ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
(ГАОУ ВО МГПУ)
Институт цифрового образования
Департамент информатики, управления и технологий

Практическая(лабораторная) работа № 2.1

по дисциплине «**Платформы Data Engineering**»

Выполнил:
студент группы БД-251м
Направление подготовки/Специальность
38.04.05 - Бизнес-информатика
Бобылева Варвара Владимировна
(Ф.И.О.)

Проверил:
Кандидат технических наук
(ученая степень, звание)
Босенко Тимур Муртазович
(Ф.И.О.)

Москва 2025

-- ==
-- Автор: Бобылева Варвара
-- Группа: БД-251м
-- Проект: superstore_dwh, student_dwh
-- Вариант индивидуального задания: 2
-- ==

1. Архитектура DWH

На рисунке ниже отображена схема Lineage Graph проекта superstore:

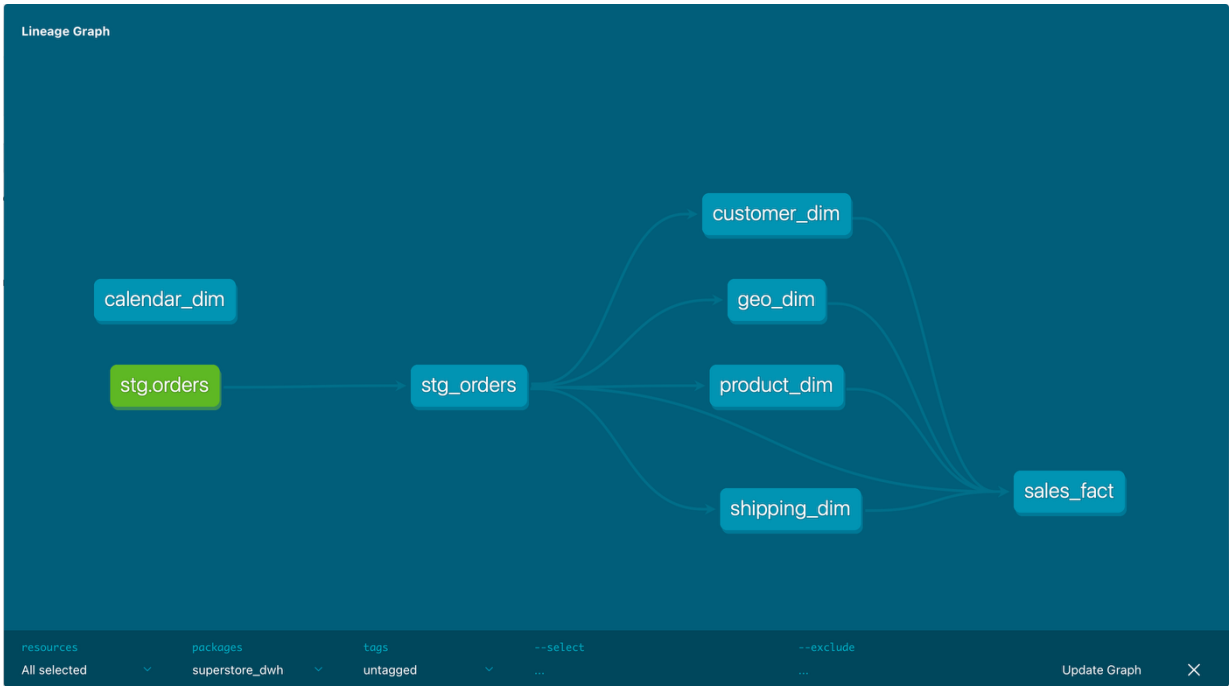


Рисунок 1 - Lineage Graph проекта superstore

На рисунке ниже отображена схема Lineage Graph проекта с учетом индивидуального задания:

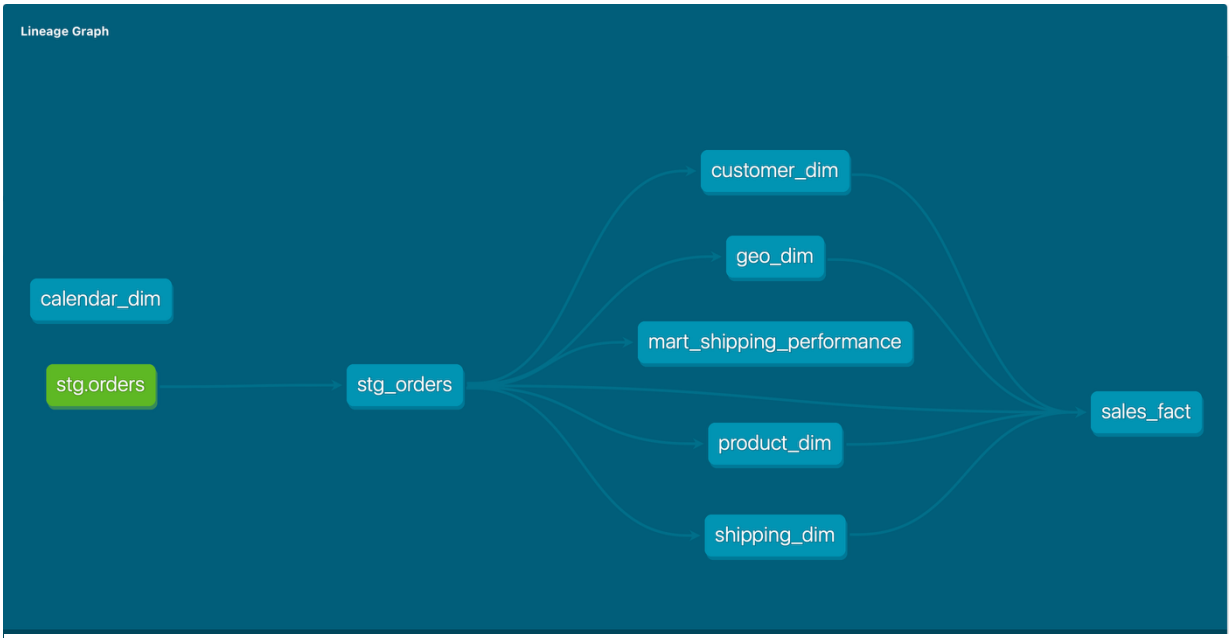


Рисунок 2 - Lineage Graph проекта с учетом индивидуального задания

2. **Ключевые фрагменты кода.**

Код модели stg_orders.sql:

```sql
-- models/staging/stg_orders.sql
-- Эта модель читает данные из исходной таблицы stg.orders,
-- приводит их к нужным типам и исправляет ошибку с почтовым кодом.
-- Все последующие модели будут ссылаться на эту, а не на исходную таблицу.

SELECT
    -- Приводим все к нижнему регистру для консистентности в dbt
    "order_id",
    ("order_date")::date as order_date,
    ("ship_date")::date as ship_date,
    "ship_mode",
    "customer_id",
    "customer_name",
    "segment",
    "country",
    "city",
    "state",
    -- Исправляем проблему с Burlington прямо здесь, один раз и навсегда
    CASE
        WHEN "city" = 'Burlington' AND "postal_code" IS NULL THEN '05401'
        ELSE "postal_code"
    END as postal_code,
    "region",
    "product_id",
    "category",
    "subcategory" as sub_category, -- переименовываем для соответствия
    "product_name",
    "sales",
    "quantity",
    "discount",
    "profit"
FROM {{ source('stg', 'orders') }}
```

Код модели sales_fact.sql:

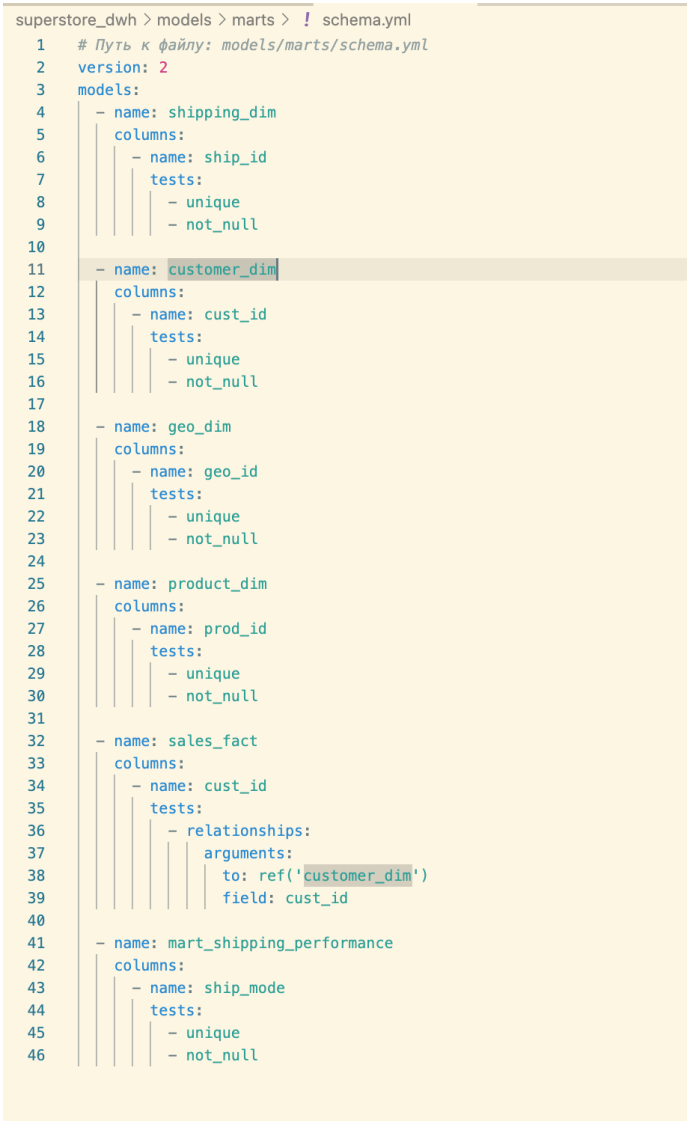
```sql
-- Создает таблицу фактов, объединяя все измерения
SELECT
    -- Суррогатные ключи из измерений
    cd.cust_id,
    pd.prod_id,
    sd.ship_id,
    gd.geo_id,
    -- Ключи для календаря
    to_char(o.order_date, 'yyyymmdd')::int AS order_date_id,
    to_char(o.ship_date, 'yyyymmdd')::int AS ship_date_id,
    -- Бизнес-ключ и метрики
    o.order_id,
    o.sales,
```

```
    o.profit,
    o.quantity,
    o.discount
FROM {{ ref('stg_orders') }} AS o
LEFT JOIN {{ ref('customer_dim') }} AS cd ON o.customer_id = cd.customer_id
LEFT JOIN {{ ref('product_dim') }} AS pd ON o.product_id = pd.product_id
LEFT JOIN {{ ref('shipping_dim') }} AS sd ON o.ship_mode = sd.ship_mode
LEFT JOIN {{ ref('geo_dim') }} AS gd ON o.postal_code = gd.postal_code AND o.city =
gd.city AND o.state = gd.state
```

Код модели mart_shipping_perfomance.sql (индивидуальное задание):

```
--Выполнение этого запроса предоставит таблицу, в которой будут перечислены способы
доставки, общее количество заказов для каждого из них и их средняя прибыльность.
SELECT
    sto.ship_mode,
    COUNT(order_id) AS total_orders,
    AVG(profit) AS average_profit
FROM {{ ref('stg_orders') }} AS sto
GROUP BY
    sto.ship_mode
ORDER BY
    average_profit DESC
```

Тесты моделей:



3. Результаты.

Результаты выполнения команды dbt test для проекта superstore_dwh:

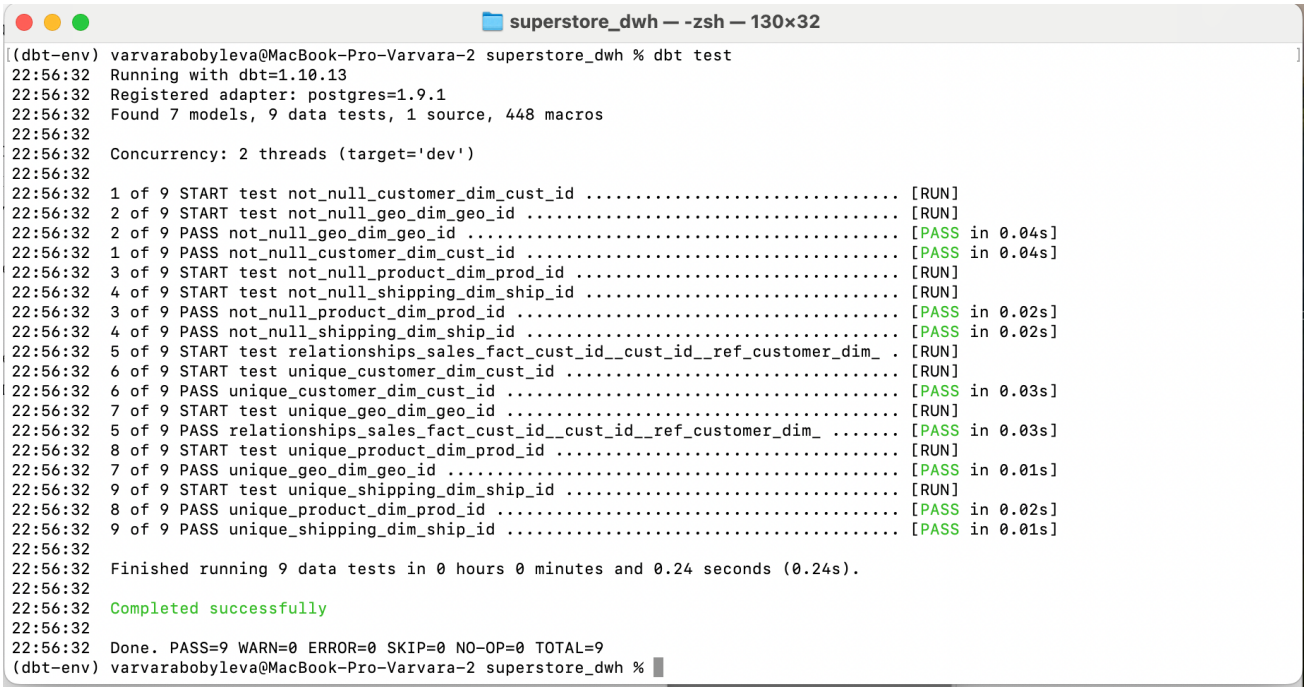


Рисунок 3 - выполнения команды dbt test

Результат выполнения команд dbt run и dbt test для проекта с индивидуальным заданием.

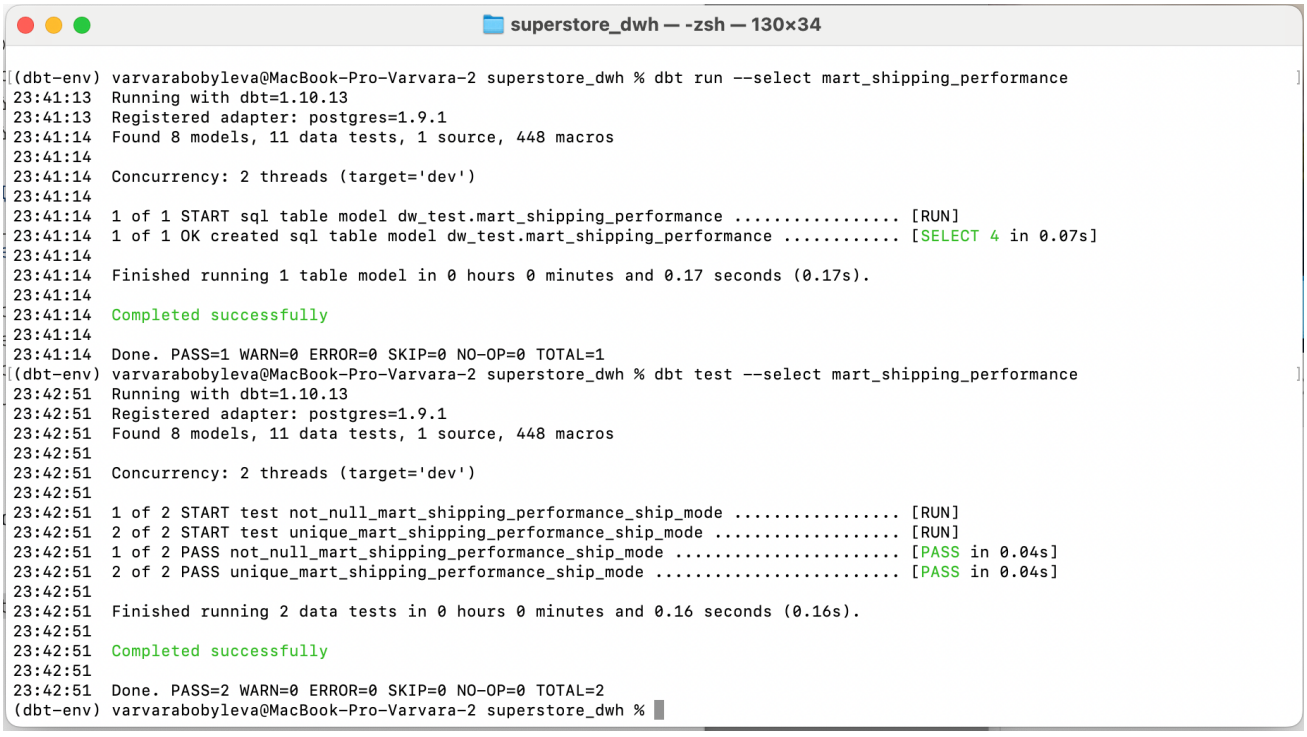
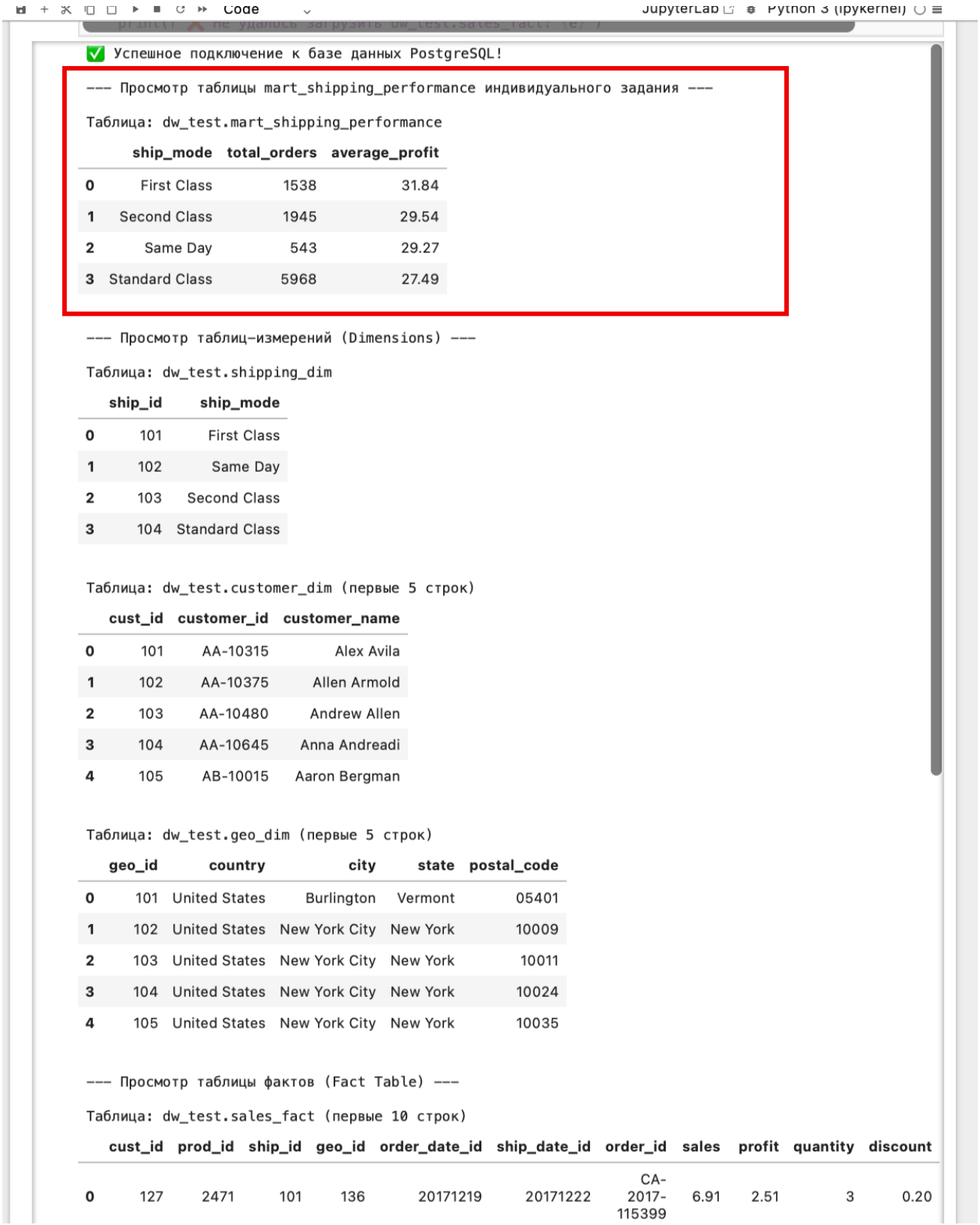


Рисунок 4 - выполнения команд dbt run и dbt test

Проверка работы в Jupyter. Вывод данных по таблице mart_shipping_performance индивидуального задания.

```
[18…] import pandas as pd
from sqlalchemy import create_engine

# --- 1. НАСТРОЙКИ ПОДКЛЮЧЕНИЯ (те же, что и раньше) ---
POSTGRES_USER = "varvarabobyleva"
POSTGRES_PASSWORD = "AB123cde"
POSTGRES_HOST = "localhost"
POSTGRES_PORT = "5432"
POSTGRES_DB = "superstore"

# Строка подключения
DATABASE_URL = f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_HOST}:{POSTGRES_PORT}/{POS

# Создаем "движок" для подключения
try:
    engine = create_engine(DATABASE_URL)
    print("✅ Успешное подключение к базе данных PostgreSQL!")
except Exception as e:
    print(f"❌ Ошибка при подключении: {e}")

# --- 2. ЗАГРУЗКА И ПРОСМОТР ДАННЫХ ИЗ СХЕМЫ dw_test ---

# Устанавливаем опцию, чтобы видеть все колонки в DataFrame
pd.set_option('display.max_columns', None)

print("\n--- Просмотр таблицы mart_shipping_performance индивидуального задания ---")
try:
    print("\nТаблица: dw_test.mart_shipping_performance")
    df_shipping = pd.read_sql("SELECT * FROM dw_test.mart_shipping_performance;", engine)
    display(df_shipping)
except Exception as e:
    print(f"❌ Не удалось загрузить dw_test.mart_shipping_performance: {e}")

print("\n--- Просмотр таблиц-измерений (Dimensions) ---")

# Загружаем справочник доставки (shipping_dim)
try:
    print("\nТаблица: dw_test.shipping_dim")
    df_shipping = pd.read_sql("SELECT * FROM dw_test.shipping_dim;", engine)
    display(df_shipping)
except Exception as e:
    print(f"❌ Не удалось загрузить dw_test.shipping_dim: {e}")

# Загружаем справочник клиентов (customer_dim)
try:
    print("\nТаблица: dw_test.customer_dim (первые 5 строк)")
    df_customer = pd.read_sql("SELECT * FROM dw_test.customer_dim LIMIT 5;", engine)
    display(df_customer)
except Exception as e:
    print(f"❌ Не удалось загрузить dw_test.customer_dim: {e}")

# Загружаем географический справочник (geo_dim)
try:
    print("\nТаблица: dw_test.geo_dim (первые 5 строк)")
    df_geo = pd.read_sql("SELECT * FROM dw_test.geo_dim LIMIT 5;", engine)
    display(df_geo)
except Exception as e:
    print(f"❌ Не удалось загрузить dw_test.geo_dim: {e}")


print("\n--- Просмотр таблицы фактов (Fact Table) ---")

# Загружаем таблицу фактов (sales_fact)
# ВАЖНО: таблицы фактов могут быть очень большими, поэтому всегда используем LIMIT при первом просмот
```

Рисунок 5 – Изменение запроса на вывод данных



✅ Успешное подключение к базе данных PostgreSQL!

--- Просмотр таблицы mart_shipping_performance индивидуального задания ---

Таблица: dw_test.mart_shipping_performance

	ship_mode	total_orders	average_profit
0	First Class	1538	31.84
1	Second Class	1945	29.54
2	Same Day	543	29.27
3	Standard Class	5968	27.49

--- Просмотр таблиц-измерений (Dimensions) ---

Таблица: dw_test.shipping_dim

	ship_id	ship_mode
0	101	First Class
1	102	Same Day
2	103	Second Class
3	104	Standard Class

Таблица: dw_test.customer_dim (первые 5 строк)

	cust_id	customer_id	customer_name
0	101	AA-10315	Alex Avila
1	102	AA-10375	Allen Armold
2	103	AA-10480	Andrew Allen
3	104	AA-10645	Anna Andreadi
4	105	AB-10015	Aaron Bergman

Таблица: dw_test.geo_dim (первые 5 строк)

	geo_id	country	city	state	postal_code
0	101	United States	Burlington	Vermont	05401
1	102	United States	New York City	New York	10009
2	103	United States	New York City	New York	10011
3	104	United States	New York City	New York	10024
4	105	United States	New York City	New York	10035

--- Просмотр таблицы фактов (Fact Table) ---

Таблица: dw_test.sales_fact (первые 10 строк)

	cust_id	prod_id	ship_id	geo_id	order_date_id	ship_date_id	order_id	sales	profit	quantity	discount
0	127	2471	101	136	20171219	20171222	CA-2017-115399	6.91	2.51	3	0.20

Рисунок 6 – Результат выполнения скрипта

Использование dbt при реализации хранилища данных имеет множество преимуществ по сравнению с ручным написанием DDL/DML скриптов. Основные из них:

- **Применение практик разработки ПО**. dbt позволяет использовать знакомые инженерам и аналитикам практики, такие как версионирование кода с помощью Git, модульность и тестирование. Это значительно повышает качество кода и упрощает совместную работу.

- **Автоматизация и эффективность**. dbt автоматизирует создание и обновление таблиц, представлений и инкрементальных моделей, избавляя от необходимости вручную писать сложный DDL/DML-код. Это ускоряет разработку и сокращает количество рутинных задач.

- **Тестирование и качество данных**. В dbt есть встроенные инструменты для тестирования моделей данных, что позволяет автоматически проверять качество данных и выявлять ошибки на ранних этапах.

- **Автоматическая документация**. dbt автоматически генерирует документацию и граф зависимостей, что обеспечивает прозрачность и понимание логики преобразований для всех членов команды.

- **Модульность и переиспользование**. dbt позволяет разбивать сложные преобразования на более мелкие, модульные части (модели), которые можно использовать повторно. Это упрощает отладку и поддержку.

- **Гибкость в материализации**. dbt предлагает разные стратегии материализации (таблицы, представления, инкрементальные модели) для оптимизации производительности и затрат, что легко настраивается через конфигурацию, а не вручную.

- **Сосредоточение на трансформации**. dbt позволяет аналитикам и инженерам сосредоточиться на бизнес-логике преобразований данных, а не на технических деталях управления схемой и загрузкой данных.