**Data Engineer Assignment**

You have been tasked with creating a **daily (or on-demand) data generation pipeline** that will:

Once these files are generated, they will land in a **landing zone** for further ingestion into a **Medallion Architecture pipeline**: **Bronze** → **Silver** → **Gold** layers. The pipeline will handle cleansing, transformations, validation, and business logic for final analytics/reporting.

1. **Assumptions & Requirements**

1. **Environment**

* You have Python 3.x installed.
* You can install additional dependencies (if required).
* You can run the script on a schedule or on-demand.

2. **Data Volume**

* By default, each run creates **500 rows** per table (customers, products, orders, order\_items), but this is **configurable** in a JSON file.

3. **File Locations**

* The location for storing **CSV files** and the **ID tracker** can vary across environments (e.g., local folder, shared storage, cloud).
* A **JSON config file** (config.json) will specify these locations.

4. **Unique IDs**

* We want **strictly ascending IDs** for each entity (customers, products, orders, order items).
* A local **ID tracker file** (JSON) keeps track of the **last used ID** for each entity, updated after every run.

5. **File Naming Convention**

* Each CSV file is prefixed with raw\_<table>\_ followed by a **timestamp** in the format YYYYMMDD\_HHMMSS.
* For example: raw\_orders\_20250124\_145030.csv.

6. **Date Handling**

* **Default**: order\_date for newly generated orders is **today’s date**.
* **Override**: You can pass a date (YYYY-MM-DD) as a command-line argument to generate orders with a specific date (useful for backfills or re-runs).

1. **Deliverables**

1. config.json

* + A **JSON config** file that specifies:
  + paths.output\_folder: where CSV files should be generated.
  + paths.tracker\_file: the path to the ID tracker JSON.
  + data\_settings.num\_rows\_per\_table: how many rows to generate per table.

2. generate\_data\_with\_config.py

* + A **Python script** that reads config.json, references the ID tracker, generates the data, and writes CSV outputs.
  + This script should be well-documented so a new developer can easily follow.

3. id\_tracker.json (autogenerated/maintained)

* A **JSON file** storing the last used IDs for each table, automatically updated after each run.
* Looks something like:
* {
* "customers": 11000,
* "products": 11000,
* "orders": 11000,
* "order\_items": 11000
* }

4. **Landing Zone** (Not included in script, but conceptually needed)

* + A folder/bucket where the generated CSV files are dropped.
  + This is the starting point for your Medallion Architecture pipeline.

5. **Medallion Architecture**:

• **Bronze**: Ingest the generated CSVs as-is with minimal changes (e.g., add ingest\_timestamp).

• **Silver**: Cleanse, ensure referential integrity, handle duplicates, quarantine invalid rows.

• **Gold**: Create curated fact/dimension tables or aggregated analytics tables.

6. **Documentation**

* Instructions on how to run the script, how to configure, how to integrate with your pipeline.
* This final set of docs includes assumptions, usage examples, and references to data validation checks.

**3. Detailed End-to-End Steps**

**3.1 Obtain the Files**

• config.json (user-provided or start with the example below).

• generate\_data\_with\_config.py (the main script).

Place both in the same directory (or specify absolute paths carefully).

**3.2 Configure Your Environment**

1. Make sure Python 3 is installed.

2. Optionally create a virtual environment:

python -m venv venv

source venv/bin/activate # Linux/Mac

# or venv\Scripts\activate.bat for Windows

3. (Optional) Install any dependencies if needed (most standard library usage should not require extra installations except perhaps if you want advanced features).

**3.3 Create/Review config.json**

Below is an example:

{

"paths": {

"output\_folder": "/tmp/daily\_data",

"tracker\_file": "/tmp/id\_tracker.json"

},

"data\_settings": {

"num\_rows\_per\_table": 500

}

}

• output\_folder: The folder where all CSV files will be written. The script will create this folder if it doesn’t exist.

• tracker\_file: The JSON file that tracks the last used IDs. If it doesn’t exist, it starts at 10,000 for each table.

• num\_rows\_per\_table: How many rows to generate for each table every run.

**3.4 Check or Initialize id\_tracker.json**

• You do **not** need to create this manually. If tracker\_file doesn’t exist, the script starts from defaults.

• If you **already** have it from a previous run, it will resume from the **last** known IDs.

**3.5 Run the Script**

python generate\_data\_with\_config.py

• **Default** run: Uses the date of **today** for orders.order\_date.

• **Output**: A set of CSV files in /tmp/daily\_data named, for example:

raw\_customers\_20250124\_145030.csv

raw\_products\_20250124\_145030.csv

raw\_orders\_20250124\_145030.csv

raw\_order\_items\_20250124\_145030.csv

• The **ID tracker** in /tmp/id\_tracker.json is updated accordingly.

**Optional: Override the Date**

python generate\_data\_with\_config.py 2025-01-30

* + - This will create the same files, but order\_date in the orders table will be 2025-01-30.
    - File names still include the **current** time, so you never overwrite previous runs.

**4. The Code:** Below is the **complete, commented script** (generate\_data\_with\_config.py) along with an example config.json.

**4.1 config.json**

{

"paths": {

"output\_folder": "/tmp/daily\_data",

"tracker\_file": "/tmp/id\_tracker.json"

},

"data\_settings": {

"num\_rows\_per\_table": 500

}

}

Adjust the paths to your environment. For Windows, you might use "C:/temp" or similar.

**4.2 generate\_data\_with\_config.py:**

* A separate python file is given along with this document.

**5. Medallion Architecture Integration:** Once the CSV files are generated in your output\_folder (landing zone), your data pipeline will:

1. **Bronze Layer**

* + - Read the CSV **as-is** into a data lake or Delta tables.
    - Append minimal metadata (e.g., ingest\_timestamp, source\_system).
    - Perform **basic** schema validation to ensure the expected columns exist.

2. **Silver Layer**

* + - Clean and transform the data:
    - **Deduplicate** if somehow files were loaded multiple times.
    - **Enforce** referential integrity (e.g., order\_items.order\_id must exist in orders, orders.customer\_id must exist in customers, etc.).
    - **Quarantine** or fix invalid rows (e.g., negative quantity).
    - Convert or standardize data types (dates, decimals, string casing).

3. **Gold Layer**

* + - Create **joined** or **aggregated** tables for analytics:
      * E.g., a fact\_sales table that combines orders, order\_items, and products.
    - Possibly join with customers for demographics.
    - Apply **business rules** (e.g., exclude CANCELLED orders from revenue).

**6. Data Validation Checks (Suggested)**

1. **Schema Integrity**:

• Confirm each CSV file has the correct column count and headers.

2. **Non-Null & Positive**:

• customer\_id, product\_id, order\_id, order\_item\_id should not be null.

• quantity and price should be positive.

3. **Referential Integrity**:

• order\_items.order\_id must appear in orders.

• orders.customer\_id must appear in customers.

• order\_items.product\_id must appear in products.

4. **Date Validation**:

• Ensure order\_date is within an acceptable range (based on business rules).

5. **Duplicate Handling**:

• Because IDs are unique, duplicates are unlikely if the ID tracker is never reset.

• If it is reset or corrupted, you may get collisions—Silver layer can check for ID collisions.

6. **Sum Checks** (Optional Business Logic):

• If orders.total\_amount is used, confirm it approximates the sum of its order items (this can vary if partial or cancelled, but still a good check).

**7. Scheduling & Next Steps**

1. **Cron or Task Scheduler**:

* + - You can schedule generate\_data\_with\_config.py to run daily or multiple times per day.

2. **Airflow Integration**:

* + - If you use Airflow, you can define a DAG that runs this script, then triggers the subsequent Bronze → Silver → Gold steps.

3. **Logging & Monitoring**:

* + - For production, log or capture the script’s output to track how many rows were generated each run, any errors, etc.

**Final Summary**

• **Deliverables**:

1. config.json: Contains your dynamic settings.

2. generate\_data\_with\_config.py: Main script to generate data.

3. id\_tracker.json: Auto-updated ID tracker to ensure unique IDs.

4. **CSV files** in the landing zone.

• **Key Points**:

* The script **never** overwrites existing CSVs thanks to **timestamped filenames**.
* **Globally unique IDs** are maintained via the **ID tracker**.
* You can run multiple times a day (with different or same date) to build up incremental data.
* **Medallion Flow**: Bronze (raw, minimal transformation) → Silver (cleaned, integrated) → Gold (curated, business aggregates).