**SecretLab Data Engineering Case Study**

This section is a test of your conceptual data engineering understanding - you may code this out for illustration, or you may choose to simply discuss the issues with diagrams and sample code.

**Data Modelling**

Create\_DB\_tables\_pg.sql depicts how the data should be formatted after it has landed; Please describe how you would design a base and subsequent mart layers for OLAP purposes? Assume the business will need to perform frequent analysis of:

* Revenue
* Item Sales
* Variant Price Changes
* By Product
* By Customer
* By Date
* if payment has been made and the time it occurred

**Data Provided - Batchwise from OLTP**

* Order
* Order Line
* Transaction Status
* Product
* Variant

Notes:

* The transaction table is a webhook which lands whenever an update is made on any transaction.

Discuss your architecture considerations as well as pk/fks (soft or hard), indexes and partitions where appropriate - you may assume the real orders table is about 20-30 M records.

1. Base Layer:

- The base layer serves as the foundation for data storage and serves as the source for building the mart layers.

- We can use a relational database management system (RDBMS) to store the raw data.

- Create the following tables: `orders`, `order\_lines`, `transaction`, `product`, and `variant`.

- Establish primary keys (PK) and foreign keys (FK) to maintain data integrity and enable relationships between tables.

Based on the provided tables schema, the following keys can be considered:

- `orders`: Primary Key (id)

- `order\_lines`: Primary Key (id), Foreign Key (order\_id references orders(id))

- `transaction`: Primary Key (id), Foreign Key (order\_id references orders(id))

- `product`: Primary Key (id)

- `variant`: Primary Key (id), Foreign Key (product\_id references product(id))

- Indexes can be added to columns that are frequently used for filtering or joining data to improve query performance.

- Considering the large number of records in the `orders` table, partitioning the table based on a suitable criterion (e.g., range-based partitioning on `PROCESSED\_TIMESTAMP`) can enhance query performance and data management.

2. Mart Layer:

- The mart layer involves transforming and aggregating data from the base layer to facilitate efficient analysis.

- Design separate mart tables for each analysis requirement (revenue, item sales, variant price changes, etc.).

- Build the mart tables by performing ETL (Extract, Transform, Load) processes on the base layer data. This involves extracting the required data, applying necessary transformations (e.g., aggregations, calculations), and loading it into the mart tables.

- We canonsider denormalizing the data in the mart tables for better query performance, reducing the need for joins during analysis.

- We can use a distributed processing framework like Apache Spark or a columnar database like Apache Cassandra for efficient data processing and querying.

- Ale we need to establish appropriate primary keys, foreign keys, and indexes based on the structure and requirements of each mart table.

- Partition the mart tables based on the analysis requirements and query patterns. For example, partitioning by date or customer ID can aid in querying data by date or customer.

**Data Pipelining**

**Pipelining**

How would you push these data sets assume each new file is **incremental** from S3. Assume that there is a central data warehouse which the BI tool connects to (not S3).

Provide a DFD or orchestration diagram, or write up an orchestration plan and tools you may use.

Orchestration plan:

1. New file uploaded to S3 triggers AWS Lambda.

2. Lambda function reads the file from S3 and extracts the data.

3. Extracted data is sent to AWS Glue for transformation.

4. AWS Glue performs data transformation and prepares it for loading.

5. Transformed data is loaded into the central data warehouse using AWS Glue or AWS Data Pipeline.

6. After loading, data validation is performed using AWS Glue or custom scripts.

7. BI tool connects to the central data warehouse for analysis and reporting.

1. Data Extraction:

* AWS Lambda can be used to trigger the extraction process whenever a new file is uploaded to S3.
* Lambda can use AWS SDKs to interact with S3, read the new file, and extract the data.

2. Data Transformation:

* AWS Glue can be used to transform the extracted data into a desired format or schema.

3. Data Loading:

* we can use AWS Glue or AWS Data Pipeline to load the transformed data into the central data warehouse.
* Both services can handle the orchestration of the loading process and perform efficient data transfers.

4. Data Validation:

* We can Validate the data before the data is loaded into the warehouse, we can validate it for quality assurance.
* AWS Glue or custom scripts can be used to perform data validation checks, such as verifying data integrity or consistency and overall data quality.

5. BI Tool Connectivity:

* Connect your BI tool to the central data warehouse using its supported connectors or APIs.
* Configure the BI tool to access the transformed and validated data for analysis and reporting.

**Cloud Engineering**

The Data Scientist has developed an unsupervised model to help analyse traffic flow and conversions into our online shop.

The model is expected to analyse web-traffic data (sourced from the data warehouse) and output some model results daily. The model is delivered to you as a python module. Model output from the main function is expected to be a dataframe.

You have been tasked to deploy the solution, and allow BI to develop Tableau dashboards based on all the data science model's output for downstream business users. Discuss how you would implement this system, and provide a simple systems diagram.

1. Provide a systems diagram only.

Model Integration in the deployment environment

Data Extraction from Data Warehouse in appropriate frequency

Model Execution scheduled on a daily basis

Store Model output into a DB/file/cloud storage service

Connect tableau to data store location

**Analytical SQL**

**Problem 1**

Sometimes products for whatever reason stop selling and a symptom can be an item that was selling well faces a stock out or delisting (or something else). Write a query that shows products that have sold for more than 30 days in the last 60 days, but hasn't had sales for the last week.

You may assume a sales table schema of your preference.

1. Date
2. Product\_id
3. Total Items sold to date
4. number of days with sales
5. number of dates in the recent history where sales have ceased

What would be the best way to implement this in Date/Product\_id/Sum(sales)... type Data Mart with other sales information (i.e. without filters and group by)?

SELECT t.Product\_id AS Products from (SELECT \* from sales\_table WHERE DATEDIFF(day,Date,GETDATE()) <=60 AND no\_of\_days\_with\_sales >30) t WHERE t.DATEDIFF(day,Date,GETDATE())<=7 AND no\_of\_days\_without\_sales >=7

**Problem 2 - SQL Optimization**

Our Data Analyst needs to compute a fulfillment promised date for each of the orderline deliveries based on the Service Level Agreements (SLA) of the logistics provider selected for the delivery.

The fulfillment promised date is computed based on the fulfillment creation date plus the number of promised working days for delivery provided by the SLA of the logistics provider.

The working\_days table contains the working days of the world up to 2030.

Part of the query he is using is as follows:

select

...

sum(is\_working\_day::integer order by working\_days.date) as number\_of\_work\_days

...

from order\_line

left join working\_days

on order\_line.fulfillment\_creation\_date < working\_days.date

He is facing query timeout issues consistently when computing the provider KPIs. What are the steps you would take to help him with this problem?

Some Steps to help:

-Use a Sub query for aggregation:

SELECT   
 …

Wd\_agg.number\_of\_work\_days

…

FROM order\_line

LEFT JOIN (

SELECT dat, SUM(is\_working\_day::integer) AS number\_of\_work\_days

FROM working\_days

GROUP BY date) wd\_agg ON order\_line.fulfillment\_creation\_date < wd\_agg.date

- Ensure that there are indexes on the join columns (`fulfillment\_creation\_date` and `date`) in both the `order\_line` and `working\_days` tables.

-