

Project Overview

Objective:

Create an end-to-end analytics pipeline for an ecommerce platform to track sales, customers, products, and performance metrics.

Scope:

- Generate synthetic data for orders, products, customers, payments, shipping, and categories (>1 million records).
- Store data in Amazon S3.
- Extract data into Snowflake using Airbyte.
- Transform and model data using dbt.
- Perform analytics in Snowflake.
- Connect Snowflake marts to Power BI for professional dashboards.

Tools & Technologies:

Layer	Tool/Technology
Data Generation	Python
Data Storage	Amazon S3
Data Integration	Airbyte
Data Warehouse	Snowflake
Data Modeling & Transform	dbt
BI & Visualization	Power BI
Programming / Scripting	Python / SQL / DAX

1- Data Generation

```
17
18 # Initialize S3 client
19 s3 = boto3.client(
20     "s3",
21     aws_access_key_id=AWS_ACCESS_KEY_ID,
22     aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
23     region_name=AWS_REGION
24 )
25
26 # -----
27 # DATASET SIZES
28 # -----
29 n_customers = 100_000
30 n_orders = 200_000
31 n_products = 5_000
32 n_regions = 20
33 n_categories = 50
34
35
36 def upload_df_to_s3(df, filename):
37     """
38     Convert DataFrame → Parquet → Upload to S3 (in-memory)
39     """
40     table = pa.Table.from_pandas(df)
41
42     buffer = BytesIO()
43     pq.write_table(table, buffer)
44     buffer.seek(0)
45
46     s3_path = f"{S3_PREFIX}{filename}.parquet"
47
48     s3.upload_fileobj(buffer, BUCKET_NAME, s3_path)
49
50     print(f"✓ Uploaded {filename}.parquet to s3://{BUCKET_NAME}/{s3_path}")
51
52
53
54 # GENERATE SYNTHETIC TABLES
55 # -----
56 regions = pd.DataFrame({
57     "region_id": range(1, n_regions + 1),
58     "region_name": [f"Region_{i}" for i in range(1, n_regions + 1)]
59 })
60
61 categories = pd.DataFrame({
62     "category_id": range(1, n_categories + 1),
63     "category_name": [f"Category_{i}" for i in range(1, n_categories + 1)]
64 })
65
66 products = pd.DataFrame({
67     "product_id": range(1, n_products + 1),
68     "product_name": [f"Product_{i}" for i in range(1, n_products + 1)],
69     "category_id": np.random.randint(1, n_categories + 1, n_products),
70     "price": np.round(np.random.uniform(5, 2000, n_products), 2)
71 })
72
73 customers = pd.DataFrame({
74     "customer_id": range(1, n_customers + 1),
75     "first_name": [f"Name_{i}" for i in range(1, n_customers + 1)],
76     "last_name": [f"Last_{i}" for i in range(1, n_customers + 1)],
77     "email": [f"user{i}@example.com" for i in range(1, n_customers + 1)],
78     "region_id": np.random.randint(1, n_regions + 1, n_customers),
79     "created_at": pd.to_datetime("2023-01-01") +
80         pd.to_timedelta(np.random.randint(0, 400, n_customers), "D")
81 })
82
83 orders = pd.DataFrame({
84     "order_id": range(1, n_orders + 1),
85     "customer_id": np.random.randint(1, n_customers + 1, n_orders),
86     "order_date": pd.to_datetime("2023-01-01") +
87         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
88     "status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders),
89     "total_amount": np.round(np.random.uniform(10, 3000, n_orders), 2)
90 })
```

Sales Snowflake > datagen.py > ...

```
80     pd.to_timedelta(np.random.randint(0, 400, n_customers), "D")
81 })
82
83 orders = pd.DataFrame({
84     "order_id": range(1, n_orders + 1),
85     "customer_id": np.random.randint(1, n_customers + 1, n_orders),
86     "order_date": pd.to_datetime("2023-01-01") +
87         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
88     "status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders),
89     "total_amount": np.round(np.random.uniform(10, 3000, n_orders), 2)
90 })
91
92 order_items = pd.DataFrame({
93     "order_item_id": range(1, n_orders * 2 + 1),
94     "order_id": np.random.randint(1, n_orders + 1, n_orders * 2),
95     "product_id": np.random.randint(1, n_products + 1, n_orders * 2),
96     "quantity": np.random.randint(1, 5, n_orders * 2)
97 })
98
99 payments = pd.DataFrame({
100     "payment_id": range(1, n_orders + 1),
101     "order_id": range(1, n_orders + 1),
102     "payment_method": np.random.choice(["Card", "PayPal", "Cash"], n_orders),
103     "amount": np.round(np.random.uniform(10, 3000, n_orders), 2),
104     "payment_date": pd.to_datetime("2023-01-01") +
105         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D")
106 })
107
108 shipping = pd.DataFrame({
109     "shipping_id": range(1, n_orders + 1),
110     "order_id": range(1, n_orders + 1),
111     "shipped_date": pd.to_datetime("2023-01-01") +
112         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
113     "delivery_date": pd.to_datetime("2023-01-01") +
114         pd.to_timedelta(np.random.randint(0, 400, n_orders), "D"),
115     "shipping_status": np.random.choice(["Pending", "Shipped", "Delivered"], n_orders)
116 })
---
```

```
119 # UPLOAD TABLES TO S3
120 # -----
121 upload_df_to_s3(regions, "raw_regions")
122 upload_df_to_s3(categories, "raw_categories")
123 upload_df_to_s3(products, "raw_products")
124 upload_df_to_s3(customers, "raw_customers")
125 upload_df_to_s3(orders, "raw_orders")
126 upload_df_to_s3(order_items, "raw_order_items")
127 upload_df_to_s3(payments, "raw_payments")
128 upload_df_to_s3(shipping, "raw_shipping")
129
130 print("\n🎉 All Parquet files uploaded to S3 successfully!")
131
```

Tables Generated:

- **Orders** (order_id, customer_id, order_date, total_amount, status)
- **Order Items** (order_item_id, order_id, product_id, quantity, price)
- **Products** (product_id, product_name, category_id, price)
- **Customers** (customer_id, first_name, last_name, email, region_id)
- **Categories** (category_id, category_name)
- **Payments** (payment_id, order_id, amount, payment_date, payment_method)
- **Shipping** (shipping_id, order_id, shipped_date, delivery_date, shipping_status)
- **Regions** (region_id, region_name)

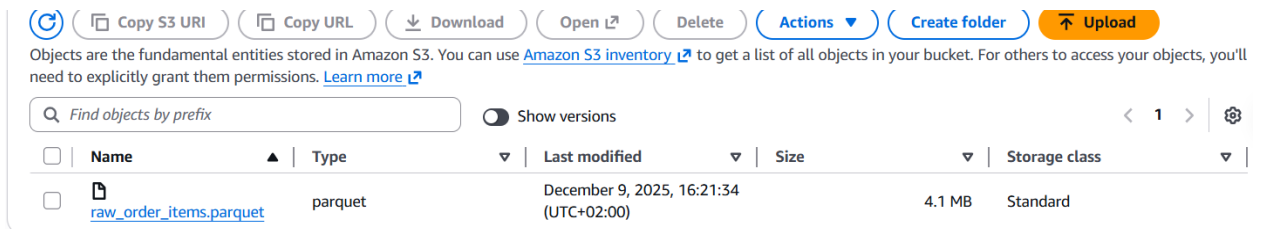
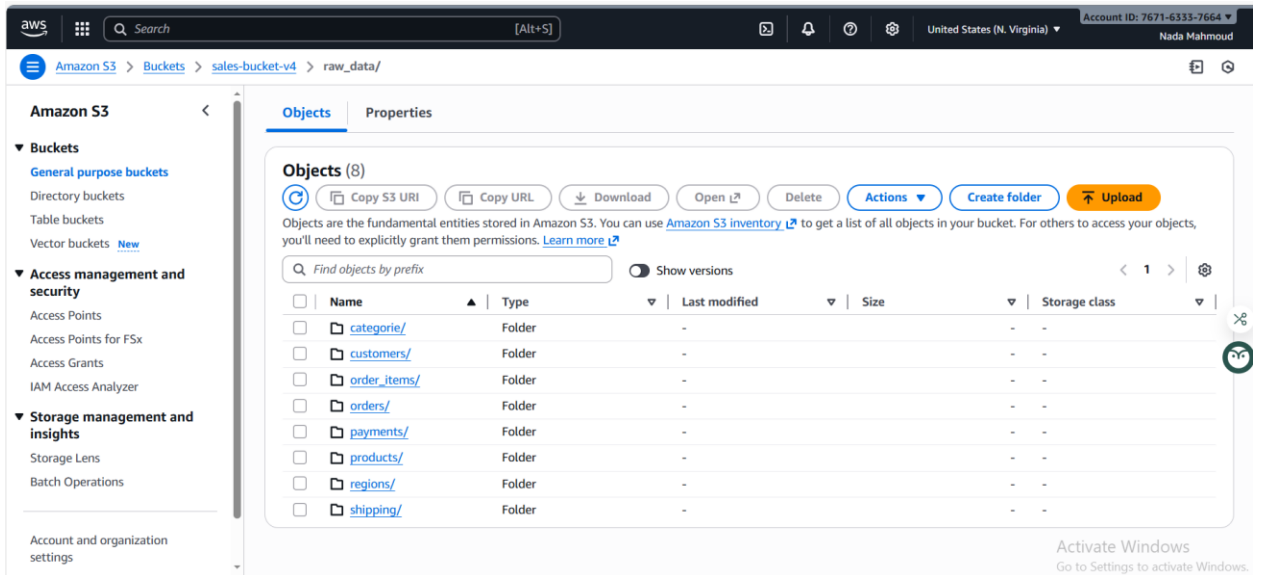
2- Data Storage

The screenshot displays the AWS IAM console interface for a user named 'python_sends_s3'. The left-hand navigation pane shows the 'Identity and Access Management (IAM)' section, with 'Users' selected. The main content area is titled 'python_sends_s3' and includes a 'Delete' button in the top right corner. Below the title, there is a 'Summary' section with the following details:

- ARN:** `arn:aws:iam::767163337664:user/python_sends_s3`
- Console access:** Disabled
- Created:** December 09, 2025, 15:53 (UTC+02:00)
- Last console sign-in:** -
- Access key 1:** AKIA3FHT7RPAIXPNOJM3 - Active
Used today. Created today.
- Access key 2:** Create access key

Below the summary, there are tabs for 'Permissions', 'Groups', 'Tags (1)', 'Security credentials', and 'Last Accessed'. The 'Permissions' tab is currently active, showing 'Permissions policies (1)'. A note states: 'Permissions are defined by policies attached to the user directly or through groups.' Below this, there is a search bar and a 'Filter by Type' dropdown set to 'All types'. A table lists the attached policies:

<input type="checkbox"/>	Policy name ↗	Type	Attached via ↗
<input type="checkbox"/>	PythonGenS3	Customer managed	Directly



s3://ecommerce-data/

- └─ orders/
- └─ order_items/
- └─ customers/
- └─ products/
- └─ categories/
- └─ payments/
- └─ shipping/
- └─ regions/

Format: Parquet

3- Data Extraction & Loading

Tool: Airbyte

The screenshot displays the AWS IAM console interface, divided into two main sections. The top section shows the details for a user named 'Airbyte-read-s3'. The bottom section shows the details for a policy named 'AirbyteS3Read'.

Top Section: User Details (Airbyte-read-s3)

- Summary:**
 - ARN: `arn:aws:iam::767163337664:user/Airbyte-read-s3`
 - Console access: Disabled
 - Created: December 09, 2025, 16:34 (UTC+02:00)
 - Last console sign-in: -
 - Access key 1: AKIA3FHT7RPAGN37GEJC - Active (Never used. Created today.)
 - Access key 2: [Create access key](#)
- Permissions policies (1):**
 - Permissions are defined by policies attached to the user directly or through groups.
 - Filter by Type: All types
 - Search:
 - Table with 3 columns: Policy name, Type, Attached via.
 - Row 1: [AirbyteS3Read](#), Customer managed, Directly

Bottom Section: Policy Details (AirbyteS3Read)

- Summary:**
 - Type: Customer managed
 - Creation time: December 09, 2025, 16:33 (UTC+02:00)
 - Edited time: December 09, 2025, 16:33 (UTC+02:00)
 - ARN: `arn:aws:iam::767163337664:policy/AirbyteS3Read`
- Permissions defined in this policy:**
 - Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.
 - JSON view:

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "s3:ListBucket",
8         "s3:GetObject",
9         "s3:GetBucketLocation"
10      ],
11       "Resource": [
12         "arn:aws:s3:::sales-bucket-v4",
13         "arn:aws:s3:::sales-bucket-v4/*"
14      ]
15     }
16   ]
17 }
```

Footer:

- S3-sales-parquet1
- S3
- 2 connections
- 58 minutes ago
- 1 error, 1 success



S3-sales-parquet1

S3 v4.15.2

[Fork in Builder](#)

Settings

Connections

Source Settings

Source name ⓘ

S3-sales-parquet1

Bucket ⓘ

sales-bucket-v4

The list of streams to sync ⓘ

Add

- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/categorie/raw_categories.par..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/customers/raw_customers.p..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/order_items/raw_order_items..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/orders/raw_orders.parquet"]... ×

Activate Window
Go to Settings to acti

The list of streams to sync ⓘ

Add

- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/categorie/raw_categories.par..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/customers/raw_customers.p..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/order_items/raw_order_items..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/orders/raw_orders.parquet"]... ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/payments/raw_payments.par..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/products/raw_products.parq..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/regions/raw_regions.parquet..." ×
- > Days To Sync If History Is Full: 3 | Format: Parquet Format | Globs: ["raw_data/shipping/raw_shipping.parqu..." ×

Optional fields

Start Date ⓘ Optional YYYY-MM-DDTHH:mm:ss.SSSSSSZ

AWS Access Key ID ⓘ Optional

.....

Edit

AWS Secret Access Key ⓘ Optional

.....

Edit

Endpoint ⓘ Optional

AWS Region ⓘ Optional

us-east-1

From s3 load to snowflake

Snowflake

Snowflake

1 connection

1 hour ago

1

Snowflake

Snowflake v4.0.31

SettingsConnections

Destination Settings

Destination name ⓘ

Snowflake

Connection

Host ⓘ

(account_name).snowflakecomputing.com or (accountname).(aws_location).aws.snowflakecomputing.com

cckybwk-yc47376.snowflakecomputing.com

Role ⓘ

ACCOUNTADMIN

Warehouse ⓘ

ecommerce_wh

Activate Windows

Go to Settings to activate \

Warehouse ⓘ

ecommerce_wh

Database ⓘ

ecommerce_db

Default Schema ⓘ

raw

Username ⓘ

dbt_user

Authorization Method ⓘ

Username and Password ▾

Password ⓘ

••••••••••

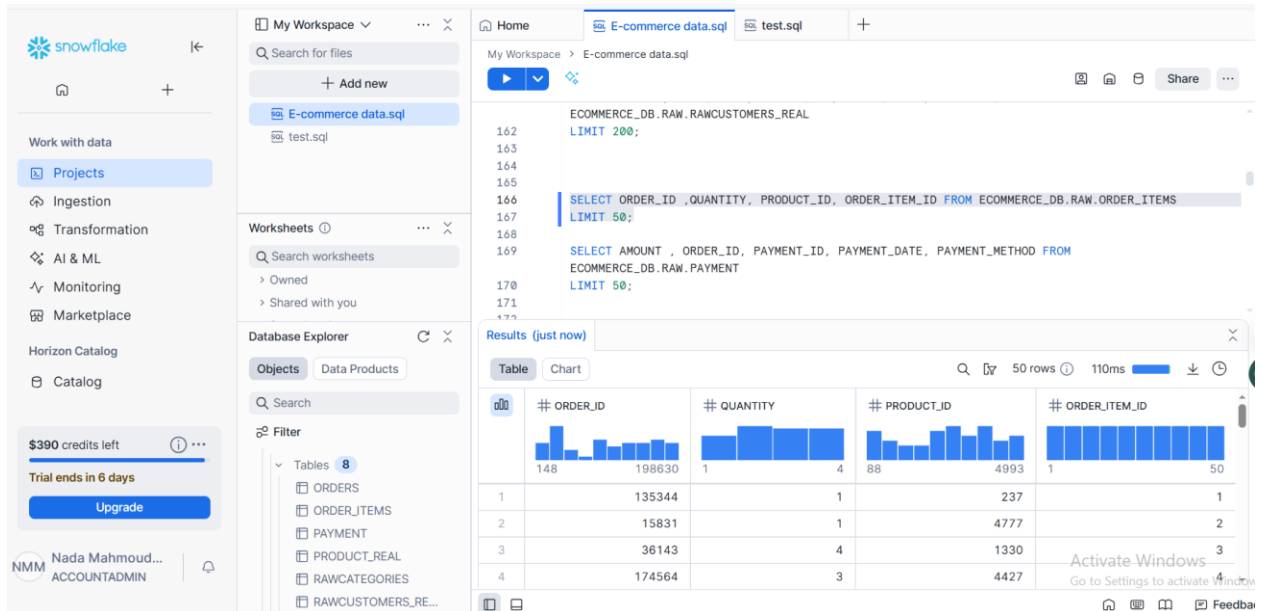
Edit

Active Streams

🔍 Search

Status	Stream name	Latest sync in <div>records ▾</div>	Data fresh as of	
✔ Synced	categorie	50 loaded	27 minutes ago	⋮
✔ Synced	Customers	100,000 loaded	27 minutes ago	⋮
✔ Synced	order_items	400,000 loaded	27 minutes ago	⋮
✔ Synced	orders	200,000 loaded	27 minutes ago	⋮
✔ Synced	payment	200,000 loaded	27 minutes ago	⋮
✔ Synced	products	5,000 loaded	27 minutes ago	⋮
✔ Synced	regions	20 loaded	27 minutes ago	⋮

Activate Windows
Go to Settings to activate Windows



4- Data Transformation & Modeling

Tool: dbt (Data Build Tool)

Purpose: Transform RAW data into **analytics-ready marts**.

```
PS F:\Sales Snowflake> dbt init ecommerce_project
22:45:38 Running with dbt=1.10.13
22:45:38
Your new dbt project "ecommerce_project" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

    https://docs.getdbt.com/docs/configure-your-profile

One more thing:

Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:

Activate Windows
Go to Settings to activate Windows.
```

```
PS F:\Sales Snowflake> dbt init ecommerce_project
22:45:38 Setting up your profile.
The profile ecommerce_project already exists in C:\Users\DeLL\dbt\profiles.yml. Continue and overwrite it? [y/N]: y
Which database would you like to use?
[1] postgres
[2] snowflake

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 2
account (https://<this_value>.snowflakecomputing.com): ccykbwk-yc47376
[1] password
[2] keypair
[3] sso

Activate Windows
Go to Settings to activate Windows.
```

```
[3] sso
Desired authentication type option (enter a number): 1
password (dev password):
role (dev role): ACCOUNTADMIN
warehouse (warehouse name): ecommerce_wh
database (default database that dbt will build objects in): ECOMMERCE_DB
schema (default schema that dbt will build objects in): RAW
threads (1 or more) [1]: 10
22:48:32 Profile ecommerce_project written to C:\Users\DeLL\.dbt\profiles.yml using target's profile_template.yml and your supplied values. Run
PS F:\Sales Snowflake> dbt debug --profiles-dir C:\Users\DeLL\.dbt
22:49:25 Running with dbt=1.10.13
```

```
Sales Snowflake > ecommerce_project > ! dbt_project.yml
1 name: 'ecommerce_project'
2 version: '1.0.0'
3
4 profile: 'ecommerce_project'
5
6 model-paths: ["models"]
7 analysis-paths: ["analyses"]
8 test-paths: ["tests"]
9 seed-paths: ["seeds"]
10 macro-paths: ["macros"]
11 snapshot-paths: ["snapshots"]
12
13 clean-targets:
14   - "target"
15   - "dbt_packages"
16
17 models:
18   ecommerce_project:
19     +materialized: view
20
```

```
C:\Users\DeLL> dbt > ! profiles.yml
1 ecommerce_project:
2   target: dev
3   outputs:
4     dev:
5       type: snowflake
6       account: ccykbwk-yc47376 # Snowflake account
7       user: dbt_user # your Snowflake username
8       password: dbt_password # your Snowflake password
9       role: ACCOUNTADMIN
10      database: ECOMMERCE_DB # default database
11      warehouse: ecommerce_wh # default warehouse
12      schema: RAW # default schema
13      threads: 10
14
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\Sales Snowflake> dbt debug
23:42:23 python version: 3.12.0
23:42:23 python path: F:\dbt project\dbt-env\Scripts\python.exe
23:42:23 os info: Windows-10-10.0.19045-SP0
23:42:24 Using profiles dir at C:\Users\DeLL\.dbt
23:42:24 Using dbt_project.yml file at F:\Sales Snowflake\dbt_project.yml
23:42:24 adapter type: snowflake
23:42:24 adapter version: 1.10.3
23:42:24 Configuration:
23:42:24   profiles.yml file [OK found and valid]
23:42:24   dbt_project.yml file [ERROR not found]
23:42:24 Required dependencies:
23:42:24   - git [OK found]
```

```
23:43:12 Registered adapter: snowflake=1.10.3
23:43:14 Connection test: [OK connection ok]
23:43:14 All checks passed!
```

Transform data

Sales Snowflake > ecommerce_project > models > models > staging > stg_categories.sql

```
1 with raw as (  
2     select  
3         category_id,  
4         category_name  
5     from {{ source('ecommerce_raw', 'RAWCATEGORIES') }}  
6 )  
7 select * from raw  
8
```

Sales Snowflake > ecommerce_project > models > models > staging > stg_customers.sql

```
1 with raw as (  
2     select  
3         customer_id,  
4         first_name,  
5         last_name,  
6         email,  
7         region_id  
8     from {{ source('ecommerce_raw', 'RAWCUSTOMERS_REAL') }}  
9 )  
10 select * from raw  
11  
12  
13  
14
```

Sales Snowflake > ecommerce_project > models > models > staging > stg_order_items.sql

```
1 with raw as (  
2     select  
3         order_item_id,  
4         order_id,  
5         product_id,  
6         quantity::integer as quantity  
7     from {{ source('ecommerce_raw', 'ORDER_ITEMS') }}  
8 )  
9 select * from raw  
10
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_orders.sql
1  -- models/staging/stg_orders.sql
2  with raw as (
3      select
4          order_id,
5          customer_id,
6          total_amount::decimal as total_amount,
7          order_date::timestamp as order_date,
8          status
9      from {{ source('ecommerce_raw', 'ORDERS') }}
10 )
11 select * from raw
12
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_payment.sql
1  with raw as (
2      select
3          payment_id,
4          order_id,
5          amount::decimal as amount,
6          payment_date::timestamp as payment_date,
7          payment_method
8      from {{ source('ecommerce_raw', 'PAYMENT') }}
9  )
10 select * from raw
11
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_products.sql
1  with raw as (
2      select
3          product_id,
4          product_name,
5          category_id,
6          price::decimal as price
7      from {{ source('ecommerce_raw', 'PRODUCT_REAL') }}
8  )
9  select * from raw
10
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_regions.sql
1 with raw as (
2   select
3     region_id,
4     region_name
5   from {{ source('ecommerce_raw', 'REGION_REAL') }}
6 )
7 select * from raw
8
```

```
Sales Snowflake > ecommerce_project > models > models > staging > stg_shipping.sql
1 with raw as (
2   select
3     shipping_id,
4     order_id,
5     shipped_date::timestamp as shipped_date,
6     delivery_date::timestamp as delivery_date,
7     shipping_status
8   from {{ source('ecommerce_raw', 'SHIPPING') }}
9 )
10 select * from raw
11
```

Creating marts facts and dimensions

```
Sales Snowflake > ecommerce_project > models > models > marts > marts > dim_categories.sql
1 select
2   category_id,
3   category_name
4 from {{ ref('stg_categories') }}
5
```

Left sidebar (File Explorer):

- UNTITLED (WORKSPACE)
- DES & BLOWFISH
- Sales Snowflake
 - ecommerce_project
 - analyses
 - .gitkeep
 - logs
 - macros
 - .gitkeep
 - models \ models
 - models \ marts
 - dim_categories.sql
 - dim_customers.sql
 - dim_products.sql
 - dim_regions.sql
 - fact_order_items.sql
 - fact_orders.sql
- OUTLINE
- TIMELINE dim_categories.sql 9 mins
- File Saved

UNTITLED (WORKSPACE) | Sales Snowflake > ecommerce_project > models > models > marts > marts > dim_customers.sql

```
1 select
2   customer_id,
3   email,
4   region_id,
5   first_name || ' ' || last_name as customer_name
6 from {{ ref('stg_customers') }}
```

dim_categories.sql
dim_customers.sql
dim_products.sql
dim_regions.sql
fact_order_items.sql
fact_orders.sql

OUTLINE
dim_customers.sql
File Saved 11 mins

UNTITLED (WORKSPACE) | Sales Snowflake > ecommerce_project > models > models > marts > marts > dim_products.sql

```
1 select
2   product_id,
3   product_name,
4   category_id,
5   price
6 from {{ ref('stg_products') }}
```

dim_categories.sql
dim_customers.sql
dim_products.sql
dim_regions.sql
fact_order_items.sql
fact_orders.sql

OUTLINE
dim_products.sql
File Saved 11 mins

UNTITLED (WORKSPACE) | Sales Snowflake > ecommerce_project > models > models > marts > marts > dim_regions.sql

```
1 select
2   region_id,
3   region_name
4 from {{ ref('stg_regions') }}
```

dim_categories.sql
dim_customers.sql
dim_products.sql
dim_regions.sql
fact_order_items.sql
fact_orders.sql

OUTLINE
dim_regions.sql
File Saved 11 mins

UNTITLED (WORKSPACE) | Sales Snowflake > ecommerce_project > models > models > marts > marts > fact_order_items.sql


```
1 select
2   oi.order_item_id,
3   oi.order_id,
4   oi.product_id,
5   oi.quantity,
6   oi.quantity * pr.price as total_price
7 from {{ ref('stg_order_items') }} oi
8 left join {{ ref('stg_products') }} pr
9   on oi.product_id = pr.product_id
10
```

OUTLINE | fact_order_items.sql | 9 mins


UNTITLED (WORKSPACE) | Sales Snowflake > ecommerce_project > models > models > marts > marts > fact_orders.sql

```
1 select
2   o.order_id,
3   o.customer_id,
4   o.order_date,
5   o.total_amount,
6   o.status,
7   p.amount as payment_amount,
8   p.payment_method,
9   s.shipped_date,
10  s.delivery_date,
11  s.shipping_status
12 from {{ ref('stg_orders') }} o
13 left join {{ ref('stg_payment') }} p
14   on o.order_id = p.order_id
15 left join {{ ref('stg_shipping') }} s
16   on o.order_id = s.order_id
17
```

OUTLINE | fact_orders.sql | 10 mins



←



+

Work with data

Projects

Ingestion

Transformation

AI & ML

Monitoring

Marketplace

Horizon Catalog

Catalog

\$390 credits left


Trial ends in 6 days

Upgrade

NMM

Nada Mahmoud...

ACCOUNTADMIN



My Workspace

Search for files

Add new

E-commerce data.sql

test.sql

Worksheets

Search worksheets

Owned

Shared with you

Database Explorer

Objects

Data Products

Search

Filter

DIM_CATEGORIES

DIM_CUSTOMERS

DIM_PRODUCTS

DIM_REGIONS

FACT_ORDERS

FACT_ORDER_ITEMS

5- Analytics in Snowflake

The screenshot displays the Snowflake SQL Editor interface. On the left, the 'Database Explorer' pane shows a list of objects including FACT_ORDERS, FACT_ORDER_ITEMS, MY_SECOND_DBT_M..., SALES_MONTHLY_O..., STG_CATEGORIES, STG_CUSTOMERS, and STG_ORDERS. The main editor area contains SQL code for creating two views. The first view, SALES_MONTHLY_ORDERS, is created with a SELECT statement that truncates the order date by month and counts the order IDs. The second view, SALES_MONTHLY_REVENUE, is created with a SELECT statement that truncates the order date by month and sums the total amount. Both views are created successfully, as indicated by the 'Results (just now)' pane at the bottom, which shows a single row with the status 'View SALES_MONTHLY_ORDERS successfully created.' and 'View SALES_MONTHLY_REVENUE successfully created.' respectively. The interface also includes a 'Worksheets' pane, a 'Search' bar, and a 'Filter' section.

```
207
208
209 CREATE OR REPLACE VIEW SALES_MONTHLY_ORDERS AS
210 SELECT
211     DATE_TRUNC('month', order_date) AS Month,
212     COUNT(order_id) AS OrderCount
213 FROM Orders
214 GROUP BY DATE_TRUNC('month', order_date)
215 ORDER BY Month;
216
217
218
219 CREATE OR REPLACE VIEW SALES_MONTHLY_ORDERS AS
```

Results (just now)

Table	Chart
status	
1	View SALES_MONTHLY_ORDERS successfully created.

Activate Windows
Go to Settings to activate Windows

Home E-commerce data.sql test.sql +

My Workspace > E-commerce data.sql

216
217
218 CREATE OR REPLACE VIEW SALES_MONTHLY_REVENUE AS
219 SELECT
220 DATE_TRUNC('month', ORDER_DATE) AS MONTH,
221 SUM(TOTAL_AMOUNT) AS TOTAL_REVENUE
222 FROM ECOMMERCE_DB.RAW.ORDERS
223 GROUP BY 1
224 ORDER BY 1;
225
226
227
228 CREATE OR REPLACE VIEW SALES_TOP_PRODUCTS AS

Results (just now)

Table	Chart
status	
1	View SALES_MONTHLY_REVENUE successfully created.

Home

E-commerce data.sql

test.sql

+

My Workspace > E-commerce data.sql

▶

▼

🔗

👤

🏠

📄

Share

⋮

226

227

228

229

230

231

232

233

234

235

236

237

238

CREATE OR REPLACE VIEW SALES_TOP_PRODUCTS AS

SELECT

p.PRODUCT_NAME AS PRODUCT_NAME,

SUM(oi.TOTAL_PRICE) AS REVENUE

FROM ECOMMERCE_DB.RAW.FACT_ORDER_ITEMS oi

JOIN ECOMMERCE_DB.RAW.DIM_PRODUCTS p

ON oi.PRODUCT_ID = p.PRODUCT_ID

GROUP BY p.PRODUCT_NAME

ORDER BY REVENUE DESC

LIMIT 5;

| Ctrl+I to generate

Results (just now)

Table

Chart

🔍

🔗

1 row ⓘ

227ms

⬇️

🕒

🔊

status

1

View SALES_TOP_PRODUCTS successfully created.

Activate Windows

Home

E-commerce data.sql

test.sql

+

My Workspace > E-commerce data.sql

▶

▼

🔗

👤

🏠

📄

Share

⋮

254

255

256

257

258

259

260

261

262

263

264

265

266

CREATE OR REPLACE VIEW SALES_REVENUE_BY_CATEGORY AS

SELECT

c.CATEGORY_NAME,

SUM(oi.QUANTITY * p.PRICE) AS REVENUE

FROM ECOMMERCE_DB.RAW.ORDER_ITEMS oi

JOIN ECOMMERCE_DB.RAW.PRODUCT_REAL p

ON oi.PRODUCT_ID = p.PRODUCT_ID

JOIN ECOMMERCE_DB.RAW.RAWCATEGORIES c

ON p.CATEGORY_ID = c.CATEGORY_ID

GROUP BY c.CATEGORY_NAME

ORDER BY REVENUE DESC;

Results (just now)

Table

Chart

🔍

🔗

1 row ⓘ

123ms

⬇️

🕒

🔊

status

1

View SALES_REVENUE_BY_CATEGORY successfully created.

My Workspace

Search for files

+ Add new

E-commerce data.sql

test.sql

Worksheets

Search worksheets

Owned

Shared with you

Database Explorer

Objects

Data Products

Search

Filter

DIM_CATEGORIES

DIM_CUSTOMERS

DIM_PRODUCTS

DIM_REGIONS

FACT_ORDERS

FACT_ORDER_ITEMS

MY_SECOND_DBT_M...

SALES_MONTHLY O...

Home

E-commerce data.sql

test.sql

+

My Workspace > E-commerce data.sql

267

268

269

270

271

272

273

274

275

276

277

278

```

CREATE OR REPLACE VIEW SALES_REVENUE_BY_REGION AS
SELECT
  r.REGION_NAME,
  SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE
FROM ECOMMERCE_DB.RAW.ORDERS o
JOIN ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
  ON o.CUSTOMER_ID = c.CUSTOMER_ID
JOIN ECOMMERCE_DB.RAW.REGION_REAL r
  ON c.REGION_ID = r.REGION_ID
GROUP BY r.REGION_NAME

```

Results (just now)

Table

Chart

1 row 106ms

status

1 View SALES_REVENUE_BY_REGION successfully created.

Activate Windows

Go to Settings to activate Windows

Feedback

Home

E-commerce data.sql

test.sql

+

My Workspace > E-commerce data.sql

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

```

GROUP BY r.REGION_NAME
ORDER BY TOTAL_REVENUE DESC;

CREATE OR REPLACE VIEW CUSTOMER_NEW_RETURNING_OVER_TIME AS
WITH customer_orders AS (
  SELECT
    CUSTOMER_ID,
    MIN(ORDER_DATE) AS FIRST_ORDER_DATE,
    COUNT(ORDER_ID) AS TOTAL_ORDERS
  FROM ECOMMERCE_DB.RAW.ORDERS
  GROUP BY CUSTOMER_ID
)
SELECT
  DATE_TRUNC('month', o.ORDER_DATE) AS MONTH,
  SUM(CASE WHEN co.TOTAL_ORDERS = 1 THEN 1 ELSE 0 END) AS NEW_CUSTOMERS,
  SUM(CASE WHEN co.TOTAL_ORDERS > 1 THEN 1 ELSE 0 END) AS RETURNING_CUSTOMERS
FROM ECOMMERCE_DB.RAW.ORDERS o
JOIN customer_orders co
  ON o.CUSTOMER_ID = co.CUSTOMER_ID
GROUP BY 1
ORDER BY 1;

```

Results (just now)

Table

Chart

1 row 132ms

status

1 View CUSTOMER_NEW_RETURNING_OVER_TIME successfully created.

Activate Windows

Go to Settings to activate Windows

My Workspace > E-commerce data.sql

278 ORDER BY 1,
299
300
301
302 CREATE OR REPLACE VIEW CUSTOMER_ORDERS_DISTRIBUTION AS
303 SELECT
304 COUNT(ORDER_ID) AS ORDERS_PER_CUSTOMER,
305 COUNT(CUSTOMER_ID) AS CUSTOMER_COUNT
306 FROM ECOMMERCE_DB.RAW.ORDERS
307 GROUP BY CUSTOMER_ID
308 ORDER BY ORDERS_PER_CUSTOMER;
309
310 Ctrl+I to generate
311
312
313

Results (just now)

Table Chart 1 row 317ms

	A status
1	View CUSTOMER_ORDERS_DISTRIBUTION successfully created.

Home E-commerce data.sql test.sql +

My Workspace > E-commerce data.sql

309
310
311
312
313
314
315
316
317
318
319
320
321
322
323

```
CREATE OR REPLACE VIEW TOP_CUSTOMERS_BY_REVENUE AS
SELECT
  c.CUSTOMER_ID,
  c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME,
  SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE,
  COUNT(o.ORDER_ID) AS TOTAL_ORDERS
FROM ECOMMERCE_DB.RAW.ORDERS o
JOIN ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
  ON o.CUSTOMER_ID = c.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, CUSTOMER_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 10;
```

Results (just now)

Table Chart 1 row 162ms

	A status
1	View TOP_CUSTOMERS_BY_REVENUE successfully created.

Home

E-commerce data.sql

test.sql

+

My Workspace > E-commerce data.sql

▶

▼

🔗

👤

🏠

📁

Share

...

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

```
CREATE OR REPLACE VIEW CUSTOMERS_BY_REGION AS
SELECT
    r.REGION_NAME,
    COUNT(DISTINCT c.CUSTOMER_ID) AS CUSTOMER_COUNT,
    SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE
FROM ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
LEFT JOIN ECOMMERCE_DB.RAW.ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID
LEFT JOIN ECOMMERCE_DB.RAW.REGION_REAL r
    ON c.REGION_ID = r.REGION_ID
GROUP BY r.REGION_NAME
ORDER BY TOTAL_REVENUE DESC;
```

Results (just now)

✕

Table

Chart

🔍 📄 1 row ⓘ 117ms 📉 ⌚

🔊

📄 status

1

View CUSTOMERS_BY_REGION successfully created.

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

```
CREATE OR REPLACE VIEW CUSTOMER_SEGMENTS AS
WITH customer_revenue AS (
    SELECT
        c.CUSTOMER_ID,
        c.FIRST_NAME || ' ' || c.LAST_NAME AS CUSTOMER_NAME,
        SUM(o.TOTAL_AMOUNT) AS TOTAL_REVENUE
    FROM ECOMMERCE_DB.RAW.RAWCUSTOMERS_REAL c
    LEFT JOIN ECOMMERCE_DB.RAW.ORDERS o
        ON c.CUSTOMER_ID = o.CUSTOMER_ID
    GROUP BY c.CUSTOMER_ID, CUSTOMER_NAME
),
quartiles AS (
    SELECT
        PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q1,
        PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q2,
        PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q3
    FROM customer_revenue
)
SELECT
```

Results (just now)

✕

Table

Chart

🔍 📄 1 row ⓘ 154ms 📉 ⌚

🔊

📄 status

1

View CUSTOMER_SEGMENTS successfully created.

Activate Windows
Go to Settings to activate Windows

```






349     SELECT
350         PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q1,
351         PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q2,
352         PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY TOTAL_REVENUE) AS Q3
353     FROM customer_revenue
354 )
355 SELECT
356     cr.CUSTOMER_ID,
357     cr.CUSTOMER_NAME,
358     cr.TOTAL_REVENUE,
359     CASE
360         WHEN cr.TOTAL_REVENUE >= q.Q3 THEN 'High'
361         WHEN cr.TOTAL_REVENUE >= q.Q2 THEN 'Medium'
362         ELSE 'Low'
363     END AS REVENUE_SEGMENT
364 FROM customer_revenue cr
365 CROSS JOIN quantiles q
366 ORDER BY cr.TOTAL_REVENUE DESC;
367
368





```

6- Connect to Power BI

The screenshot shows the Microsoft Power BI Desktop application. The 'Navigator' pane on the left lists data sources under 'airbyte_internal'. The 'FACT_ORDERS' table is selected. The main view displays a table of 'FACT_ORDERS' with the following data:

ORDER_ID	CUSTOMER_ID	ORDER_DATE	TOTAL_AMOUNT	STATUS
1	55885	11/4/2023 12:00:00 AM	1031	Delivered
2	55330	12/1/2023 12:00:00 AM	478	Delivered
3	94523	11/7/2023 12:00:00 AM	779	Shipped
4	43229	9/30/2023 12:00:00 AM	639	Pending
5	26869	10/2/2023 12:00:00 AM	2243	Shipped
6	61057	11/21/2023 12:00:00 AM	1308	Shipped
7	99964	11/1/2023 12:00:00 AM	1447	Shipped
8	20344	8/8/2023 12:00:00 AM	2813	Shipped
9	12761	2/11/2023 12:00:00 AM	1765	Delivered
10	45412	11/23/2023 12:00:00 AM	2661	Delivered
11	13780	12/14/2023 12:00:00 AM	1167	Shipped
12	56460	5/26/2023 12:00:00 AM	871	Delivered
13	52400	11/21/2023 12:00:00 AM	649	Delivered
14	76053	8/12/2023 12:00:00 AM	1254	Pending
15	93064	3/7/2023 12:00:00 AM	222	Shipped
16	30596	3/30/2023 12:00:00 AM	2072	Delivered
17	64261	8/10/2023 12:00:00 AM	798	Delivered
18	95325	2/23/2023 12:00:00 AM	2946	Pending
19	24514	10/3/2023 12:00:00 AM	2076	Shipped
20	6441	12/1/2023 12:00:00 AM	441	Pending
21	82406	1/2/2023 12:00:00 AM	1620	Pending
22	59648	4/19/2023 12:00:00 AM	1750	Shipped
23	45333	3/28/2023 12:00:00 AM	2081	Delivered

- >  SALES_MONTHLY_OR...
- >  SALES_MONTHLY_... ...
- >  SALES_REVENUE_BY_C...
- >  SALES_REVENUE_BY_R...
- >  SALES_TOP_PRODUCTS

- >  CUSTOMER_NEW_RETURNING_OVER_TIME
- >  CUSTOMER_ORDERS_DISTRIBUTION
- >  CUSTOMER_SEGMENTS
- >  CUSTOMERS_BY_REGION