremental Data**:
   - Add a new set of transactions representing the next four days (`2024-09-04` to `2024-09-07`).
   - Ensure that these transactions are loaded incrementally into the Delta table.
3. **Implement Incremental Load**:
   - Create a pipeline that reads new transactions only (transactions after `2024-09-03`) and appends them to the Delta table without overwriting existing data.
   - Verify that the incremental load only processes new data and does not duplicate or overwrite existing records.
4. **Monitor Incremental Load**:
   - Check the Delta Lake version history to ensure only the new transactions are added, and no old records are reprocessed.
