## Data Engineer Assignment Solution Walkthrough PsychoBunny

## Abhijeet Bhattachaya

For a detailed walkthrough of this project, it is highly recommended to read the documentation from the GitHub repo (<https://github.com/abhi0618/PsychoBunny_DataEngineering_Assignment>). Keep the Python notebook scripts open alongside while reviewing the doc to follow along with the working examples and outputs.  
  
Part 1: Data Engineering; Architecture and Fundamentals.

<https://github.com/abhi0618/PsychoBunny_DataEngineering_Assignment>

### **1. High-Level Problem Overview**

We are tasked with ingesting multiple CSV files from an S3 bucket into a Snowflake data warehouse. The CSVs are of three types:

1. **de\_shop\_customers** → customer master data
2. **de\_shop\_transactions** → transactional data
3. **de\_dates** → date reference data

Challenges observed in the sample data:

* Inconsistent column names across files for the same entity (**PROVINCE** vs **STATE**, **ZIP** vs **POSTAL**).
* Missing values in certain columns.
* Duplicate records across files.
* Unknown final size of datasets.
* Requirement for idempotent ingestion (re-runs should not duplicate data).
* Need for secure access and automated, systematic ingestion.

## 

### **2. Proposed Data Architecture**

#### **2.1 Storage Layer**

* **Source**: Amazon S3 bucket (**psychobunnytechassignment**)  
  + Files are prefixed by type (**de\_shop\_customers**, **de\_shop\_transactions**, **de\_dates**)
  + Access via **IAM role**, avoiding hardcoded keys for security.
* **Considerations for large data**: Partition S3 by date (year/month) to enable incremental processing and parallel ingestion in the future.

#### **2.2 Compute & Processing Layer**

* **Pandas** (for current solution):  
  + Chosen for simplicity and ease of column normalization and cleaning.
  + Handles small to medium datasets efficiently.
* **Alternative for larger datasets**:  
  + **Apache Spark** or **Snowpark** could be used to handle high-volume, distributed processing.
  + Can read files in parallel and push transformations down to Snowflake for scalability.

#### **2.3 Data Warehouse Layer**

* **Snowflake**:  
  + Tables: **CUSTOMERS**, **TRANSACTIONS**, **DATES**
  + Columns standardized and normalized for consistent schema.
  + Insertions designed to be **idempotent** by:  
    - Dropping duplicates in Pandas before insertion.
    - Soft deduplication using Snowflake merge or except queries for production-scale systems.
  + Snowflake’s **secure roles and warehouses** provide scalability and access control.

## **Architecture Diagram**

### 

### **3. Data Ingestion Design**

#### **3.1 File Discovery**

* Use **S3 prefix filtering** to select only files of interest.
* Avoid hardcoding file names; dynamically discover all matching CSVs.

#### **3.2 Data Cleaning and Normalization**

* **Column normalization**: Convert all column names to uppercase, strip quotes and whitespace.
* **Variant mapping**: Map alternate column names to a standard schema (**PROVINCE** → **STATE**, **ZIP** → **POSTAL**).
* **Missing columns**: Fill missing columns with **NULL** to ensure schema consistency.
* **Value cleaning**: Strip whitespace, convert "NULL" strings and NaN to **None**.

#### **3.3 Deduplication and Idempotency**

* Deduplicate **at the row level** in Pandas using **drop\_duplicates()** before insertion.
* Optional for production: Check against existing Snowflake rows to avoid re-inserting already ingested records.

#### **3.4 Insertion into Snowflake**

* Inserted **directly from Pandas** into Snowflake.  
  1. Avoided temp tables to simplify the pipeline.
  2. Columns quoted to prevent identifier issues **("FIRST\_NAME").**
* Alternative approaches:  
  1. **Temporary tables**: Useful if:  
     + Preprocessing is heavy.
     + Multiple staging transformations needed.
     + Large datasets require batch operations before merging.
  2. **Merge statements**: For production, use **MERGE INTO** for deduplication at Snowflake side.

## **Ingestion Workflow Diagram (Conceptual)**

[S3 Bucket: de\_shop\_\* CSVs] ---> [Pandas Data Cleaning & Normalization]

| - Column standardization

| - Missing value handling

| - Deduplication (soft check)

V

[Snowflake Staging / Direct Load]

| - Insert normalized data

| - Merge / deduplicate

V

[Snowflake Final Tables]

CUSTOMERS, TRANSACTIONS, DATES

|

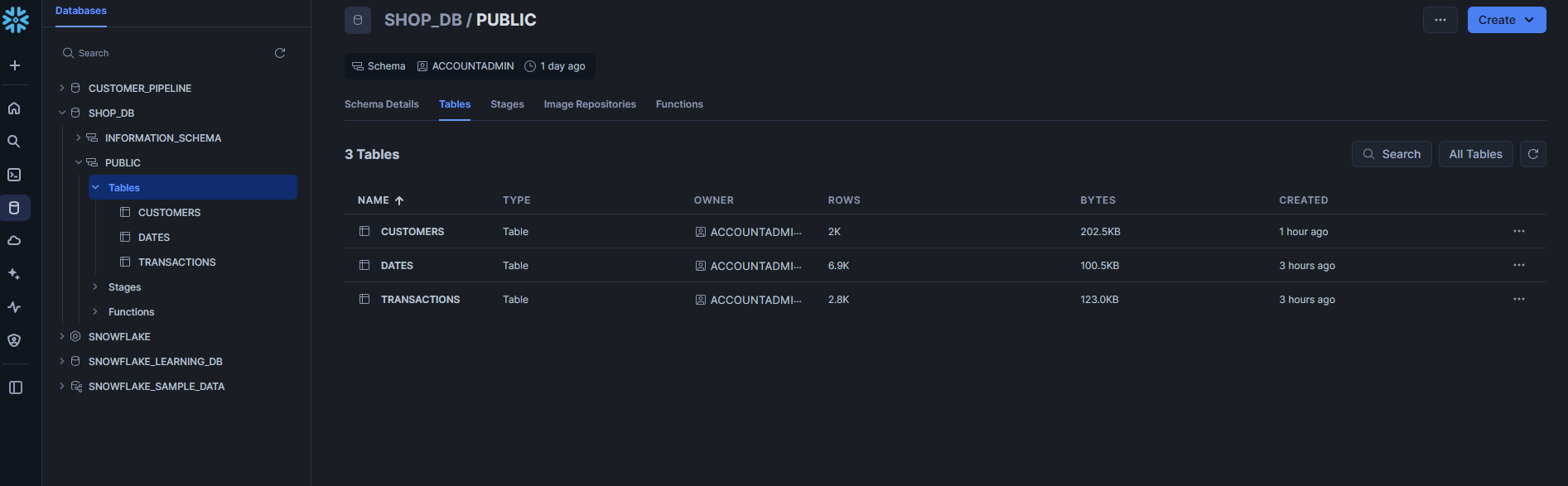
| - Analytics & Reporting

V

[Business Stakeholders / BI Tools]

**Key Notes:**

* **Direct Pandas → Snowflake** avoids temporary tables for small-medium datasets.
* For **very large datasets**, consider **staging tables** to:  
  + Apply batch inserts
  + Validate schema
  + Minimize transaction conflicts
* Deduplication is handled either:  
  + In Pandas (drop\_duplicates)
  + Or via Snowflake MERGE for incremental loads



### **4. Infrastructure Provisioning & Security**

* **IAM role** on S3:  
  + Avoids storing secrets in code.
  + Follows the principle of least privilege.
* **Snowflake roles and warehouses**:  
  + Separation of duties: **LOAD\_ROLE** vs **ANALYST\_ROLE**.
  + Warehouse scaling on-demand for batch ingestion.
* **Monitoring & logging**:  
  + Python logs to track processed files and row counts.
  + Snowflake query history for auditing.
* **Resilience considerations**:  
  + Re-runable pipelines (idempotent ingestion).
  + Automatic retries for failed insertions.
  + Optional integration with AWS Lambda or Airflow for orchestration.

## **Architecture for Business, Audit, and Regulatory Requirements**

**Goal:** Ensure the architecture allows stakeholders to answer business questions efficiently, supports audits, and complies with regulations such as GDPR or data retention policies.

**Current Measures Implemented:**

1. **Business Analytics Readiness**
   * All source files (customers, transactions, dates) are standardized and normalized in Snowflake.
   * Deduplication ensures metrics (e.g., total sales per customer) are accurate.
   * Column naming conventions are consistent for simplified querying.
   * Data is partitioned logically: Customers, Transactions, Dates → enables joins for reporting.
2. **Audit & Traceability**
   * **S3 object tracking:** Every ingested file is logged with timestamp and row counts.
   * **Snowflake query history:** All insertion queries and row counts can be audited.
   * Each row’s ingestion can be traced back to a source file.
3. **Regulatory Compliance**
   * Personally Identifiable Information (PII) like emails and phone numbers is stored securely in Snowflake.
   * Access control is implemented via Snowflake roles. Only authorized roles can read/write sensitive tables.
   * S3 access uses **IAM roles**, not hardcoded credentials, following least-privilege principles.

## **Infrastructure Provisioning & Resiliency**

**Current Measures Implemented:**

* **Compute:** Snowflake warehouses provide elastic compute for ingestion and queries.
* **Storage:** S3 stores source CSV files; IAM roles are used to control access.
* **Pipeline Execution:** Python scripts for automated ingestion and cleaning.
* **Error Handling:** Python scripts log failed rows for troubleshooting.

## **Optimal Approach / Best Practices :**

* **Workflow Orchestration:** Use **AWS Step Functions**, **Airflow**, or **Glue Jobs** for automated, retryable pipelines.
* **Monitoring & Alerts:** AWS CloudWatch for S3 events, Snowflake query monitoring for failed inserts.
* **Disaster Recovery:** Snowflake handles data replication across regions; enable **S3 versioning** and **backup policies**.
* **Scaling:** Snowflake warehouses auto-scale to handle large loads efficiently.
* Use **Snowflake Time Travel** for row-level historical audits (Not familiar with).
* Integrate **AWS Secrets Manager** for Snowflake credentials rather than hardcoding passwords.
* Implement **row-level security** if certain customer or transaction info is restricted.
* Consider **data masking** for sensitive columns (e.g., emails or phone numbers) in production.

## **1. Optimal ERD Design**

**Tables & Keys**

| **Table** | **Primary Key** | **Notes** |
| --- | --- | --- |
| **CUSTOMERS** | CUSTOMER\_ID (surrogate PK) | Surrogate key ensures consistent FK relationships; EMAIL can remain unique index. |
| **TRANSACTIONS** | ORDERNUMBER + ORDERLINENUMBER | Use CUSTOMER\_ID FK instead of string name; ORDERDATE FK links to DATES table. |
| **DATES** | DATE\_ID (surrogate) or CALENDAR\_DATE | Surrogate key DATE\_ID helps performance for joins; CALENDAR\_DATE unique constraint. |

### **ERD Diagram (Textual Representation)**

+------------------+ +--------------------+ +------------------+

| CUSTOMERS | 1 \* | TRANSACTIONS | \* 1 | DATES |

|------------------| |-------------------| |------------------|

| CUSTOMER\_ID PK |-----------| CUSTOMER\_ID FK | | DATE\_ID PK |

| FIRST\_NAME | | ORDERNUMBER PK |-----------| CALENDAR\_DATE |

| LAST\_NAME | | ORDERLINENUMBER PK | | WEEKDAY\_NUMBER |

| COMPANY\_NAME | | PRODUCTCODE | | FISCAL\_WEEK\_OF\_YEAR |

| ADDRESS | | QUANTITYORDERED | | FISCAL\_MONTH\_NUMBER |

| CITY | | TOTAL\_AMOUNT | | FISCAL\_QUARTER |

| COUNTY | | ORDERDATE FK | | FISCAL\_YEAR |

| STATE | | ADDRESSLINE1 | | ... |

| POSTAL | | ADDRESSLINE2 | +------------------+

| PHONE1 | | CITY |

| PHONE2 | | STATE |

| EMAIL UNIQUE | | POSTALCODE |

| WEB | | COUNTRY |

+------------------+ | TERRITORY |

| CONTACTFIRSTNAME |

| CONTACTLASTNAME |

| DEALSIZE |

+-------------------+

# PART 2 Analytics

The Full analytics and

## Proposing a scheduling approach to provide insights on a daily basis.

**Key Components:**

1. **ETL/Transformation Job**
   * Extract latest transactions from Snowflake (incremental load — only new/updated rows since last run).
   * Clean & standardize date columns (as we did for **STD\_ORDERDATE**).
   * Aggregate metrics (weekly, monthly, quarterly fiscal metrics).
   * Calculate additional KPIs (refunds after restocking fee, net revenue, items sold, top customers, etc.).
2. **Storage/Reporting Tables**
   * Maintain pre-aggregated tables in Snowflake:  
     + **FISCAL\_WEEKLY\_METRICS**
     + **FISCAL\_MONTHLY\_METRICS**
     + **FISCAL\_QUARTERLY\_METRICS**
     + **CUSTOMER\_SEGMENT\_METRICS**
   * Tables can be **incrementally updated** daily.
3. **Visualization / BI Layer**
   * Tools like **Power BI, Tableau, Looker, or Plotly Dash** can point to these Snowflake tables.
   * Reports automatically refresh daily when new data is available.
4. **Scheduling/Orchestration**
   * Use a **workflow orchestration tool** to schedule the daily job:  
     + **Apache Airflow** (open-source, supports DAGs, retries, alerts).
     + **AWS Glue / Step Functions** if fully on AWS.
     + **dbt Cloud** with Snowflake integration for transformations.
     + **Cron Job** (simpler setups) running a Python notebook/script via Papermill or Jupyter nbconvert.

## **2. Recommended Daily Workflow**

1. **Extract**
   * Pull new transactions from Snowflake using a timestamp filte**r (WHERE STD\_ORDERDATE > last\_run\_date).**
2. **Transform**
   * Standardize dates.
   * Merge with fiscal periods.
   * Calculate refunds, net revenue, top customers, segments, and other KPIs.
3. **Load**
   * Update the pre-aggregated tables in Snowflake using **MERGE** statements to prevent duplicates.
   * Maintain historical snapshots for auditability.
4. **Visualize**
   * Dashboards (Power BI, Tableau, Plotly Dash) query these pre-aggregated tables.
   * Optionally generate summary PDFs or email reports for stakeholders.

## **3. Best Practices**

* **Incremental loads**: Only process new or updated transactions each day to reduce compute and runtime.
* **Idempotent operations**: Ensure the daily job can safely run multiple times without duplicating rows.
* **Monitoring & Alerts**: Email or Slack notifications if the job fails.
* **Version Control**: Keep scripts and notebook code in Git for reproducibility.
* **Testing**: Schedule a weekly validation job comparing aggregates against source to catch anomalies.

**Optional Enhancements**

* **Caching / Incremental Aggregates**: Store previous aggregates and only update the delta.
* **Alerts on KPI anomalies**: Trigger emails if revenue or refund metrics are unusual.
* **Automated visualization generation**: Export top 10 customers, charts, and summary tables as PDFs for stakeholders.