DS 206: GROUP PROJECT #2

PART 0. PROJECT INITIATION (0 POINTS)

Create a project directory according to the following minimal structure:

📁 infrastructure_initiation

   📄 relational_db_creation.sql

   📄 relational_db_table_creation.sql

   📄 dimensional_db_creation.sql

   📄 relational_db_table_creation.sql

   📄 relational_db_add_PK_FK_constraints.sql

📁 pipeline_relational_data

   🐍 flow.py (class)

   🐍 tasks.py (flow-specific functions for executing sql scripts)

   🐍 config.py (flow-specific relational table names, database name)

   📁 queries

      📄 SQL scripts (parametrized) to insert data into the relational database

📁 Pipeline_dimensional_data

   🐍 flow.py (class)

   🐍 tasks.py (flow-specific functions for executing sql scripts)

   🐍 config.py (flow-specific relational table names, database name)

   📁 queries

      📄 SQL scripts (parametrized) to update the dimension and fact tables

📁 logs

   📄 logs_relational_data_pipeline.txt

   📄 logs_dimensional_data_pipeline.txt

🐍 main.py

🐍 utils.py

🐍 logging.py

PART 1. RELATIONAL DATA STORE (RDS) CREATION AND POPULATION (40)

1. Create a file named **relational_database_creation.sql** in the **infrastructure_initiation** folder. Add a DDL query to create an empty database named **ORDERS_RELATIONAL_DB** and run it within SQL Server.

2. Store the database connection configurations in the **sql_server_config.cfg** file (feel free to name the config file differently).

3. Create a static (non-parametrized) SQL script named **relational_db_table_creation.sql** (in the **infrastructure_initiation** folder) containing multiple **CREATE TABLE** statements for all relational database tables (sheets in the **raw_data_source.xlsx** data file). DO NOT add any PKs or FKs at this stage. Run the queries within SQL Server.

4. Create a static (non-parametrized) SQL script named **relational_db_add_PK_FK_constraints.sql** in the **infrastructure_initiation** folder containing multiple **ALTER TABLE** statements to establish the primary and foreign key constraints based on Table 1 and Table 2 respectively.

5. Create parametrized SQL scripts named **insert_into_{table}.sql** to ingest data into the empty tables from **raw_data_source.xlsx** in a correct order. Store the queries in the **pipeline_relational_data/queries/** folder.

6. Update the Python file named **utils.py** and put all the flow-agnostic utility functions (i.e., a function for reading an SQL script from an .sql file and executing it). Keep in mind that utility functions are helpers that might be used for both flows.

7. Create Python functions (necessary for creating the tables for the relational database and ingesting data into them) and place them in the **tasks.py** file (in **pipeline_relational_data** folder). Ensure the atomicity and the reproducibility of Your Python scripts.

8. Create a class named **RelationalDataFlow** in **pipeline_relational_data/flow.py** to import and sequentially execute all the tasks from **pipeline_relational_data/tasks.py.** Make sure that the class has an **exec** method (with no arguments) for executing tasks in a right order.

    1. The class should generate a unique uuid() (attribute name: execution_id) upon object creation. It should use an imported 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.

9. Set up a logger for the relational data flow (**logging.py**). The logs should be nicely formatted and must include a) the execution_id (aka uuid) for each instance, b) the timestamp of each log. Logs from the relational data flow should be inserted into the **logs/logs_relational_data_pipeline.txt** file.

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

10. Create a file named **dimensional_database_creation.sql** in the **infrastructure_initiation** folder for the   Add a DDL query to create an empty database named **ORDERS_DIMENSIONAL_DB** and run it (within SQL Server).

11. Store the database connection configurations in the **sql_server_config.cfg** file (feel free to name the config file differently).

12. 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 SK_PK columns (IDENTITY) as well. Run the queries within SQL Server.

13. Create parametrized SQL scripts named **update_dim_{table}.sql** to ingest data into the dimension tables from relational database tables. Store the queries in the **pipeline_dimensional_data/queries/** folder.

14. Create a parametrized SQL script named **update_fact_{table}.sql** to ingest data into the fact table from relational database tables (in addition to the SK_PK columns picked up from various 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.

15. 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).

16. Create Python functions (necessary for creating the tables for the relational 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.

17. 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 (no arguments) for sequential executions of tasks.
    1. 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.

18. 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 relational data flow should be inserted into the **logs/logs_dimensional_data_pipeline.txt** file.

19. You should be able to import both classes (from **flow.py** files) into **main.py**, create instances and use the **exec()** method of each class to create an execution. You can check the results in SQL Server (tables created, rows inserted/updated) and in the logging txt files.

PART 3. DASHBOARD CREATION USING POWER BI (3-4% OF FINAL GRADE)

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

YOUR SUBMISSION SHOULD INCLUDE THE FOLLOWING COMPONENTS:

- **a zip file** containing all the files mentioned in the **PROJECT INITIATION** part above,
- a PBIX file containing the dashboard created using the data warehouse (**extra-credit**).

Table 3. Dimensional database variations

GROUP ID	DIMENIONS	
GROUP 1 **GROUP 3**	DimCategories	SCD1 with delete
	DimCustomers	SCD2
	DimEmployees	SCD4 with delete
	DimProducts	SCD4
	DimRegion	SCD3 (2 attributes, current and prior)
	DimShippers	SCD1
	DimSuppliers	SCD4
	DimTerritories	SCD3
	FactOrders	SNAPSHOT
GROUP 2 **GROUP 4** **GROUP 8**	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 **GROUP 6**	DimCategories	SCD1 with delete
	DimCustomers	SCD4
	DimEmployees	SCD1
	DimProducts	SCD4
	DimRegion	SCD2
	DimShippers	SCD3 (one attribute, current and prior)
	DimSuppliers	SCD3 (one attribute, current and prior)
	DimTerritories	SCD2
	FactOrders	INSERT
GROUP 7	DimCategories	SCD1 with delete
	DimCustomers	SCD3 (one attribute, current and prior)
	DimEmployees	SCD2
	DimProducts	SCD1 with delete
	DimRegion	SCD1
	DimShippers	SCD1
	DimSuppliers	SCD4
	DimTerritories	SCD2
	FactOrders	INSERT