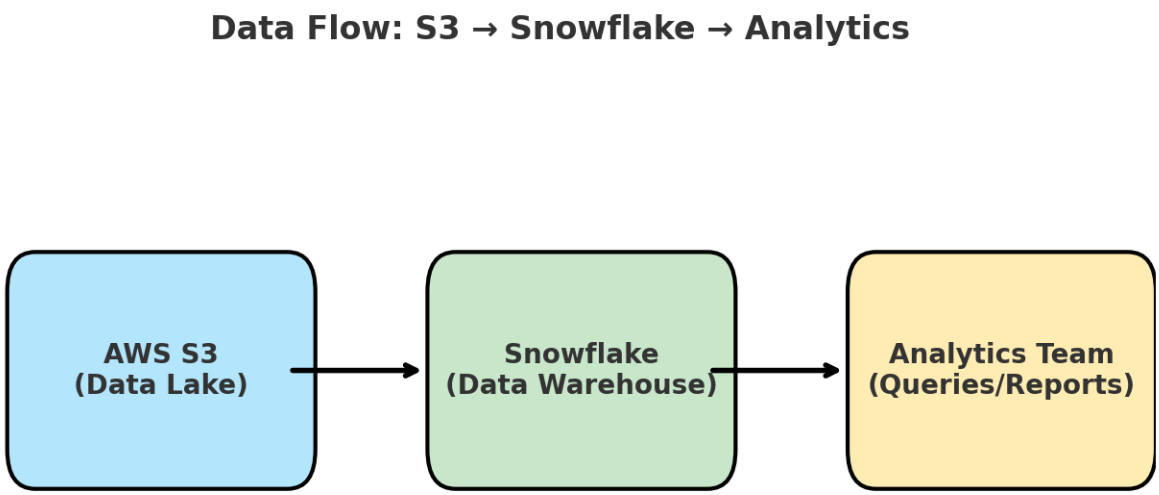
Dimensional Modeling Report – Instamart (Snowflake + S3)

Introduction

Instacart is a grocery ordering and delivery app that helps customers conveniently stock their refrigerators and pantries with their favourite items and daily essentials. After customers place orders through the Instacart app, personal shoppers pick the products in-store and deliver them directly to the customer's doorstep.

To support the Instamart analytics team, we need to design a dimensional model for a data warehouse. This will allow the team to run their analytics efficiently, ensuring smooth performance and faster query execution. An optimized schema is essential to provide insights at minimal execution time.

For this project, we will be leveraging **Snowflake** as our cloud data warehouse solution, with raw data stored in an **AWS S3 data lake**. This setup will enable scalable storage, seamless integration, and high-performance analytics.



1. Business Requirements

Using this business needs answers to the following questions from analytics team and to design a Tableau dashboard if needed to answer the following questions

Business Questions for Instamart Dimensional Model

Customer Behaviour & Ordering Patterns

- What are the most frequently reordered products?
- How often do customers place repeat orders (days between orders)?
- At what time of day and day of week do customers place the most orders?
- What is the average number of items in a customer's cart per order?

Product & Category Insights

- Which products are most commonly added to carts first?
- Which **aisles** and **departments** generate the highest order volumes?
- What are the top-selling products by number of orders?
- Are certain product categories more likely to be reordered?

Operational & Inventory Questions

- How many products are ordered per department/aisle in a given week or month?
- What percentage of products in each department are reordered at least once?
- Which aisles contribute most to overall sales volume?

Trend & Performance Tracking

- What is the trend of total orders over time (daily, weekly, monthly)?
- How does order frequency vary by season, day, or hour?
- What percentage of orders contain products from multiple departments?

Key Metrics (Facts)

The fact table will mainly be built from orders and order_products since they capture measurable events.

- Number of Orders → Count of order_id
- Number of Products per Order → Count of product_id grouped by order_id
- Cart Position → From add_to_cart_order (sequence of product placement in cart)
- Reorder Rate → Based on reordered flag (0 = new order, 1 = reordered)
- Average Days Between Orders → From days_since_prior_order
- Orders by Time/Day → Derived from order_dow and order_hour_of_day

(Note: Since no pricing/revenue data exists in these tables, facts are focused on order frequency, product mix, and reorders.)

3. Key Categories (Dimensions)

Customer Dimension (derived from orders)

- user_id → Unique customer
- order_number → Sequence of orders per customer
- Behavior attributes can be derived (e.g., frequency, reorder tendency).

Product Dimension *(from products, linked to aisles & departments)*

- product_id
- product_name
- aisle_id → joined with Aisles table → aisle (e.g., beverages, dairy)
- department_id → joined with Departments table → department (e.g., produce, frozen)

Time Dimension *(from orders)*

- order_dow → Day of week (0 = Sunday, 6 = Saturday)
- order_hour_of_day → Hour product was ordered (0–23)
- Derived hierarchy: Day → Week → Month → Year

Order Dimension *(from orders & order_products)*

- order_id
- eval_set (train, test, prior – depending on dataset split)
- add_to_cart_order (cart sequence)
- reordered (yes/no)

Aisle Dimension

- aisle_id
- aisle (e.g., snacks, dairy, frozen foods)

Department Dimension

- department_id
- department (e.g., produce, household, beverages)

Fact Tables & Measures

1) FactOrderItems

Grain: one record per (order_id, product_id)

Keys

- order_id → (joins to Order/Time/Customer dimensions)
- product_id → (joins to Product, Aisle, Department dims)

Measures

- item_count (always 1 per row; useful for summing quantity)
- add_to_cart_order (sequence position)
- reordered (0/1; supports reorder rate)

Degenerate attributes (optional)

- None required (kept in Order dim/degenerate order)

Common derived KPIs

- Reorder rate = avg(reordered)
- Average cart position = avg(add_to_cart_order)
- Items per order = sum(item_count) by order_id

2. Dimension Tables

ProductDim

From **products** (enriched with aisle/department names)

- Keys: product_id (surrogate product_key recommended)
- Attributes: product_name, aisle_id, aisle_name, department_id, department_name
- Hierarchies: **Department → Aisle → Product**

AisleDim (optional if not denormalizing into ProductDim)

- Keys: aisle_id (surrogate aisle_key)
- Attributes: aisle

DepartmentDim (optional if not denormalizing into ProductDim)

- Keys: department_id (surrogate department_key)
- Attributes: department

In a star schema you can **denormalize** aisle/department into ProductDim to keep joins simple; keep separate dims only if they're reused or have their own attributes.

TimeDim

Derived from order timestamps

- Keys: date_key (YYYYMMDD), optionally time_key (hour)
- Attributes: date, day_name, order_dow, week, month, quarter, year, order_hour_of_day, is_weekend

CustomerDim *(if/when you have user attributes)*

- Keys: user_id (surrogate customer_key)
- Attributes: e.g., signup_date, segment (if available later)
- Until you have user profile data, keep it minimal: just user_id

OrderDegenerateDim (very light)

- Degenerate attributes that don't merit a full dimension:
  - order_id (often treated as a **degenerate dimension** on facts)
  - order_number, eval_set (if present)

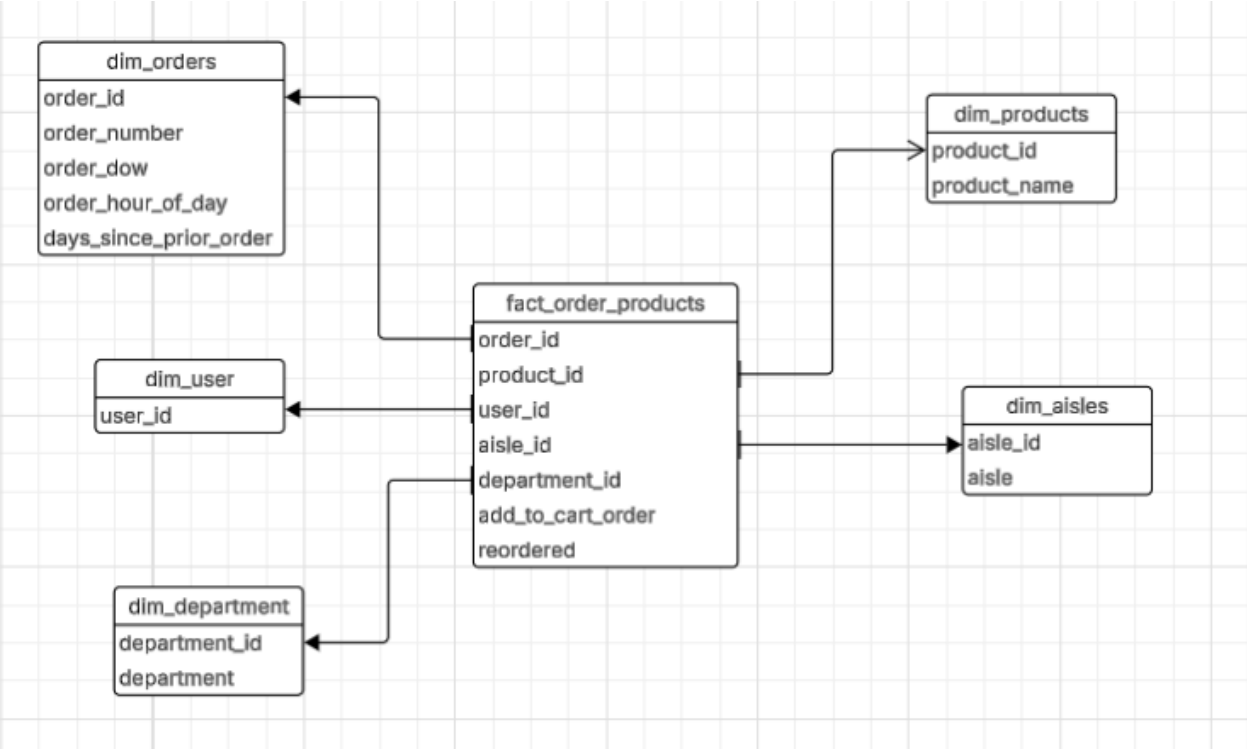
Notes for Snowflake Implementation

- Staging: Load CSVs from S3 stage → Snowflake staging tables.
- ELT: Build dims first (ProductDim; optionally Aisle/Department dims; TimeDim), then FactOrderItems, then FactOrders.

- Surrogate keys: Use IDENTITY or SEQUENCE for dimension *_key columns; retain natural keys (product_id, etc.) for lineage.
- Performance: Consider clustering on (order_date_key, product_key) or (order_date_key, department_name) for heavy time/category queries.
- SCDs: If product/category attributes change, start with Type 1 (overwrite) and evolve to Type 2 if history is needed.

3. Schema Diagram

Star Schema:



We are using a Star Schema for the Instamart data warehouse because it is the most effective design for analytical workloads such as reporting, dashboards, and ad-hoc queries.

1. **Query Performance**

- Star schemas reduce the number of joins compared to normalized schemas.
- Facts are stored in a central Fact table, while descriptive attributes are in smaller Dimension tables.
- This structure allows Snowflake's query engine to scan less data, improving speed for common analytical queries (e.g., sales by aisle, orders by hour).

2. **Simplicity & Usability**

- Easy for analysts and business users to understand: one central fact connected to multiple descriptive dimensions.
- Intuitive structure: “What happened?” (facts) vs. “How/Who/When/Where?” (dimensions).

3. **Flexibility in Analysis**
    - Supports slice-and-dice analysis across different dimensions (e.g., orders by product, by department, by time).
    - Enables drill-down hierarchies (e.g., Department → Aisle → Product, or Day → Month → Year).
4. **Scalability with Snowflake**
    - Star schemas are efficient in columnar cloud warehouses like Snowflake, which handle large fact tables with billions of rows.
    - Dimensions are relatively small and can be cached, minimizing repeated scans.
5. **Extensibility**
    - Easy to add new dimensions (e.g., Customer profile, Delivery performance) or new facts (e.g., Payment transactions) without redesigning the entire schema.

Conclusion

The dimensional model for Instamart is designed using a **Star Schema** with a central fact table (orders and order line items) surrounded by descriptive dimensions (products, time, customer, aisle, department, and order details). This design ensures that the Instamart analytics team can perform queries quickly, explore data from multiple perspectives, and generate actionable insights with minimal execution time.

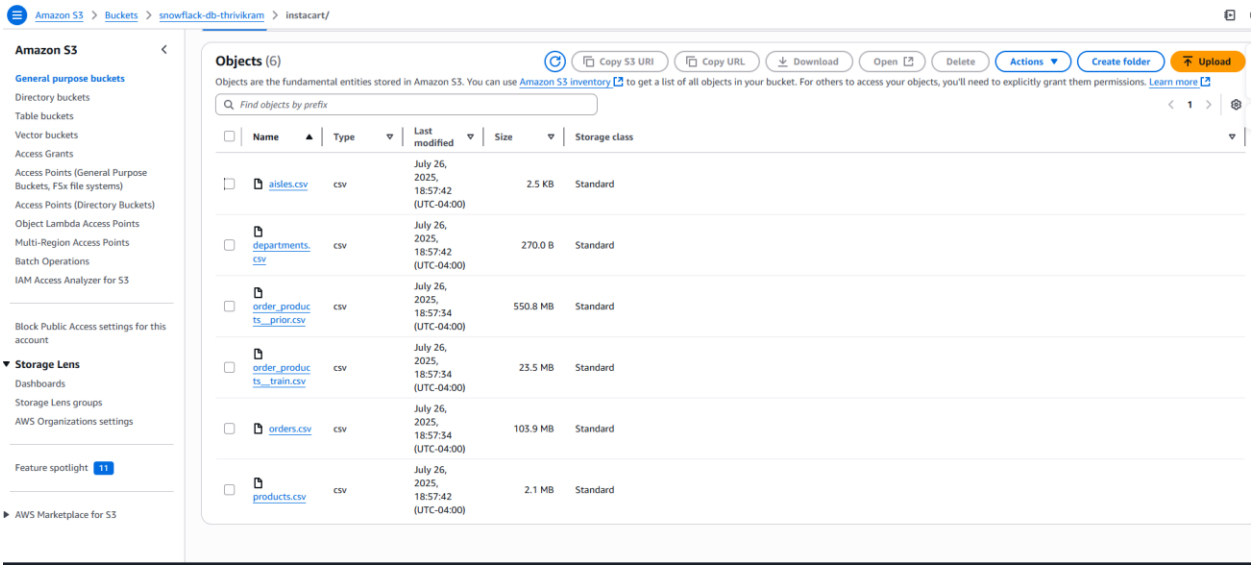
By leveraging **Snowflake** as the data warehouse and **AWS S3** as the data lake, we achieve a scalable, cloud-native solution that supports both large data volumes and flexible analytics. The star schema structure also provides simplicity, extensibility, and strong performance — making it well-suited for ongoing reporting and research needs.

The following **Evidence & Appendix** section will include screenshots of the implementation, SQL scripts for building fact and dimension tables, and example analytical queries to demonstrate the effectiveness of this model.

Evidence & SQL Appendix (Process Walkthrough)

This section documents the step-by-step process we followed to build the dimensional model in Snowflake using data staged in AWS S3. Each step includes an explanation, the SQL code used, and placeholders for screenshots (SS) as evidence.

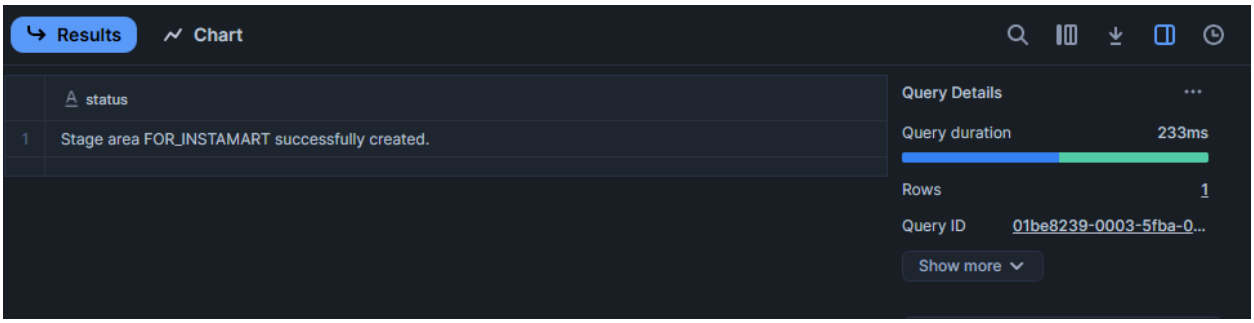
Below is a SS of files in S3



Step 1: Stage Creation

We first created a Snowflake stage pointing to our S3 bucket, which serves as the raw data lake.

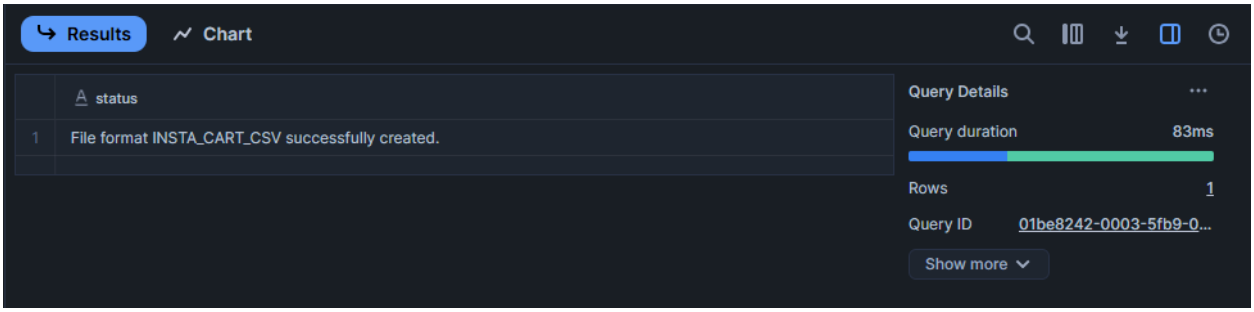
```
CREATE OR REPLACE STAGE for_instamart
  URL = 's3://snowflack-db-thrivikram/instacart/'
  CREDENTIALS = (
    AWS_KEY_ID = '<REDACTED>',
    AWS_SECRET_KEY = '<REDACTED>'
  );
```



Step 2: File Format Definition

Defined a CSV file format to consistently parse incoming files.

```
CREATE OR REPLACE FILE FORMAT insta_cart_csv
  TYPE = 'CSV'
  FIELD_DELIMITER = ','
  SKIP_HEADER = 1
  FIELD_OPTIONALLY_ENCLOSED_BY = '"';
```

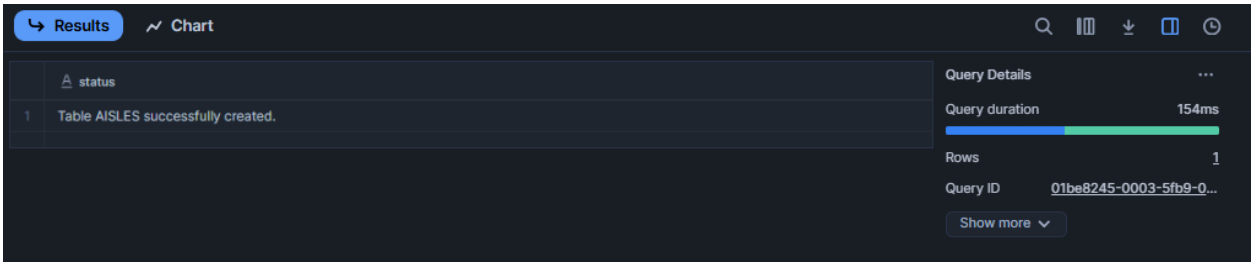


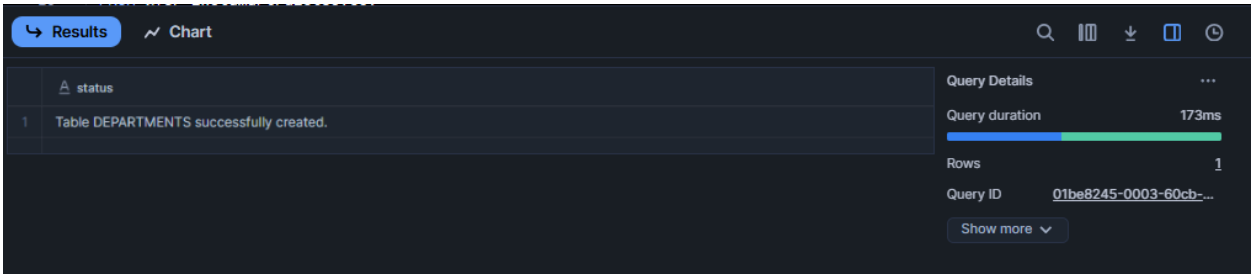
Step 3: Base Table Creation

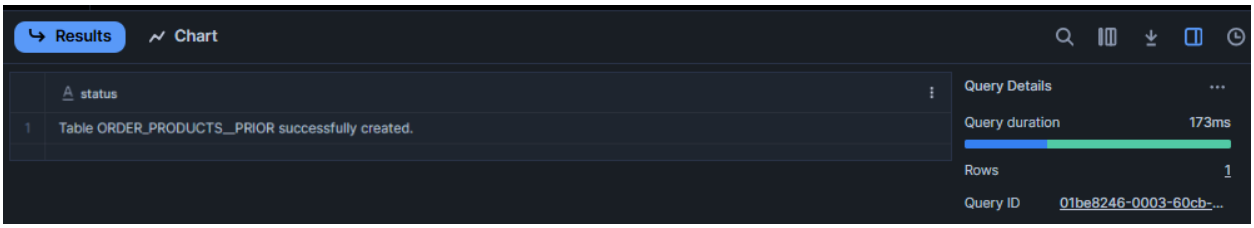
Created raw tables to hold aisles, departments, products, orders, and order-product mappings.

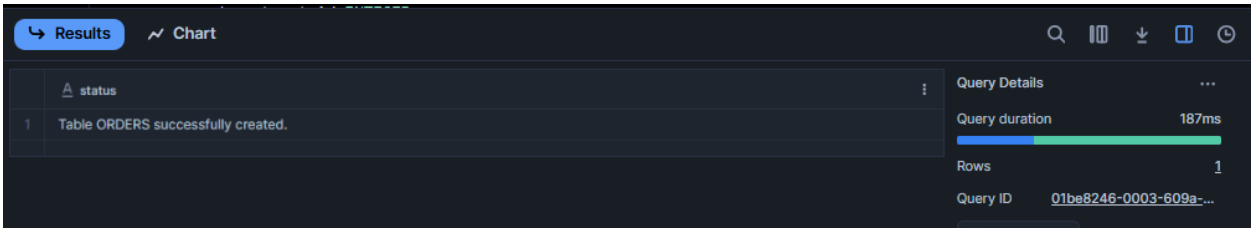
```
CREATE OR REPLACE TABLE aisles (
  aisle_id INTEGER,
  aisle    VARCHAR
);

CREATE OR REPLACE TABLE departments (
  department_id INTEGER,
  department    VARCHAR
);

CREATE OR REPLACE TABLE products (
  product_id    INTEGER,
  product_name  VARCHAR,
  aisle_id      INTEGER,
  department_id INTEGER
);

CREATE OR REPLACE TABLE orders (
  order_id              INTEGER,
  user_id               INTEGER,
  eval_set              VARCHAR,
  order_number          INTEGER,
  order_dow             INTEGER,
  order_hour_of_day     INTEGER,
  days_since_prior_order INTEGER
);

CREATE OR REPLACE TABLE order_products__prior (
  order_id          INTEGER,
  product_id        INTEGER,
  add_to_cart_order INTEGER,
  reordered         INTEGER
);
```

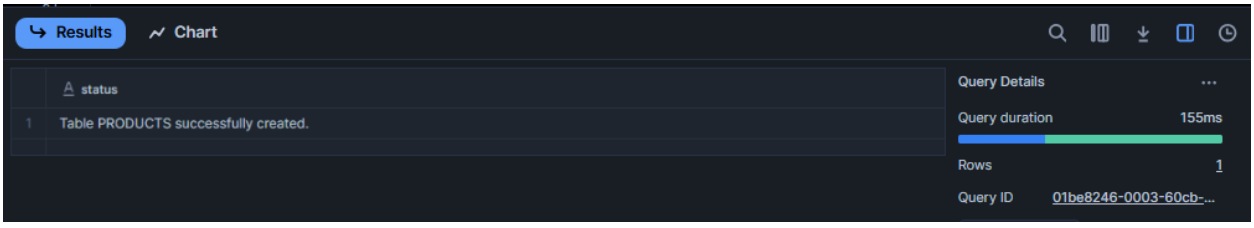
Results:


Results
Chart
status
1
Table AISLES successfully created.
Query Details
Query duration
154ms
Rows
1
Query ID
01be8245-0003-5fb9-0...
Show more


Results
Chart
status
1
Table DEPARTMENTS successfully created.
Query Details
Query duration
173ms
Rows
1
Query ID
01be8245-0003-60cb-...
Show more


Results
Chart
status
1
Table ORDER_PRODUCTS__PRIOR successfully created.
Query Details
Query duration
173ms
Rows
1
Query ID
01be8246-0003-60cb-...


Results
Chart
status
1
Table ORDERS successfully created.
Query Details
Query duration
187ms
Rows
1
Query ID
01be8246-0003-609a-...

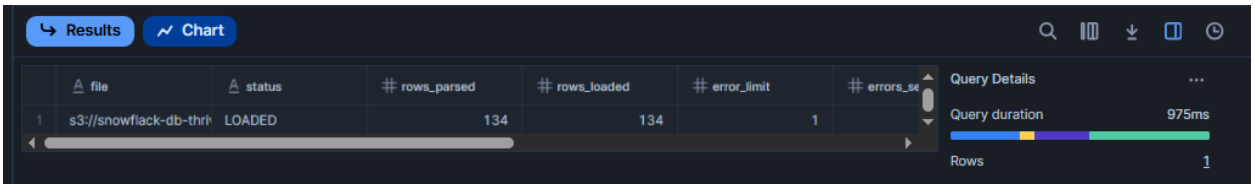

Results
Chart
status
1
Table PRODUCTS successfully created.
Query Details
Query duration
155ms
Rows
1
Query ID
01be8246-0003-60cb-...

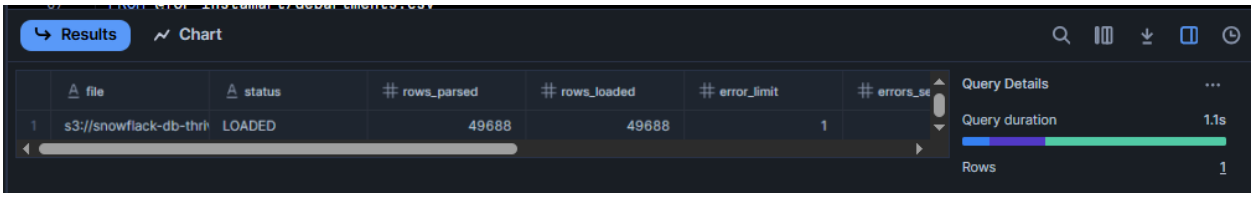
Step 4: Loading Data from S3

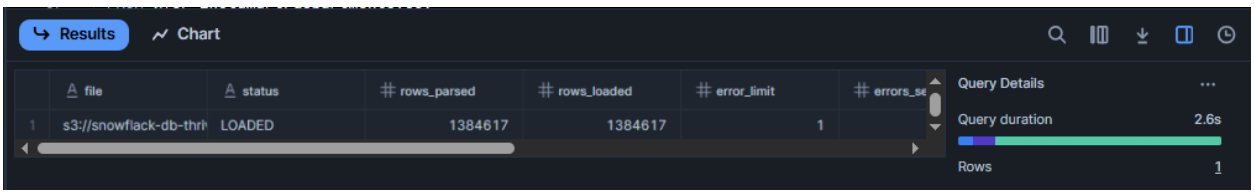
We loaded CSV files from S3 into Snowflake tables using COPY INTO.

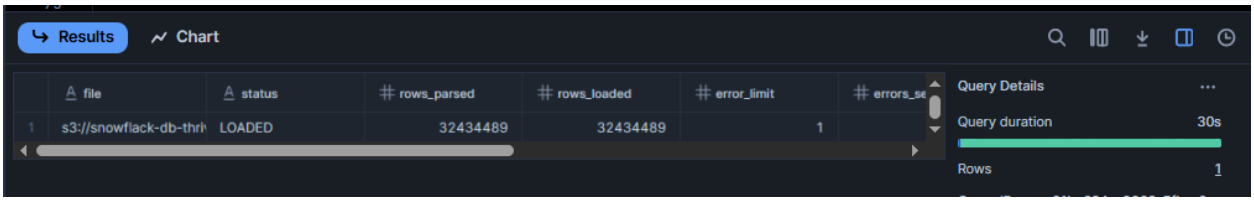
```
COPY INTO aisles (aisle_id, aisle)
FROM @for_instamart/aisles.csv
FILE_FORMAT = (FORMAT_NAME = 'insta_cart_csv');

COPY INTO departments
FROM @for_instamart/departments.csv
FILE_FORMAT = (FORMAT_NAME = 'insta_cart_csv');

COPY INTO products (product_id, product_name, aisle_id, department_id)
FROM @for_instamart/products.csv
FILE_FORMAT = (FORMAT_NAME = 'insta_cart_csv');

COPY INTO orders
FROM @for_instamart/orders.csv
FILE_FORMAT = (FORMAT_NAME = 'insta_cart_csv');

COPY INTO order_products__prior
FROM @for_instamart/order_products__prior.csv
FILE_FORMAT = (FORMAT_NAME = 'insta_cart_csv');
```

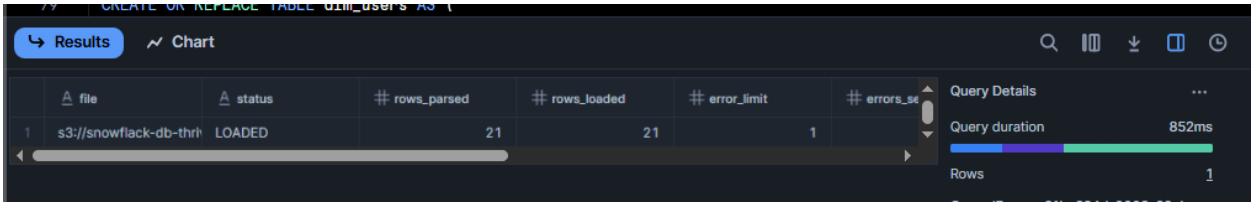
Results:









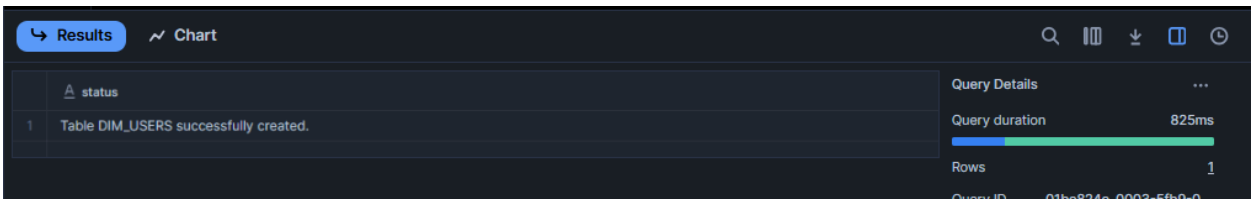


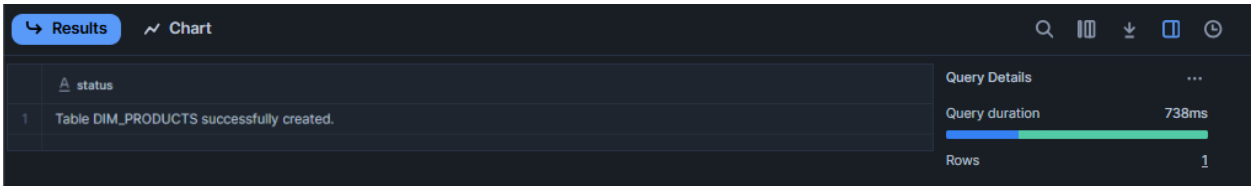
Step 5: Building Dimensions

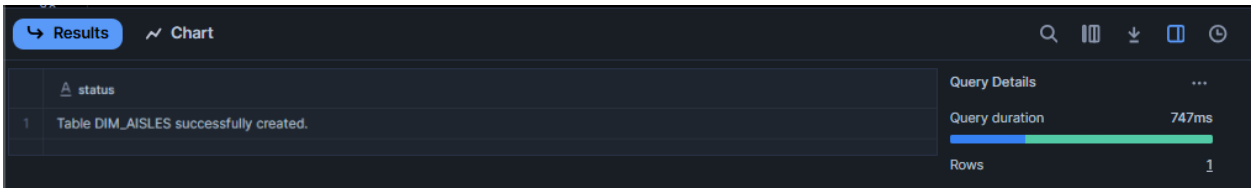
We created dimension tables from the base tables.

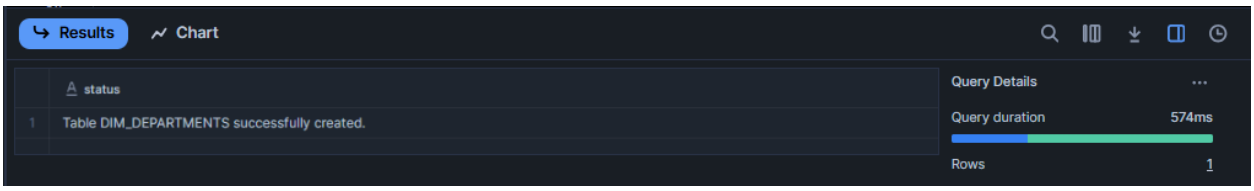
```
CREATE OR REPLACE TABLE dim_users AS (
  SELECT DISTINCT user_id FROM orders
);

CREATE OR REPLACE TABLE dim_products AS (
  SELECT product_id, product_name FROM products
);

CREATE OR REPLACE TABLE dim_aisles AS (
  SELECT aisle_id, aisle FROM aisles
);

CREATE OR REPLACE TABLE dim_departments AS (
  SELECT department_id, department FROM departments
);

CREATE OR REPLACE TABLE dim_orders AS (
  SELECT order_id, order_number, order_dow, order_hour_of_day, days_since_prior_order
  FROM orders
);
```

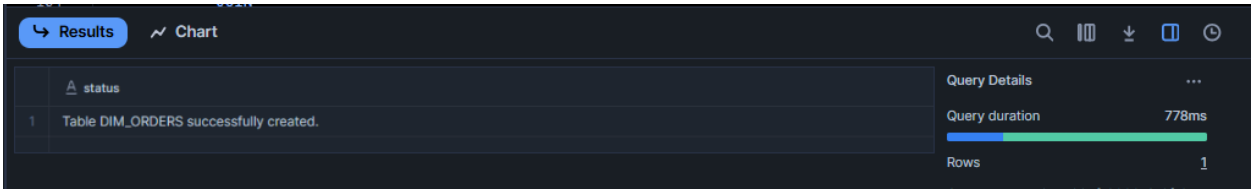
Results:







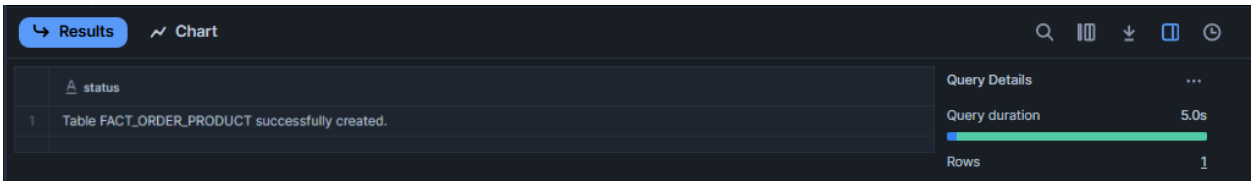




Step 6: Building the Fact Table

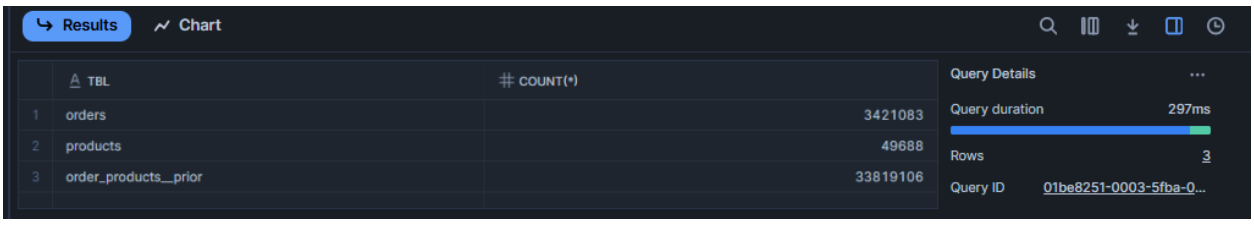
We joined orders, products, and order-product mappings to construct the fact table.

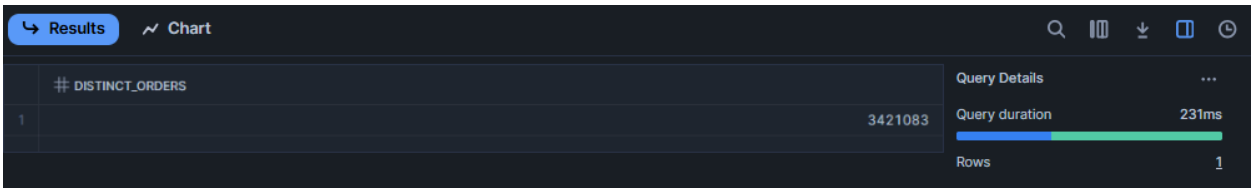
```
CREATE OR REPLACE TABLE fact_order_product AS (
  SELECT
    op.order_id,
    op.product_id,
    o.user_id,
    p.department_id,
    p.aisle_id,
    op.add_to_cart_order,
    op.reordered
  FROM order_products__prior op
  JOIN orders   o ON op.order_id  = o.order_id
  JOIN products p ON op.product_id = p.product_id
);
```

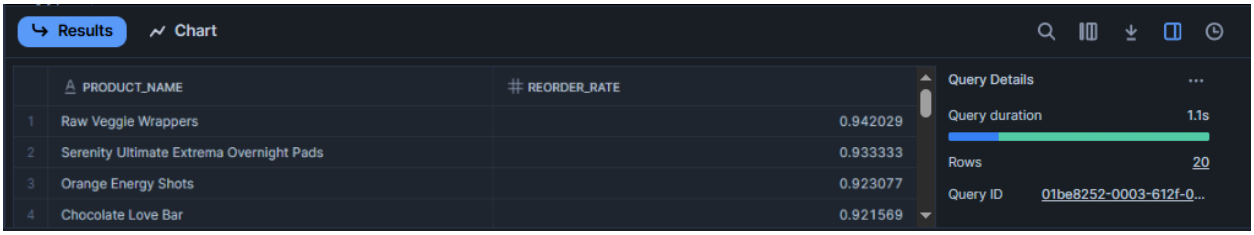


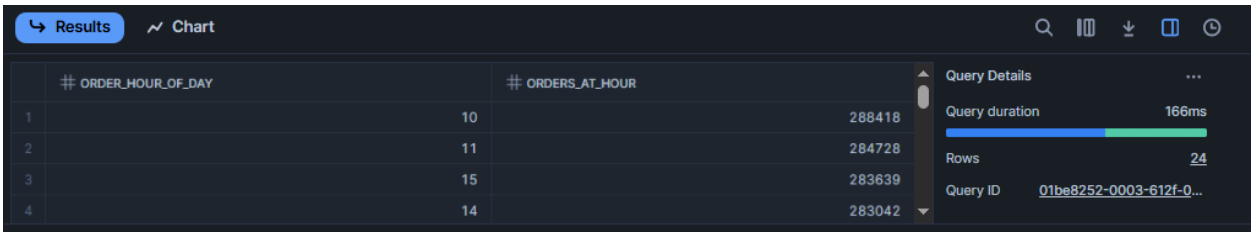
Step 7: QA Checks & Sample Analytics

```sql
-- Count checks
SELECT 'orders' tbl, COUNT(*) FROM orders
UNION ALL SELECT 'products', COUNT(*) FROM products
UNION ALL SELECT 'order_products__prior', COUNT(*) FROM order_products__prior;

-- Distinct keys
SELECT COUNT(DISTINCT product_id) AS distinct_products FROM products;
SELECT COUNT(DISTINCT order_id)   AS distinct_orders   FROM orders;

-- Example: Reorder rate by product
SELECT p.product_name, AVG(op.reordered) AS reorder_rate
FROM fact_order_product op
JOIN dim_products p USING (product_id)
GROUP BY 1
ORDER BY 2 DESC
LIMIT 20;

-- Example: Peak ordering hour
SELECT d.order_hour_of_day, COUNT(*) AS orders_at_hour
FROM dim_orders d
GROUP BY 1
ORDER BY 2 DESC;
```









References

- Kimball & Ross, The Data Warehouse Toolkit
- Snowflake Docs: Loading data from S3, Clustering, SCD patterns