DATAWAREHOUSE TECHINCAL CHALLENGE(LakshmiPriyaVaragogula)

Given:

The Challenge: Data Warehouse and Data Pipeline assets like raw data provided The provdied data folder that has several pipe delimited gzipped files of raw data. The names of the files start with either hier or fact to signify whether they have hierarchy (dimension) or fact data. The word following hier or fact indicates the table name for the raw data. Each file has a header row with column names. The hier files have id and label columns for each level in the hierarchy. For the most part you can assume that the left most column is the primary key, but you should ensure that you draw out a proper structure by looking at the many-to-one relationships that the data manifests.

1. You must draw out an ER diagram showing raw table structure and any relationships between them that you can infer using column names. You may use schema inference tools, but you must document what you used and why. You must add the final ER diagram and any documentation explaining it to your submission's Github repository.

2. You must draw a dataflow and architecture diagram illustrating how data moves from raw to refined stages and highlighting key processing steps. In your video, please describe how this aligns with best practices for data pipelines built using your chosen stack.

3. You must build a pipeline that a. Loads this raw data into the data warehouse from external storage such as Azure Blobs, AWS S3 or the like. You must write basic checks such as non-null, uniqueness of primary key, data types. Also check for foreign key constraints between fact and dimension tables. Do it for at least one hier (dimension), and one fact table.

b. Create a staging schema where the hierarchy table has been normalized into a table for each level and the staged fact table has foreign key relationships with those tables.

c. Create a refined table called mview_weekly_sales which totals sales_units, sales_dollars, and discount_dollars by pos_site_id, sku_id, fsclwk_id, price_substate_id and type. d. BONUS: write transformation logic that will incrementally calculate all the totals in the above table for partially loaded data.

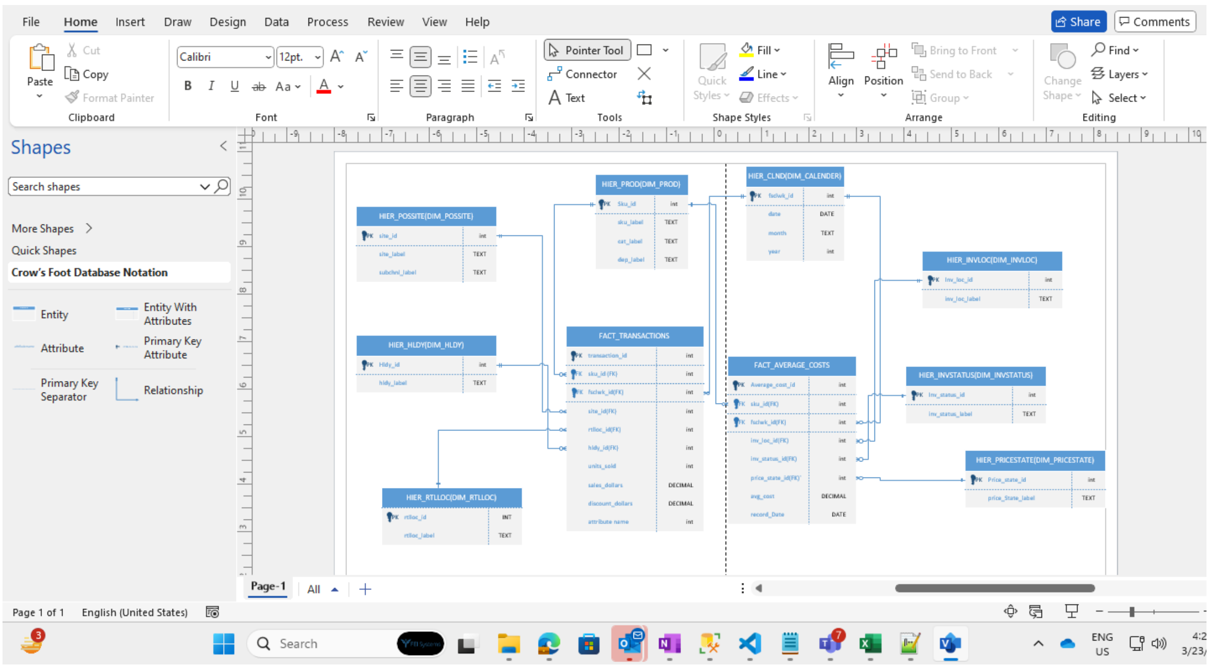
SOLUTION:

This challenge aligns well with my expertise in **Snowflake, AWS (S3, IAM, Lambda, DMS), SQL, Power BI, and data pipeline automation.**

I am using **AWS + Snowflake** for the solution:

- **Storage:** AWS S3 (Raw Data Storage)
- **ER Diagram:** Microsoft Visio
- **Ingestion:** Snowpipe for continuous ingestion
- **Processing:** Snowflake Staging & Transformations
- **Orchestration:** AWS Lambda & Snowflake Tasks
- **Visualization:** Power BI (Optional)

ER diagram



Step 1: Open Visio and Select the Right Template

1. **Open Microsoft Visio**
2. Click **New > Crow's Foot Database Notation** (Best for ER diagrams)
3. Click **Create**

Step 2: Add Fact and Dimension Tables

1. In the **left panel**, find **"Entity" (or Table)**.
2. Drag and drop **two fact tables** (fact_averagecosts, fact_transactions) onto the canvas.
3. Drag and drop **all dimension tables** (hier_prod, hier_clnd, etc.) onto the canvas.
4. Rename each table according to your schema.

Step 3: Define Primary Keys (PK) and Foreign Keys (FK)

For Each Fact Table:

1. **Click on the Fact Table** → Go to the **Columns section**
2. Set the **Primary Key (PK)** (e.g., average_cost_id for fact_averagecosts)
3. Add **Foreign Keys (FKs)** that reference dimension tables

  - Example for fact_transactions:
    - sku_id (FK → hier_prod.sku_id)
    - fsclwk_id (FK → hier_clnd.fsclwk_id)
    - pos_site_id (FK → hier_possite.pos_site_id)
    - rtlloc_id (FK → hier_rtlloc.rtlloc_id)

Step 4: Connect Fact Tables to Dimension Tables

1. **Go to the Connector Tool (Relationships Tool)**
   - In **Crow's Foot Notation**, relationships are represented with **lines**.
2. **Click & Drag from Fact Table FK to Dimension Table PK**
   - Example:
     - Drag from fact_transactions.sku_id **→** hier_prod.sku_id
     - Drag from fact_transactions.fsclwk_id **→** hier_clnd.fsclwk_id
3. **Choose Relationship Type**
   - In Crow's Foot Notation, use:
     - **One-to-Many (1:M)** → Dimension Table (1) to Fact Table (M)
     - **Example:** hier_prod.sku_id (1) → fact_transactions.sku_id (M)
4. **Repeat for all other dimension tables.**

Step 5: Verify the Diagram

- Ensure **each fact table is connected to the correct dimension tables**.
- Check that **foreign keys correctly reference primary keys**.

After keen analysis with raw data where each table contains it's id and label names etc.,

As per my understanding with given data In the Entity RelationShip diagram you will see two fact tables with dimensions, Hier(DIM) tables as listed below

1)fact.avergecosts.dlm

2)fact.transactions.dlm

3)hier.clnd.dlm

4)hier.hldy.dlm

5)hier.invloc.dlm

6)hier.invstatus.dlm

7)hier.possite.dlm

8)hier.pricestate.dlm

9)hier.prod.dlm

10)hier.rtlloc.dlm

PREREQUISITES for implementing pipeline:

-A Snowflake account with required permissions.

- AWS S3 bucket access.

- Snowflake external stage configured.

2) DATAPIPELINE STEPS

- **Infer Schema** from raw .dlm file.
- **Auto-Load Data** into **Landing Tables** from **S3 via Snowpipe**.
- **Create Streams** for incremental data tracking.
- **Use Dynamic Tables** to transform raw data.
- **Normalize Hierarchies** and establish fact-dimension relationships.
- **Load into Fact & Dimension Tables** with constraints.
- **Implement Incremental Updates** using Streams and Tasks.
- **Create Materialized View** for **weekly sales aggregation**.
- **Define a Stored Procedure** to update data.
- **Schedule the Update Task** every 5 minutes.

STEP :1 Using INFER SCHEMA

I am using snowflake here , as I ended with free trail account .For now, I have used my working snowflake account and it does work with below code:

Considering FACT_TRANSACTIONS AND HIER_PRODUCTS for my pipeline implementation

For fact_transactions:

```
SELECT *

FROM TABLE(

  INFER_SCHEMA(

    LOCATION => '@my_s3_stage/fact_transactions.dlm',
        FILE_FORMAT => '(TYPE=CSV, FIELD_DELIMITER=''|'', SKIP_HEADER=1)'
    )
);
```

For hier_products:

```
SELECT *
FROM TABLE(
    INFER_SCHEMA(
        LOCATION => '@my_s3_stage/hier_products.dlm',
        FILE_FORMAT => '(TYPE=CSV, FIELD_DELIMITER=''|'', SKIP_HEADER=1)'
    )
);
```

To extract schema structure from .dlm files **without loading data**, use the above code in snowflake This will analyze the **header row** and detect column names, data types, and delimiters.

2. Create Snowflake Objects (Stages, File Format, Snowpipe)

Create File Format

```
CREATE OR REPLACE FILE FORMAT my_dlm_format
TYPE = 'CSV'
FIELD_OPTIONALLY_ENCLOSED_BY='"'
FIELD_DELIMITER='|';
```

Create External Stage for S3

```
CREATE OR REPLACE STAGE s3_stage
URL='s3://your-bucket-name/'
STORAGE_INTEGRATION=your_integration
FILE_FORMAT = my_dlm_format;
```

Create Snowpipe for Auto-Loading

```
CREATE OR REPLACE PIPE transactions_pipe AUTO_INGEST = TRUE AS
```

```
COPY INTO raw_transactions
FROM @s3_stage/fact_transactions.dlm
FILE_FORMAT = (FORMAT_NAME = my_dlm_format);
```

---

3. Create Landing Tables (Raw Data Storage)

```
CREATE OR REPLACE TABLE raw_transactions (
  transaction_id INT,
  product_id INT,
  store_id INT,
  sales_amount FLOAT,
  sales_units INT,
  discount FLOAT,
  transaction_date DATE,
  load_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

CREATE OR REPLACE TABLE raw_products (
  product_id INT,
  product_name STRING,
  category STRING,
  price FLOAT,
  load_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);
```

---

4. Create Streams for Change Tracking

```
CREATE OR REPLACE STREAM transaction_stream ON TABLE raw_transactions;
CREATE OR REPLACE STREAM product_stream ON TABLE raw_products;
```

---

5. Create Dynamic Tables (Transform & Normalize Data)

```
CREATE OR REPLACE DYNAMIC TABLE dt_fact_transactions
LAG = '5 minutes'
WAREHOUSE = my_wh
AS
SELECT
   transaction_id,
   product_id,
   store_id,
   sales_amount,
   sales_units,
   discount,
   transaction_date
FROM raw_transactions;

CREATE OR REPLACE DYNAMIC TABLE dt_dim_products
LAG = '5 minutes'
WAREHOUSE = my_wh
AS
SELECT
   product_id,
   product_name,
   category,
   price
FROM raw_products;
```

6. Create Final Fact & Dimension Tables

```
CREATE OR REPLACE TABLE fact_transactions (
    transaction_id INT PRIMARY KEY,
    product_id INT,
    store_id INT,
    sales_amount FLOAT,
    sales_units INT,
    discount FLOAT,
    transaction_date DATE,
    FOREIGN KEY (product_id) REFERENCES dim_products(product_id)
);

CREATE OR REPLACE TABLE dim_products (
    product_id INT PRIMARY KEY,
    product_name STRING,
    category STRING,
    price FLOAT
);
```

---

7. Task for Incremental Data Load

```
CREATE OR REPLACE TASK update_fact_dim
WAREHOUSE = my_wh
SCHEDULE = '5 MINUTE'
WHEN SYSTEM$STREAM_HAS_DATA('transaction_stream') OR SYSTEM$STREAM_HAS_DATA('product_stream')
AS
```

```
MERGE INTO fact_transactions t
USING (SELECT * FROM transaction_stream) s
ON t.transaction_id = s.transaction_id
WHEN MATCHED THEN UPDATE SET
    t.sales_amount = s.sales_amount,
    t.sales_units = s.sales_units,
    t.discount = s.discount
WHEN NOT MATCHED THEN
INSERT (transaction_id, product_id, store_id, sales_amount, sales_units, discount, transaction_date)
VALUES (s.transaction_id, s.product_id, s.store_id, s.sales_amount, s.sales_units, s.discount, s.transaction_date);
```

---

8. Materialized View for Weekly Sales Summary

```
CREATE OR REPLACE MATERIALIZED VIEW mview_weekly_sales AS
SELECT
    store_id,
    product_id,
    DATE_TRUNC('WEEK', transaction_date) AS fsclwk_id,
    SUM(sales_units) AS total_sales_units,
    SUM(sales_amount) AS total_sales_dollars,
    SUM(discount) AS total_discount_dollars
FROM fact_transactions
GROUP BY store_id, product_id, fsclwk_id;
```

---

9. Stored Procedure for Incremental Updates

```
CREATE OR REPLACE PROCEDURE update_sales_summary()
RETURNS STRING
LANGUAGE SQL
AS
$$
BEGIN
```

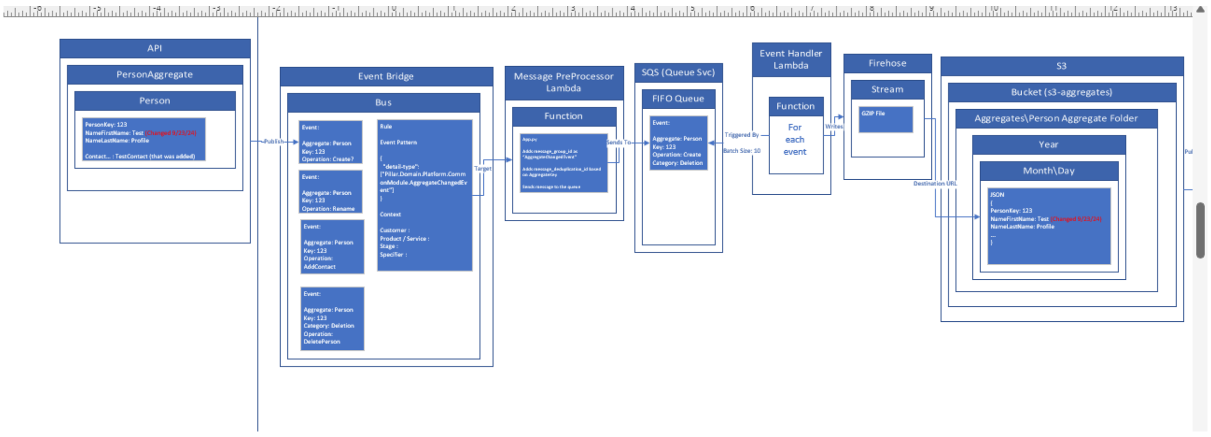
```
    INSERT INTO mview_weekly_sales
    SELECT
        store_id,
        product_id,
        DATE_TRUNC('WEEK', transaction_date) AS fsclwk_id,
        SUM(sales_units),
        SUM(sales_amount),
        SUM(discount)
    FROM transaction_stream
    GROUP BY store_id, product_id, fsclwk_id;

    RETURN 'Sales summary updated successfully';
END;
$$;
```

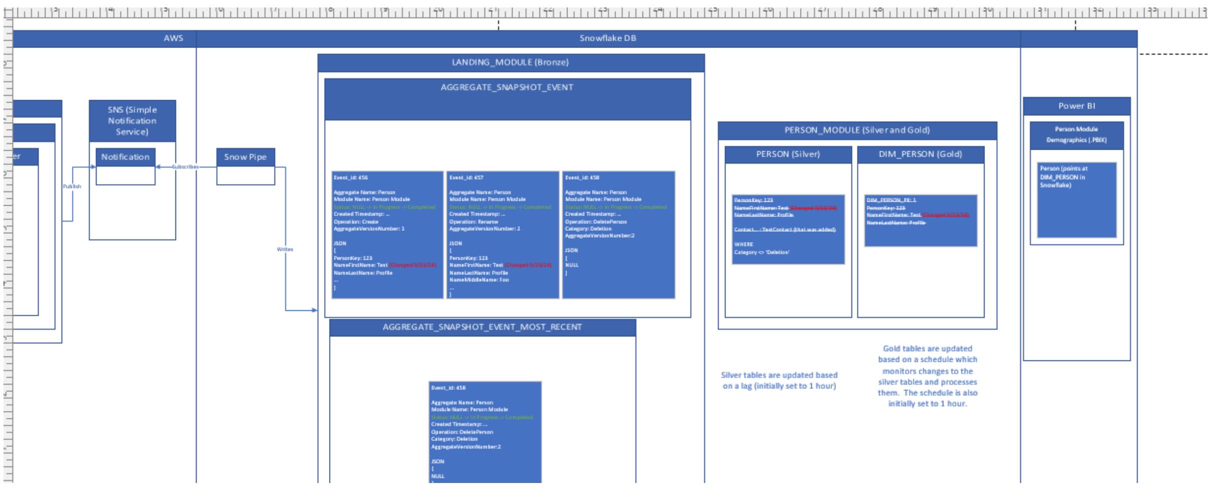
---

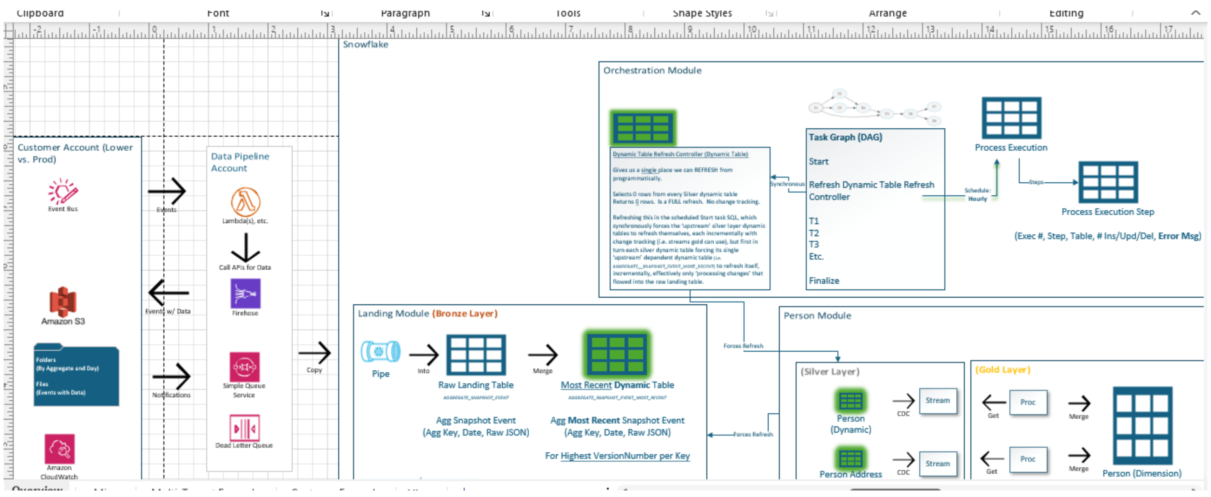
10. Schedule the Stored Procedure

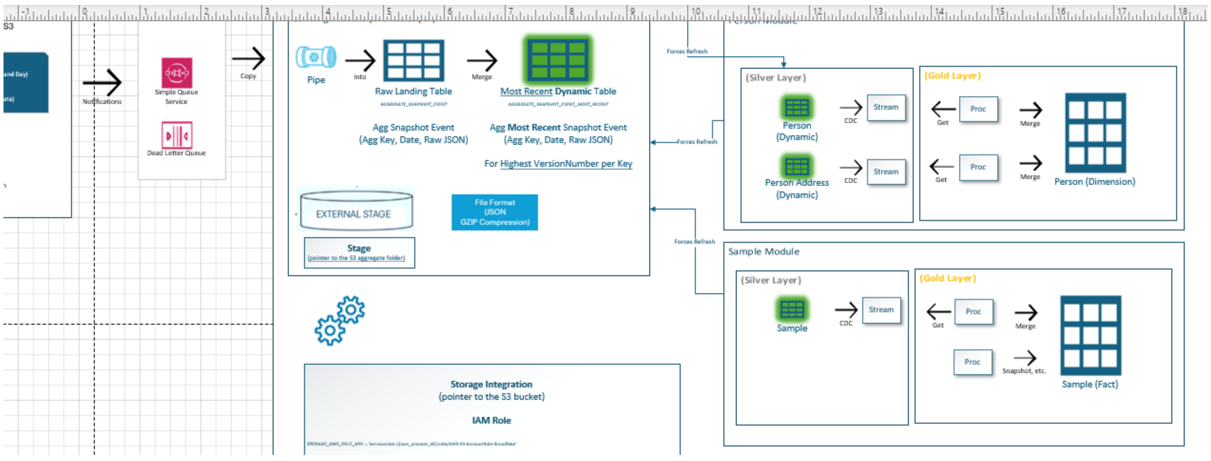
```
CREATE OR REPLACE TASK run_sales_summary
WAREHOUSE = my_wh
SCHEDULE = '5 MINUTE'
AS CALL update_sales_summary();
```

I am attaching the datapipeline flowcharts which explains end to end data pipeline from using AWS Services to Creating reports in which I got skilled using the below approach.


API
PersonAggregate
Person
Event Bridge
Bus
Message PreProcessor Lambda
Function
SQS (Queue Svc)
FIFO Queue
Event Handler Lambda
Function
For each event
Firehose
Stream
S3
Bucket (s3-aggregates)
Aggregates\Person Aggregate Folder
Year
Month\Day


AWS
Snowflake DB
LANDING_MODULE (Bronze)
AGGREGATE_SNAPSHOT_EVENT
SNS (Simple Notification Service)
Notification
Snow Pipe
AGGREGATE_SNAPSHOT_EVENT_MOST_RECENT
PERSON_MODULE (Silver and Gold)
PERSON (Silver)
DIM_PERSON (Gold)
Power BI
Silver tables are updated based on a lag (initially set to 1 hour)
Gold tables are updated based on a schedule which monitors changes to the silver tables and processes them. The schedule is also initially set to 1 hour.


Clipboard
Font
Paragraph
Tools
Shape Styles
Arrange
Editing
Snowflake
Customer Account (Lower vs. Prod)
Event Bus
Amazon S3
Amazon CloudWatch
Events
Event w/ Data
Notifications
Data Pipeline Account
Lambda(s), etc.
Call APIs for Data
Firehose
Simple Queue Service
Dead Letter Queue
Copy
Orchestration Module
Task Graph (DAG)
Start
Refresh Dynamic Table Refresh Controller
T1
T2
T3
Etc.
Finalize
Process Execution
Process Execution Step
(Exec #, Step, Table, # Ins/Upd/Del, Error Msg)
Landing Module (Bronze Layer)
Pipe
Into
Raw Landing Table
Merge
Most Recent Dynamic Table
Agg Snapshot Event (Agg Key, Date, Raw JSON)
Agg Most Recent Snapshot Event (Agg Key, Date, Raw JSON)
For Highest Version Number per Key
Person Module
(Silver Layer)
Person (Dynamic)
Person Address
CDC
Stream
(Gold Layer)
Proc
Get
Merge
Person (Dimension)
Forces Refresh


S3
Notifications
Simple Queue Service
Dead Letter Queue
Copy
Pipe
Into
Raw Landing Table
AGGREGATE_SNAPSHOT_EVENT
Merge
Most Recent Dynamic Table
AGGREGATE_SNAPSHOT_EVENT_MOST_RECENT
Agg Snapshot Event (Agg Key, Date, Raw JSON)
Agg Most Recent Snapshot Event (Agg Key, Date, Raw JSON)
For Highest VersionNumber per Key
EXTERNAL STAGE
File Format (JSON GZIP Compression)
Stage (pointer to the S3 aggregate folder)
Storage Integration (pointer to the S3 bucket)
IAM Role
Forces Refresh
Forces Refresh
Forces Refresh
(Silver Layer)
Person (Dynamic)
CDC
Stream
Person Address (Dynamic)
CDC
Stream
(Gold Layer)
Get
Proc
Merge
Get
Proc
Merge
Person (Dimension)
Sample Module
(Silver Layer)
Sample
CDC
Stream
(Gold Layer)
Get
Proc
Merge
Proc
Snapshot, etc.
Sample (Fact)