

## Question 4:

In general terms, how would you design a system where data from three different sources is combined into one large SQL table? Assume that the source data changes at different rates, and the destination table should update more than once per day.

## Answer 4:

The process to update data into a target table from three different data sources involves several key steps:

### 1. Data Ingestion into Source OLTP Table:

- First, data is collected from various external sources. These could be databases, APIs, or other systems where transactions happen. This raw data is stored in a **source table** (which is like a temporary holding area) for further processing.

### 2. Define Target Table schema based on the business requirement:

- Based on what the business needs, we create a **target table** that will store the final data in a structured way. This table's structure is designed to be efficient and easy to query, even though the data is coming from multiple sources that may have very different formats. We define the fields, data types, and relationships so it all makes sense for the business.

### 3. Processing the Data While Ensuring No Duplicate Work:

- For each data source, we write a **procedure** that extracts the data from the source, transforms it as needed, and loads it into the target table.
- To avoid processing the same data more than once (for example, if the process runs again), we use a **processed flag**. Once a piece of data is successfully moved to the target table, the flag is set to "True," so it's not processed again. This way, the process is **idempotent** meaning we can run it multiple times without affecting data that's already been processed.

### 4. Handling Errors During Data Processing:

- Sometimes, things can go wrong during the process. Maybe the data is in the wrong format, or some fields are missing. To manage this, we have a special **error-handling procedure**.
- If something goes wrong, this procedure catches the error, logs it in a separate error table, and allows us to review and fix issues later. This way, we don't lose track of any problems.

### 5. Orchestrating the Whole Process:

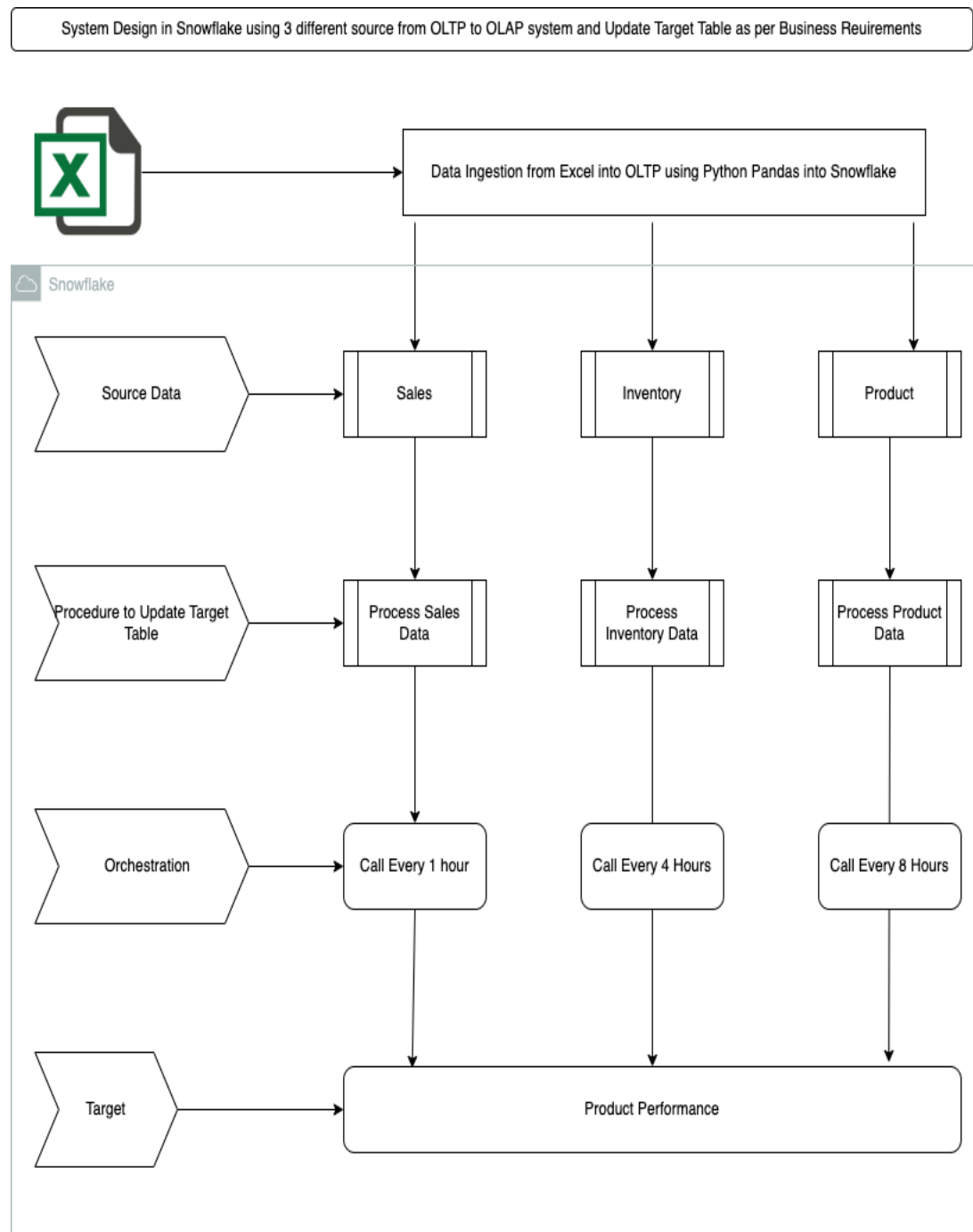
- Since data from each source might update at different rates, we use an orchestration tool like **Airflow** to manage the scheduling. Airflow allows us to set up a workflow that calls the different procedures at the right times, ensuring that data from all sources gets processed regularly and efficiently.

### 6. Using the Final Data:

- Once the target table is populated with all the necessary data, it's ready to be used. The business can now access this cleaned and processed data for reporting, analytics, or whatever purpose is required.

Let's understand using example in detail:

## Architecture



This system is designed to take data from three different sources — **Sales**, **Inventory**, and **Product** — and combine them into a central table in **Snowflake**, called **product\_performance**. The goal is to ensure that the central table is always up to date with the latest data from each source, even though each source updates at different times.

### Step 1: Data Ingestion

The process begins by **ingesting data** from Excel (or a similar data source) using **Python with Pandas**. This data ingestion pulls data from the three different sources: Sales, Inventory, and Product. The data is then loaded into separate staging tables in the OLTP system, which eventually feeds into Snowflake for further processing.

## Step 2: Source Tables in Snowflake

Once the data is ingested, it is stored in three separate **source tables** in Snowflake:

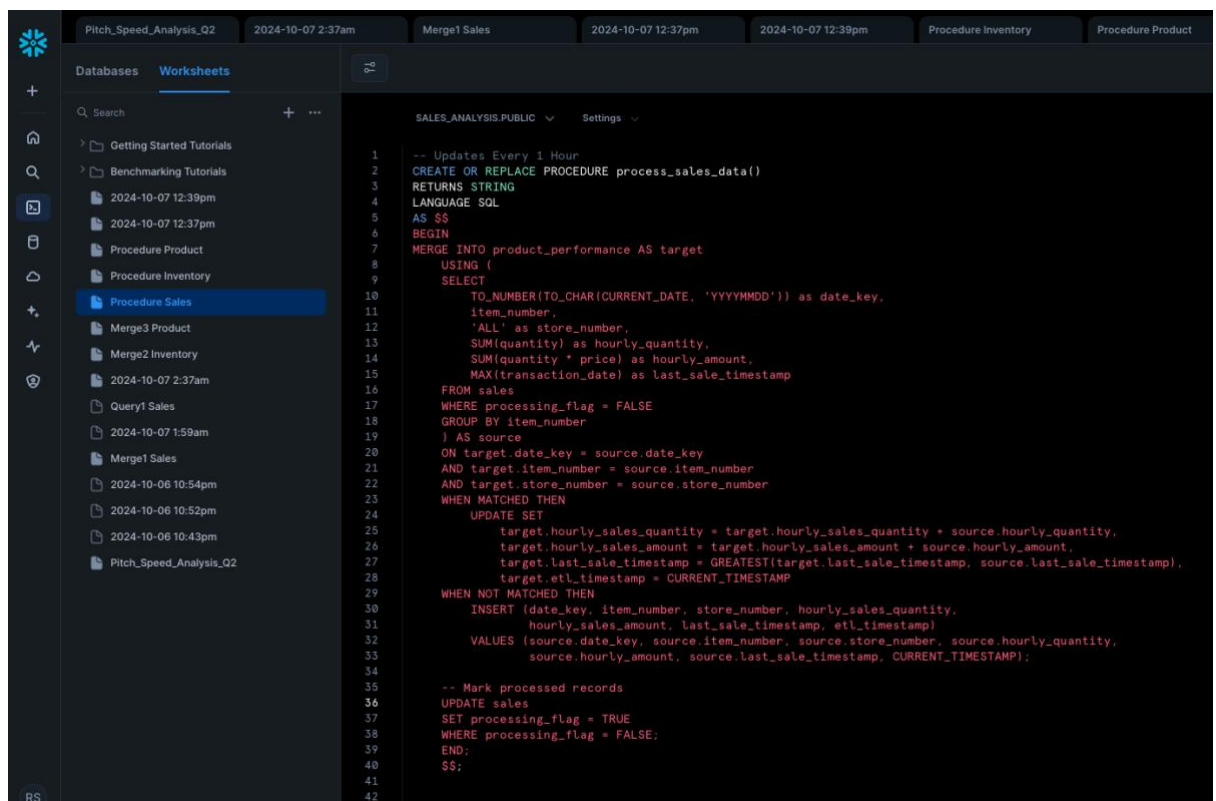
- **Sales Table:** This stores all sales-related data, such as transaction details.
- **Inventory Table:** This holds data related to stock levels and inventory movements.
- **Product Table:** This contains information about product pricing and details.

Each of these tables is updated at different intervals because the rate of change for each type of data is different. Sales might need to be updated more frequently, while inventory and product data may change less often.

## Step 3: Stored Procedures to Process Data

Three **stored procedures** are created in Snowflake to process the data from each of the source tables and push the updates to the **product\_performance** table:

- **process\_sales\_data:** This procedure is responsible for processing and updating the sales data.



The screenshot displays the Snowflake SQL Editor interface. On the left, a sidebar shows a list of databases and worksheets, with 'Procedure Sales' selected. The main editor area shows the SQL code for the 'process\_sales\_data' stored procedure. The code is written in SQL and includes comments indicating it updates every hour. It uses a MERGE statement to update the 'product\_performance' table with data from the 'sales' table. The code includes logic for handling existing records (UPDATE SET) and new records (INSERT). The procedure also marks processed records by setting a 'processing\_flag' to TRUE.

```
1  -- Updates Every 1 Hour
2  CREATE OR REPLACE PROCEDURE process_sales_data()
3  RETURNS STRING
4  LANGUAGE SQL
5  AS $$
6  BEGIN
7  MERGE INTO product_performance AS target
8  USING (
9  SELECT
10     TO_NUMBER(TO_CHAR(CURRENT_DATE, 'YYYYMMDD')) as date_key,
11     item_number,
12     'ALL' as store_number,
13     SUM(quantity) as hourly_quantity,
14     SUM(quantity * price) as hourly_amount,
15     MAX(transaction_date) as last_sale_timestamp
16 FROM sales
17 WHERE processing_flag = FALSE
18 GROUP BY item_number
19 ) AS source
20 ON target.date_key = source.date_key
21 AND target.item_number = source.item_number
22 AND target.store_number = source.store_number
23 WHEN MATCHED THEN
24     UPDATE SET
25         target.hourly_sales_quantity = target.hourly_sales_quantity + source.hourly_quantity,
26         target.hourly_sales_amount = target.hourly_sales_amount + source.hourly_amount,
27         target.last_sale_timestamp = GREATEST(target.last_sale_timestamp, source.last_sale_timestamp),
28         target.etl_timestamp = CURRENT_TIMESTAMP
29 WHEN NOT MATCHED THEN
30     INSERT (date_key, item_number, store_number, hourly_sales_quantity,
31            hourly_sales_amount, last_sale_timestamp, etl_timestamp)
32     VALUES (source.date_key, source.item_number, source.store_number, source.hourly_quantity,
33            source.hourly_amount, source.last_sale_timestamp, CURRENT_TIMESTAMP);
34
35 -- Mark processed records
36 UPDATE sales
37 SET processing_flag = TRUE
38 WHERE processing_flag = FALSE;
39 END;
40 $$;
```

- **process\_inventory\_data:** This one handles updates related to inventory data.

The screenshot shows a database IDE with a sidebar on the left containing a 'Databases' tab and a 'Worksheets' tab. The 'Worksheets' tab is active, displaying a list of worksheets including 'Getting Started Tutorials', 'Benchmarking Tutorials', and several data files. The 'Procedure Inventory' worksheet is selected. The main editor area shows the SQL code for the 'process\_inventory\_data' procedure, which is a PL/SQL procedure that updates the 'product\_performance' table. The code includes comments, a 'CREATE OR REPLACE' statement, a 'RETURN' statement, and a 'MERGE INTO' statement. The procedure is designed to update the 'current\_inventory' and 'last\_inventory\_update' fields in the 'product\_performance' table based on the 'inventory' table. It also includes an 'UPDATE' statement to set the 'processing\_flag' to 'TRUE' for the 'inventory' table.

```

1  -- Updates Every 4 Hours
2  CREATE OR REPLACE PROCEDURE process_inventory_data()
3  RETURNS STRING
4  LANGUAGE SQL
5  AS $$
6  BEGIN
7
8  MERGE INTO product_performance AS target
9  USING (
10     SELECT
11         TO_NUMBER(TO_CHAR(CURRENT_DATE, 'YYYYMMDD')) as date_key,
12         item_number,
13         store_number,
14         quantity as current_inventory,
15         updated_date as last_updated
16     FROM inventory
17     WHERE processing_flag = FALSE
18 ) AS source
19 ON target.date_key = source.date_key
20 AND target.item_number = source.item_number
21 WHEN MATCHED THEN
22     UPDATE SET
23         target.current_inventory = source.current_inventory,
24         target.last_inventory_update = source.last_updated,
25         target.etl_timestamp = CURRENT_TIMESTAMP
26 WHEN NOT MATCHED THEN
27     INSERT (date_key, item_number, store_number, current_inventory,
28            last_inventory_update, etl_timestamp)
29     VALUES (source.date_key, source.item_number, source.store_number,
30            source.current_inventory, source.last_updated, CURRENT_TIMESTAMP);
31
32 UPDATE inventory
33 SET processing_flag = TRUE
34 WHERE processing_flag = FALSE;
35
36 END;
37 $$;

```

- **process\_product\_data:** This procedure deals with product updates like pricing changes.

The screenshot shows the same database IDE as the first image, but with the 'Procedure Product' worksheet selected. The main editor area displays the SQL code for the 'process\_product\_data' procedure. This procedure is a PL/SQL procedure that updates the 'product\_performance' table. It includes comments, a 'CREATE OR REPLACE' statement, a 'RETURN' statement, and a 'MERGE INTO' statement. The procedure is designed to update the 'current\_base\_price', 'current\_discount', 'effective\_price', 'last\_price\_update', and 'etl\_timestamp' fields in the 'product\_performance' table based on the 'product' table. It also includes an 'UPDATE' statement to set the 'processing\_flag' to 'TRUE' for the 'product' table.

```

1  -- Updates Every 8 Hours
2  CREATE OR REPLACE PROCEDURE process_product_data()
3  RETURNS STRING
4  LANGUAGE SQL
5  AS $$
6  BEGIN
7
8  MERGE INTO product_performance AS target
9  USING (
10     SELECT
11         TO_NUMBER(TO_CHAR(CURRENT_DATE, 'YYYYMMDD')) as date_key,
12         item_number,
13         base_price,
14         discount,
15         base_price * (1 - discount/100) as effective_price,
16         effected_date
17     FROM product
18     WHERE processing_flag = FALSE
19 ) AS source
20 ON target.date_key = source.date_key
21 AND target.item_number = source.item_number
22 WHEN MATCHED THEN
23     UPDATE SET
24         target.current_base_price = source.base_price,
25         target.current_discount = source.discount,
26         target.effective_price = source.effective_price,
27         target.last_price_update = source.effected_date,
28         target.etl_timestamp = CURRENT_TIMESTAMP
29 WHEN NOT MATCHED THEN
30     INSERT (date_key, item_number, store_number, current_base_price,
31            current_discount, effective_price, last_price_update, etl_timestamp)
32     VALUES (source.date_key, source.item_number, 'ALL', source.base_price,
33            source.discount, source.effective_price, source.effected_date,
34            CURRENT_TIMESTAMP);
35
36 UPDATE product
37 SET processing_flag = TRUE
38 WHERE processing_flag = FALSE;
39
40 END;
41 $$;

```

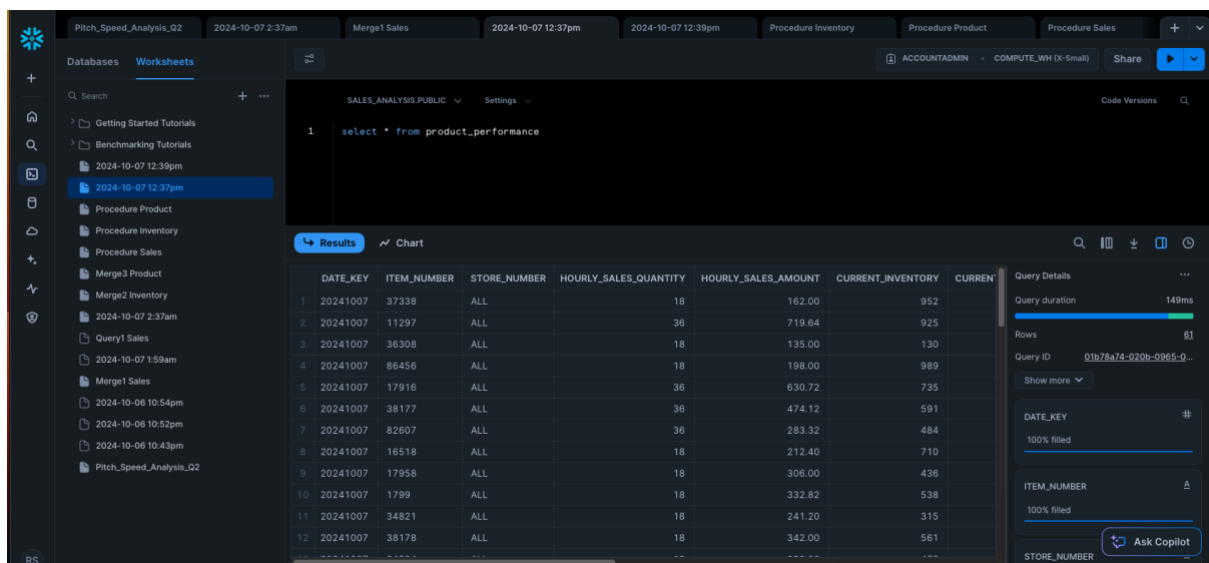
## Step 4: Scheduling with Airflow (Future Scope)

To make sure that the data is updated regularly, the system uses **Airflow** to schedule and automate the execution of these stored procedures:

- The **process\_sales\_data** procedure is scheduled to run **every hour**, ensuring that the latest sales data is always up to date in the product\_performance table.
- The **process\_inventory\_data** procedure runs **every 4 hours** since inventory data doesn't need to be updated as frequently as sales data.
- The **process\_product\_data** procedure is scheduled to run **every 8 hours**, as product information, like pricing, typically changes less often.

## Step 5: Updating the Target Table

- The **product\_performance** table is the final destination for all the processed data. Each time a procedure runs, the data from the corresponding source table (Sales, Inventory, or Product) is processed and the central **product\_performance** table is updated. This table combines all the data into one place, providing a single view of how products are performing based on sales, inventory, and pricing information.



The screenshot displays a SQL IDE interface with a query editor and a results pane. The query editor shows a SQL query: `select * from product_performance`. The results pane displays a table with 12 rows of data. The table has columns: DATE\_KEY, ITEM\_NUMBER, STORE\_NUMBER, HOURLY\_SALES\_QUANTITY, HOURLY\_SALES\_AMOUNT, CURRENT\_INVENTORY, and CURRENT\_INVENTORY. The data is as follows:

	DATE_KEY	ITEM_NUMBER	STORE_NUMBER	HOURLY_SALES_QUANTITY	HOURLY_SALES_AMOUNT	CURRENT_INVENTORY	CURRENT_INVENTORY
1	20241007	37338	ALL	18	162.00	952	
2	20241007	11297	ALL	36	719.64	925	
3	20241007	36308	ALL	18	135.00	130	
4	20241007	86456	ALL	18	198.00	989	
5	20241007	17916	ALL	36	630.72	735	
6	20241007	38177	ALL	36	474.12	591	
7	20241007	82607	ALL	36	283.32	484	
8	20241007	16518	ALL	18	212.40	710	
9	20241007	17958	ALL	18	306.00	436	
10	20241007	1799	ALL	18	332.82	538	
11	20241007	34821	ALL	18	241.20	315	
12	20241007	38178	ALL	18	342.00	561	