DS 206: GROUP PROJECT #2

SUBMISSION GUIDELINES:

- One member should create a GitHub repository named DS206_Spring2025_Group# (# is your group number).
- You should push all your code to a new GitHub repository where each group member **should contribute with at least 1 pull request.** You should share the repository with me (https://github.com/Arman-Asryan).
- You should then download the repository as **a zip file** and submit it on Moodle (one submission per group).
- The submission deadline for submitting the project: **23:59, May 17, 2025.**

PART 0. PROJECT INITIATION AND STRUCTURE (0 POINTS):

Create a project directory according to the following minimal structure:

- infrastructure_initiation
  - dimensional_db_creation.sql
  - dimenional_db_table_creation.sql
  - staging_raw_table_creation.sql
- pipeline_dimensional_data
  - flow.py (class)
  - tasks.py (flow-specific functions for executing sql scripts)
  - config.py (flow-specific dimensional table names, database name)
  - queries
    - SQL scripts (parametrized) to update the dimension and fact tables
- logs
  - logs_dimensional_data_pipeline.txt
- dashboard
  - logs_dimensional_data_pipeline.txt
- main.py
- utils.py
- logging.py

DIMENSIONAL DATA STORE (DDS) CREATION AND POPULATION (60)

1. Create a file named **dimensional_database_creation.sql** in the **infrastructure_initiation** folder. Add a DDL query to create an empty database named **ORDER_DDS** and run it (within SQL Server).
2. Store the database connection configuration in the **sql_server_config.cfg** file (feel free to name the config file differently).
3. Create a static (non-parametrized) SQL script named **staging_raw_table_creation.sql** (in the **infrastructure_initiation** folder) containing **CREATE TABLE** statements for each source table mentioned in Table 1 below (for your group). Make sure to include a **staging_raw_id_sk** column (IDENTITY) for each staging raw table.
4. Use the information stored in Table 1 and Table 2 below to derive the proper structure of the dimensional database.
5. Create a static (non-parametrized) SQL script named **dimensional_db_table_creation.sql** (in the **infrastructure_initiation** folder) containing **CREATE TABLE** statements for each dimension/fact table mentioned in Table 3 below (for your group). Make sure to declare the proper surrogate keys for each dimension table (check the lectures and the answer key of Midterm2).
6. Make sure to include a dimension called **Dim_SOR** that has two columns covering the key (SOR_SK) and the staging raw table names. Run the queries within SQL Server.
7. Create parametrized SQL scripts named **update_dim_{table}.sql** to ingest data into the dimension tables from staging tables. Store the queries in the **pipeline_dimensional_data/queries/** folder.
   - Make sure to include the surrogate key (SOR_SK) of each staging raw table (from **Dim_SOR**) along with its staging_raw_id from within each dimension table.
8. Create a parametrized SQL script named **update_fact.sql** to ingest data into the fact table from staging raw and dimension tables. Keep in mind that some fact tables require a MERGE- based ingestion while others need an INSERT-based one. Store the query in the **pipeline_dimensional_data/queries/** folder.
   - This script should contain additional parameters **start_date** and **end_date** (in addition to the remaining common parameters, i.e., database name, schema name, table name) that controls the period of ingestion from the respective staging table into the fact table.
   - Make sure to include the surrogate key of each staging raw table (from **Dim_SOR**) along with its staging_raw_id in each dimension table.
9. Create a parametrized SQL script named **update_fact_error.sql** to ingest faulty rows into the fact_error table from staging raw and dimension tables. Keep in mind that this table should contain rows that haven’t gotten into the fact table due to missing/invalid natural keys.
   - This script should contain additional parameters **start_date** and **end_date** (in addition to the remaining common parameters, i.e., database name, schema name, table name)

that controls the period of ingestion from the respective staging table into the fact table.

- Make sure to include the surrogate key of each staging raw table (from **Dim_SOR**) along with its staging_raw_id in each dimension table.

10. Update the Python file named **utils.py** and put all the new flow-agnostic utility functions (i.e., a function for reading an SQL script from an .sql file and executing it).

11. Create Python functions (necessary for creating the tables for the dimensional database and ingesting data into them) and place them in the **tasks.py** file (in **pipeline_dimensional_data** folder).

- Ensure the atomicity and the reproducibility of Your Python scripts. One python task could be responsible for ingesting data into multiple destination tables.
- These Python scripts (tasks) will be responsible for passing various parameters to the parametrized SQL scripts (database name, schema name, destination and source table names, start and end dates).
- These Python tasks will be responsible for the sequential logic of the pipeline. Each task should return something like {'success': True}, and the task afterwards should run only if its prerequisites have completed successfully (you might need to add arguments to your Python tasks to achieve this).

12. Create a class named **DimensionalDataFlow** in **pipeline_dimensional_data/flow.py** to import and sequentially execute all the tasks from **pipeline_dimensional_data/tasks.py.** Make sure the class has an **exec** method (with arguments **start_date** and **end_date**) for sequential executions of tasks.

- The class should generate a unique uuid() (attribute name: execution_id) upon object creation. In other words, we should use a function from **utils.py** to add an attribute named execution_id in the _ _init_ _ constructor of the class. This uuid will be used for tracking/monitoring.

13. Set up a logger for the dimensional data flow (**logging.py**). The logs should be nicely formatted and should include the execution_id (aka uuid) for each instance. Logs from the dimensional data flow should be inserted into the **logs/logs_dimensional_data_pipeline.txt** file.

14. The pipeline must be executable from the command line using `python main.py --start_date=YYYY-MM-DD --end_date=YYYY-MM-DD`. The script should parse the start_date and end_date arguments, instantiate the class defined in flow.py, and call its exec() method with the parsed dates as parameters.

Table 1. Primary Key Constraints

Schema_name	**Table_name**	**Columns**
dbo	Categories	CategoryID
dbo	Customers	CustomerID
dbo	Employees	EmployeeID
dbo	Order Details	OrderID, ProductID

dbo	Orders	OrderID
dbo	Products	ProductID
dbo	Region	RegionID
dbo	Shippers	ShipperID
dbo	Suppliers	SupplierID
dbo	Territories	TerritoryID

Table 2. Foreign Key Constraints

FK_Table	**FK_Column**	**PK_Table**	**PK_Column**
Employees	ReportsTo	Employees	EmployeeID
Order Details	OrderID	Orders	OrderID
Order Details	ProductID	Products	ProductID
Orders	CustomerID	Customers	CustomerID
Orders	EmployeeID	Employees	EmployeeID
Orders	ShipVia	Shippers	ShipperID
Orders	TerritoryID	Territories	TerritoryID
Products	CategoryID	Categories	CategoryID
Products	SupplierID	Suppliers	SupplierID
Territories	RegionID	Region	RegionID

PART 3. DASHBOARD CREATION USING POWER BI (2-3% OF FINAL GRADE)

- Your solution should be based on the DDS from Part 2 (DirectQuery or Live Connection).
- Your solution should be **at least 2 pages** long (please refer to Power BI pages).
- Your solution should contain
  1. **at least 3-4 topic-consistent visualizations** on each page,
  2. **at least 2 slicers** on each page.
- Your solution should have **at least 5 distinct DAX measures**:
  1. at least 8 Date Intelligence measures
  2. at least 5 CALCULATE() + FILTER() combination,
- Your solution should have **at least 2 tabular**, at least **2 categorical**, and **at least 1 trend** (time series) visualizations.
- Your dashboard must contain **at least 1 tooltip** and **1 drill through.**
- Each page on Your solution should have a separate **Reset All Slicers button**.
- The information provided on each page should be consistent and holistic.

Table 3. Dimensional database variations

GROUP ID	DIMENIONS	COMMENT
GROUP 1 GROUP 3	DimCategories	SCD1 with delete
	DimCustomers	SCD2
	DimEmployees	SCD1 with delete
	DimProducts	SCD4
	DimRegion	SCD1
	DimShippers	SCD3 (2 attributes, current and prior)
	DimSuppliers	SCD4
	DimTerritories	SCD3
	FactOrders	SNAPSHOT
GROUP 2 GROUP 4	DimCategories	SCD1
	DimCustomers	SCD2
	DimEmployees	SCD1 with delete
	DimProducts	SCD1
	DimRegion	SCD4
	DimShippers	SCD1 with delete
	DimSuppliers	SCD3 (one attribute, current and prior)
	DimTerritories	SCD4
	FactOrders	INSERT
GROUP 5	DimCategories	SCD1
	DimCustomers	SCD2
	DimEmployees	SCD1 with delete
	DimProducts	SCD1
	DimRegion	SCD4
	DimShippers	SCD1 with delete
	DimSuppliers	SCD3 (one attribute, current and prior)
	DimTerritories	SCD4
	FactOrders	INSERT