

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

Лабораторная работа № 2.1

по дисциплине «Платформы Data Engineering»

Выполнила:

студентка группы БД-251м

Направление подготовки/Специальность

38.04.05 - Бизнес-информатика

Савкина Мария Алексеевна

St_84

Вариант 25

Проверил:

Кандидат технических наук, доцент

Босенко Тимур Муртазович

Москва 2025

1. **Описание архитектуры.** Кратко описать назначение слоёв *intermediate* и *marts*.

Слой *intermediate* – промежуточный слой, в котором происходят все сложные преобразования. Основная задача данного слоя – создать переиспользуемые «строительные блоки», на данном этапе происходят основные JOIN’ы.

Слой *marts* – витрины, материализованные таблицы, специально подготовленные и агрегированные для бизнес-пользователей и решения конкретных бизнес-задач. Задача данного слоя – предоставить конечным пользователям быстрые и понятные данные для BI и аналитики.

2. Архитектура DWH.

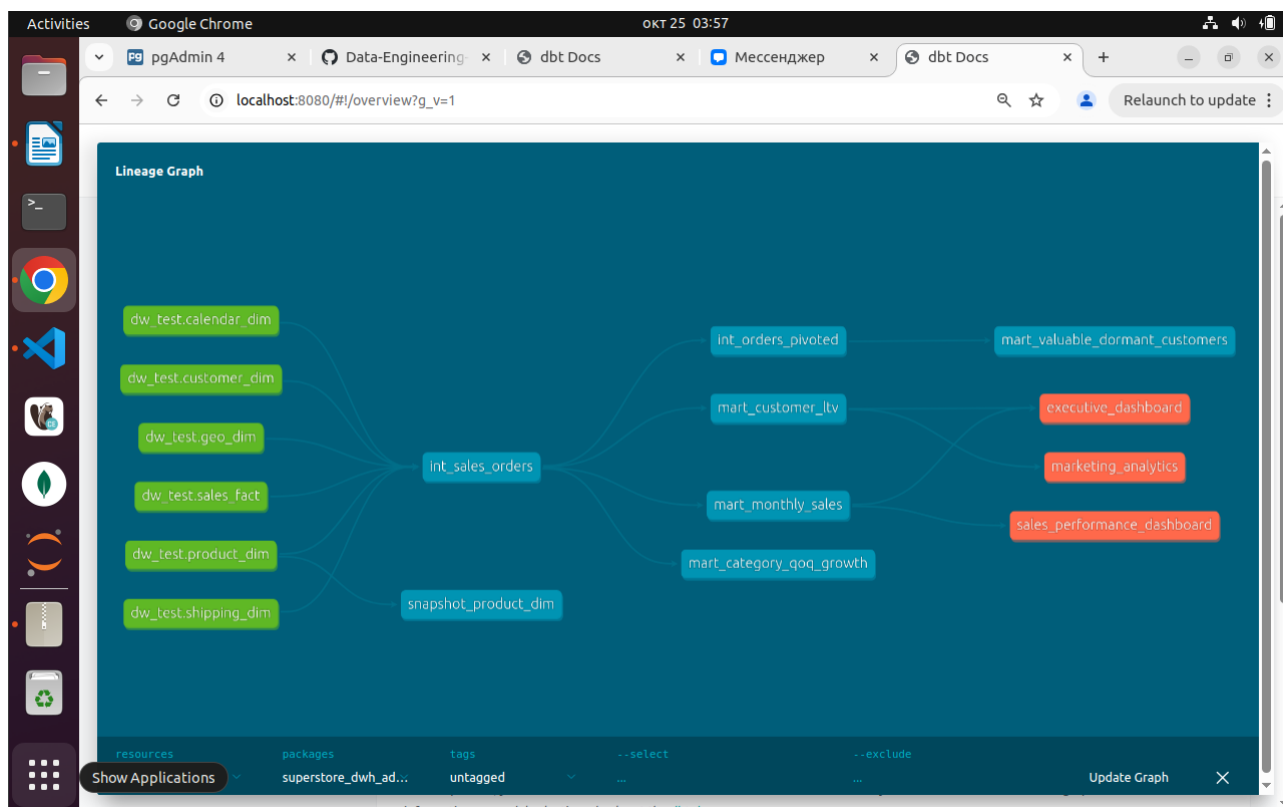


Рис. 1. Граф зависимостей lineage

Многоуровневая архитектура данных: staging → intermediate → marts

3. Ключевые фрагменты кода.

Код промежуточной модели int_sales_orders.sql

```
-- models/intermediate/int_sales_orders.sql
-- Эта модель объединяет факты со всеми измерениями, создавая
-- широкую, денормализованную таблицу для легкого использования в витринах.

SELECT
    -- Ключи
    f.order_id,

    -- Измерения из customer_dim
    c.customer_id,
    c.customer_name,

    -- Измерения из product_dim
    p.product_id,
    p.product_name,
    p.category,
    p.sub_category,
    p.segment,

    -- Измерения из geo_dim
    g.city,
    g.state,

    -- Измерения из shipping_dim
    s.ship_mode,

    -- Даты из calendar_dim (с правильными псевдонимами)
    cal_order.date as order_date,
    cal_ship.date as ship_date,

    -- Метрики из sales_fact
    f.sales,
    f.profit,
    f.quantity,
    f.discount

FROM {{ source('dw_test', 'sales_fact') }} AS f
LEFT JOIN {{ source('dw_test', 'customer_dim') }} AS c ON f.cust_id =
c.cust_id
LEFT JOIN {{ source('dw_test', 'product_dim') }} AS p ON f.prod_id =
p.prod_id
LEFT JOIN {{ source('dw_test', 'shipping_dim') }} AS s ON f.ship_id =
s.ship_id
LEFT JOIN {{ source('dw_test', 'geo_dim') }} AS g ON f.geo_id = g.geo_id
-- ИСПРАВЛЕНО: Добавляем псевдонимы, так как календарь используется дважды
LEFT JOIN {{ source('dw_test', 'calendar_dim') }} AS cal_order ON
f.order_date_id = cal_order.dateid
LEFT JOIN {{ source('dw_test', 'calendar_dim') }} AS cal_ship ON
f.ship_date_id = cal_ship.dateid
```

Код индивидуальной mart-модели mart_category_qoq_growth.sql (Вариант 25)

```
-- models/marts/mart_category_qoq_growth.sql

WITH quarterly_sales AS (
    SELECT
        date_trunc('quarter', order_date)::date AS sales_quarter,
        category,
        SUM(sales) AS total_sales
    FROM {{ ref('int_sales_orders') }}
    GROUP BY 1, 2
),

sales_with_previous_quarter AS (
    SELECT
        *,
        LAG(total_sales, 1) OVER (PARTITION BY category ORDER BY sales_quarter)
    AS previous_quarter_sales
    FROM quarterly_sales
)

SELECT
    sales_quarter,
    category,
    total_sales,
    previous_quarter_sales,
    COALESCE(
        (total_sales - previous_quarter_sales) / NULLIF(previous_quarter_sales,
0) * 100, 0) AS qoq_growth_percentage
FROM sales_with_previous_quarter
ORDER BY category, sales_quarter
```

Дополнительный тест в models/marts/schema.yml:

```
- name: mart_category_qoq_growth
  columns:
    - name: total_sales
      tests:
        - is_positive
```

Код кастомного теста test_is_positive.sql

```
-- tests/generic/test_is_positive.sql
{% test is_positive(model, column_name) %}
SELECT *
FROM {{ model }}
WHERE {{ column_name }} < 0
{% endtest %}
```

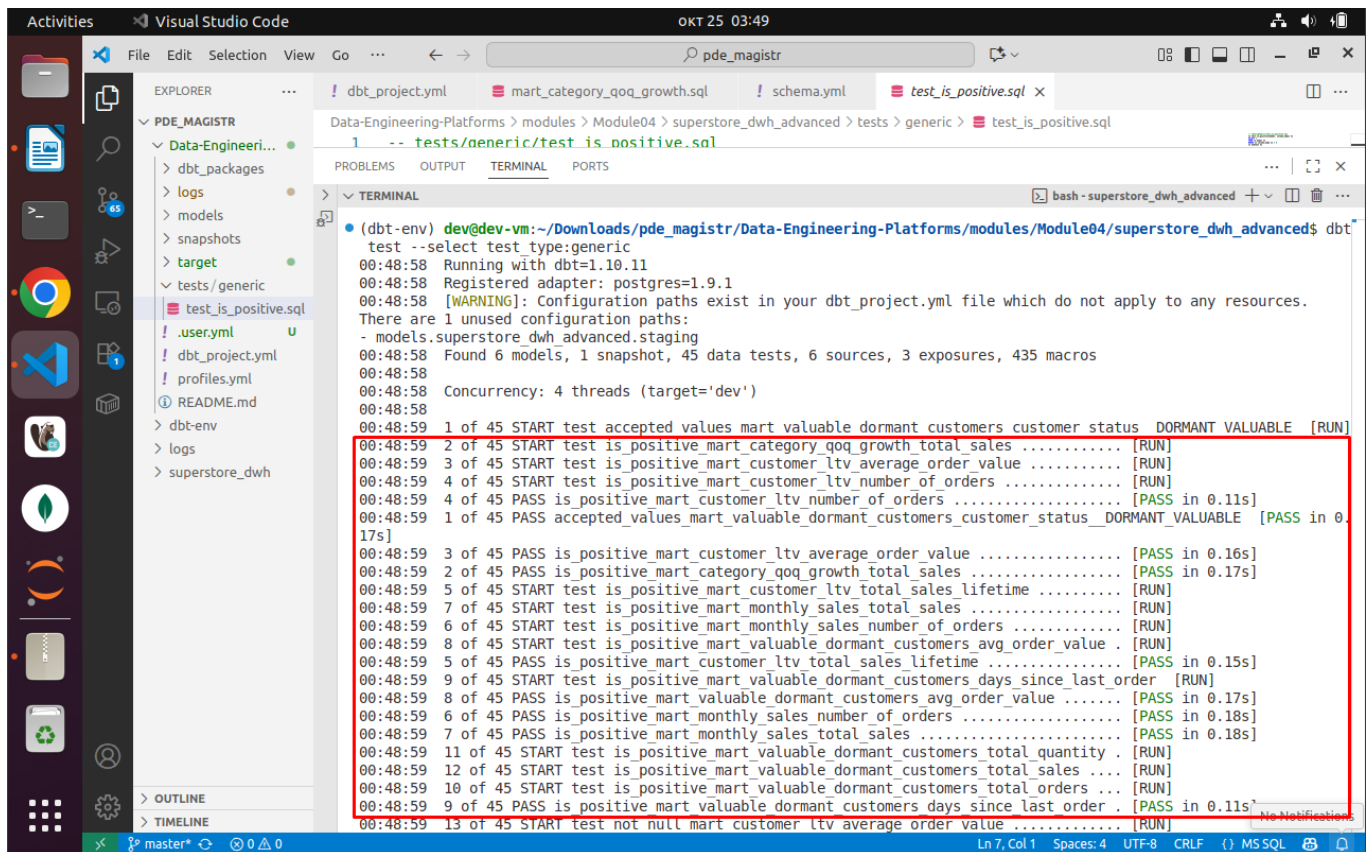
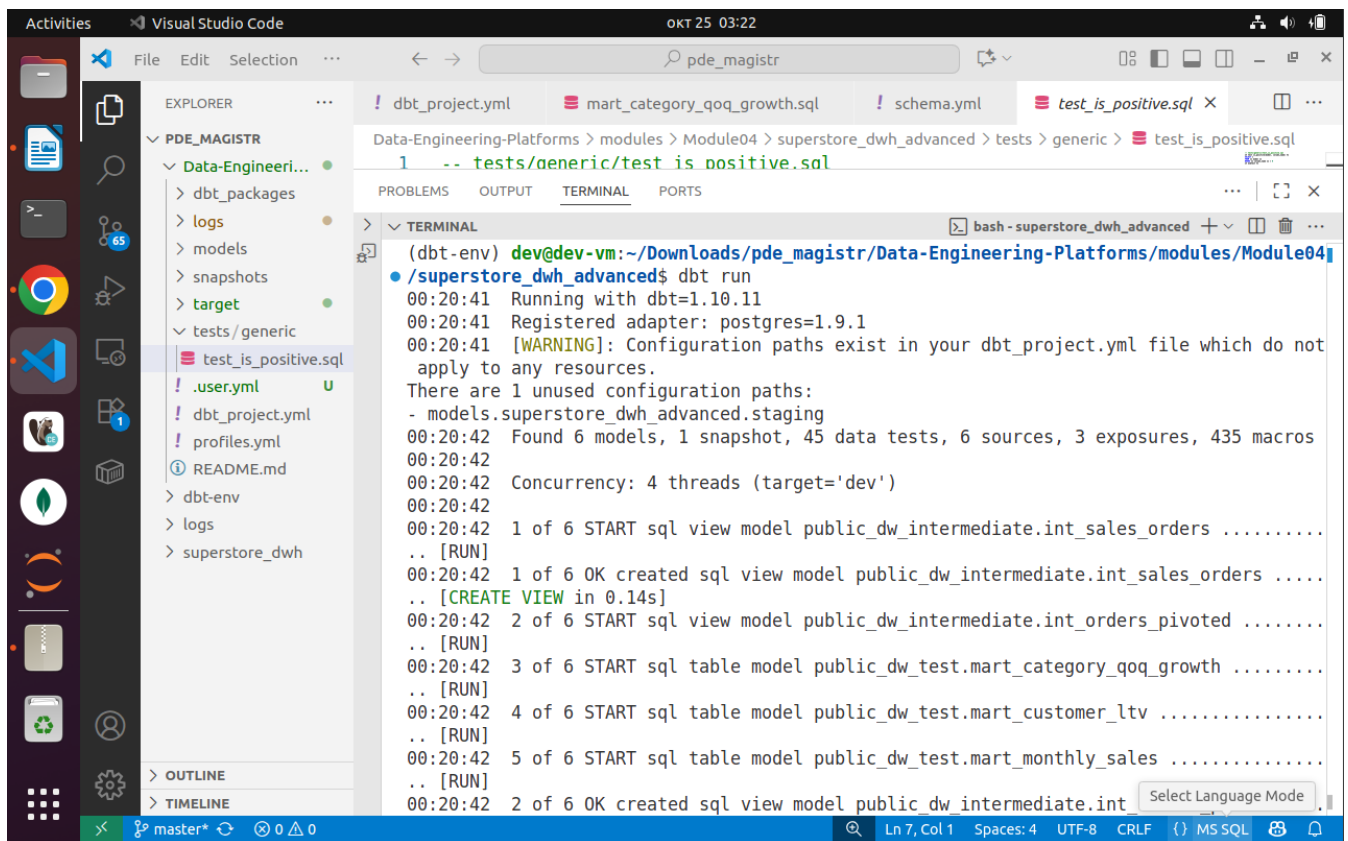


Рис. 2. Пример применения кастомного теста test is positive

Код snapshot_product_dim.sql

```
-- snapshots/snapshot_product_dim.sql
{% snapshot snapshot_product_dim %}
{{
    config(
        target_schema='dw_snapshots',
        strategy='check',
        unique_key='prod_id',
        check_cols=['segment', 'category'],
    )
}}
SELECT prod_id, product_id, segment, category FROM {{ source('dw_test',
'product_dim') }}
{% endsnapshot %}
```

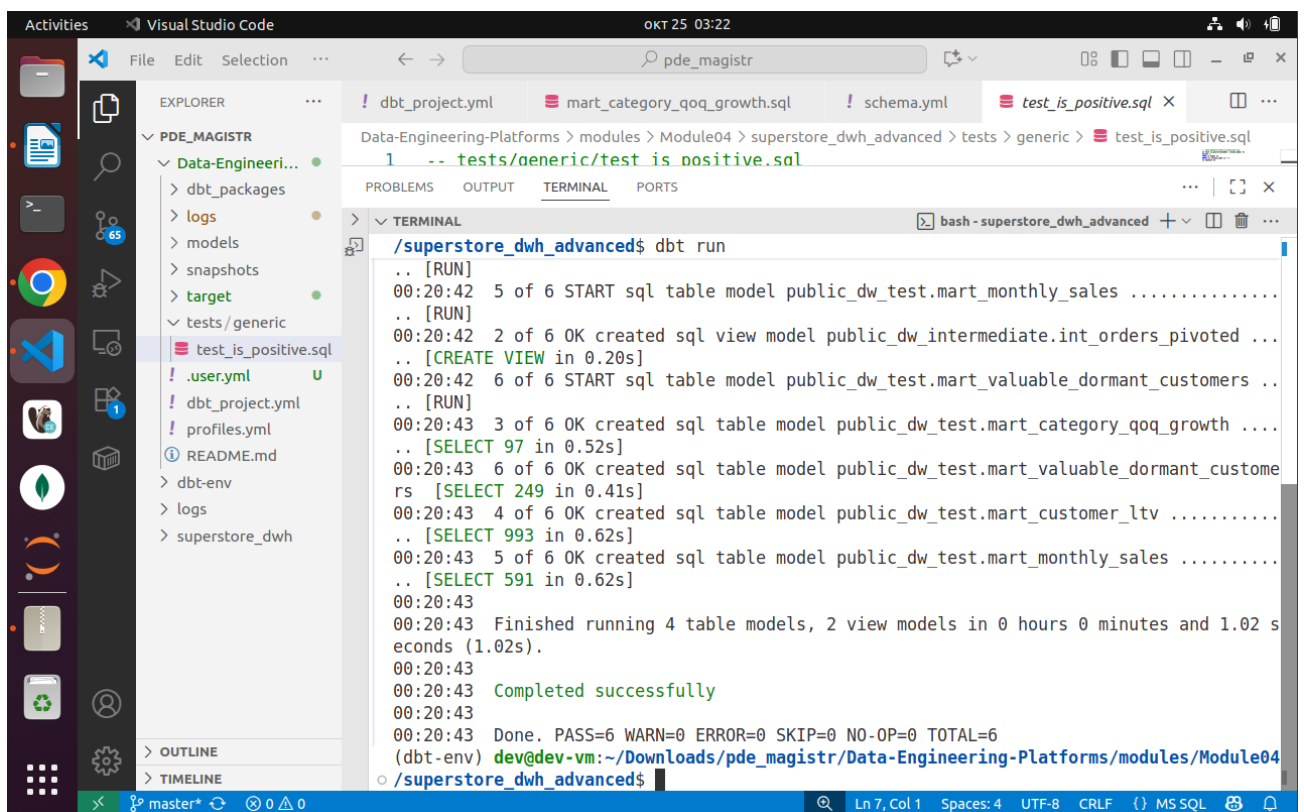
Результаты



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** The file tree on the left shows the project structure under 'PDE_MAGISTR', including 'Data-Engineering-Platforms', 'dbt_packages', 'logs', 'models', 'snapshots', 'target', and 'tests/generic'. The file 'test_is_positive.sql' is selected.
- Terminal:** The terminal window shows the execution of 'dbt run' in a shell environment. The output includes:
 - Running with dbt=1.10.11
 - Registered adapter: postgres=1.9.1
 - [WARNING]: Configuration paths exist in your dbt_project.yml file which do not apply to any resources.
 - There are 1 unused configuration paths: - models.superstore_dwh_advanced.staging
 - Found 6 models, 1 snapshot, 45 data tests, 6 sources, 3 exposures, 435 macros
 - Concurrency: 4 threads (target='dev')
 - Execution progress: 1 of 6 START sql view model public_dw_intermediate.int_sales_orders, 1 of 6 OK created sql view model public_dw_intermediate.int_sales_orders, 2 of 6 START sql view model public_dw_intermediate.int_orders_pivoted, 3 of 6 START sql table model public_dw_test.mart_category_qoq_growth, 4 of 6 START sql table model public_dw_test.mart_customer_ltv, 5 of 6 START sql table model public_dw_test.mart_monthly_sales, 2 of 6 OK created sql view model public_dw_intermediate.int

Рис. 3. Скриншот успешного выполнения dbt run (часть 1)



The screenshot shows the continuation of the dbt run in the Visual Studio Code terminal. The output includes:

- Execution progress: 5 of 6 START sql table model public_dw_test.mart_monthly_sales, 2 of 6 OK created sql view model public_dw_intermediate.int_orders_pivoted, 6 of 6 START sql table model public_dw_test.mart_valuable_dormant_customers, 3 of 6 OK created sql table model public_dw_test.mart_category_qoq_growth, 6 of 6 OK created sql table model public_dw_test.mart_valuable_dormant_customers, 4 of 6 OK created sql table model public_dw_test.mart_customer_ltv, 5 of 6 OK created sql table model public_dw_test.mart_monthly_sales.
- Final status: Finished running 4 table models, 2 view models in 0 hours 0 minutes and 1.02 seconds (1.02s). Completed successfully.
- Statistics: Done. PASS=6 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=6.

Рис. 4. Скриншот успешного выполнения dbt run (часть 2)

```

Data-Engineering-Platforms > modules > Module04 > superstore_dwh_advanced > tests > generic > test_is_positive.sql
1  -- tests/generic/test_is_positive.sql

PROBLEMS  OUTPUT  TERMINAL  PORTS
> TERMINAL
bash - superstore_dwh_advanced +
(dbt-env) dev@dev-vm:~/Downloads/pde_magistr/Data-Engineering-Platforms/modules/Module04
/superstore_dwh_advanced$ dbt test
.. [PASS in 0.15s]
00:23:40 42 of 45 PASS not_null_mart_valuable_dormant_customers_total_quantity .....
.. [PASS in 0.14s]
00:23:40 44 of 45 START test unique_mart_customer_ltv_customer_id .....
.. [RUN]
00:23:40 45 of 45 START test unique_mart_valuable_dormant_customers_customer_id .....
.. [RUN]
00:23:40 40 of 45 PASS not_null_mart_valuable_dormant_customers_total_orders .....
.. [PASS in 0.20s]
00:23:41 43 of 45 PASS not_null_mart_valuable_dormant_customers_total_sales .....
.. [PASS in 0.16s]
00:23:41 44 of 45 PASS unique_mart_customer_ltv_customer_id .....
.. [PASS in 0.13s]
00:23:41 45 of 45 PASS unique_mart_valuable_dormant_customers_customer_id .....
.. [PASS in 0.15s]
00:23:41 Finished running 45 data tests in 0 hours 0 minutes and 2.46 seconds (2.46s).
00:23:41 Completed successfully
00:23:41 Done. PASS=45 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=45
(dbt-env) dev@dev-vm:~/Downloads/pde_magistr/Data-Engineering-Platforms/modules/Module04
/superstore_dwh_advanced$

```

Рис. 5. Скриншот успешного выполнения dbt test (итог)

```

Data-Engineering-Platforms > modules > Module04 > superstore_dwh_advanced > tests > generic > test_is_positive.sql
1  -- tests/generic/test_is_positive.sql

PROBLEMS  OUTPUT  TERMINAL  PORTS
> TERMINAL
bash - superstore_dwh_advanced +
(dbt-env) dev@dev-vm:~/Downloads/pde_magistr/Data-Engineering-Platforms/modules/Module04
/superstore_dwh_advanced$ dbt snapshot
00:26:10 Running with dbt=1.10.11
00:26:11 Registered adapter: postgres=1.9.1
00:26:11 [WARNING]: Configuration paths exist in your dbt_project.yml file which do not
apply to any resources.
There are 1 unused configuration paths:
- models.superstore_dwh_advanced.staging
00:26:11 Found 6 models, 1 snapshot, 45 data tests, 6 sources, 3 exposures, 435 macros
00:26:11 Concurrency: 4 threads (target='dev')
00:26:11 1 of 1 START snapshot dw_snapshots.snapshot_product_dim .....
.. [RUN]
00:26:12 1 of 1 OK snapshotted dw_snapshots.snapshot_product_dim .....
.. [INSERT 0 0 in 0.29s]
00:26:12 Finished running 1 snapshot in 0 hours 0 minutes and 0.47 seconds (0.47s).
00:26:12 Completed successfully
00:26:12 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=1
(dbt-env) dev@dev-vm:~/Downloads/pde_magistr/Data-Engineering-Platforms/modules/Module04
/superstore_dwh_advanced$

```

Рис. 6. Скриншот успешного выполнения dbt snapshot

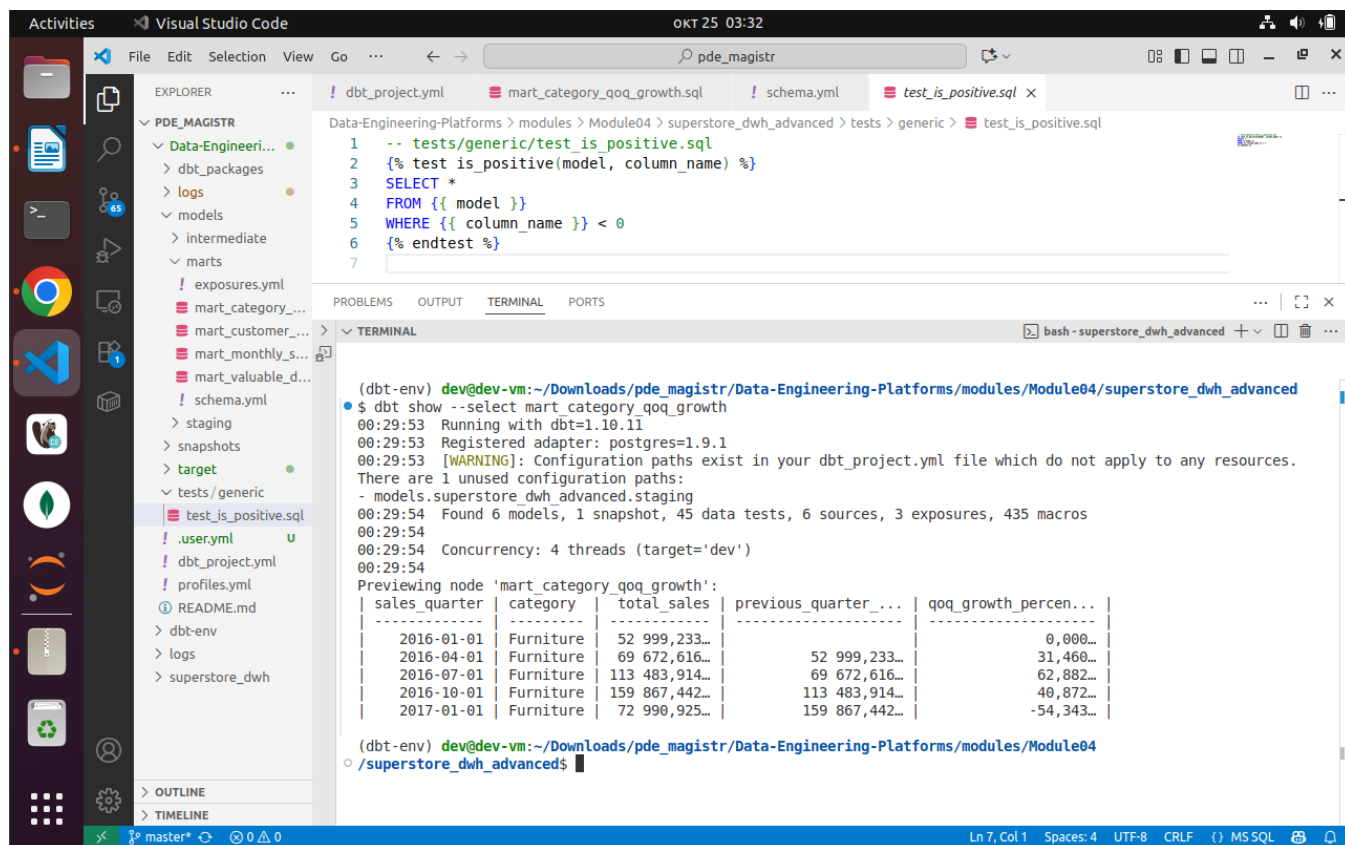


Рис 7. Скриншот с данными из mart_category_qoq_growth

Выводы.

В ходе выполнения лабораторной работы по построению аналитических витрин и внедрению продвинутых dbt-концепций мною были выявлены следующие преимущества использования промежуточных моделей и витрин по сравнению с работой напрямую с единой таблицей фактов:

Во-первых, при росте количества продаж JOIN'ы в моделях становятся неэффективными и долгими, поэтому использование промежуточной модели экономит время и уменьшает количество ошибок.

Во-вторых, упрощение тестирования при использовании кастомных тестов (в частности, is_positive, который можно применить ко множеству колонок).

В-третьих, нет необходимости предоставлять доступ пользователю (например, конкретному отделу) ко всей системе — достаточно предоставить доступ к определённой дашборду. В случае ошибки, пользователь обращается к data

engineer'у, который с помощью графа lineage может проанализировать на каком этапе происходит ошибка.