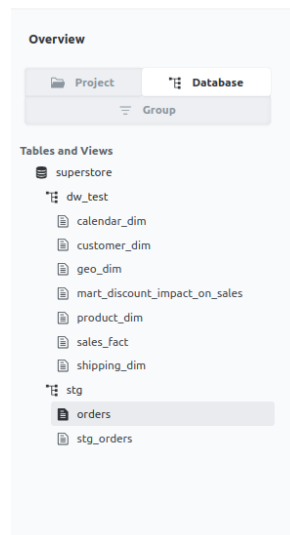
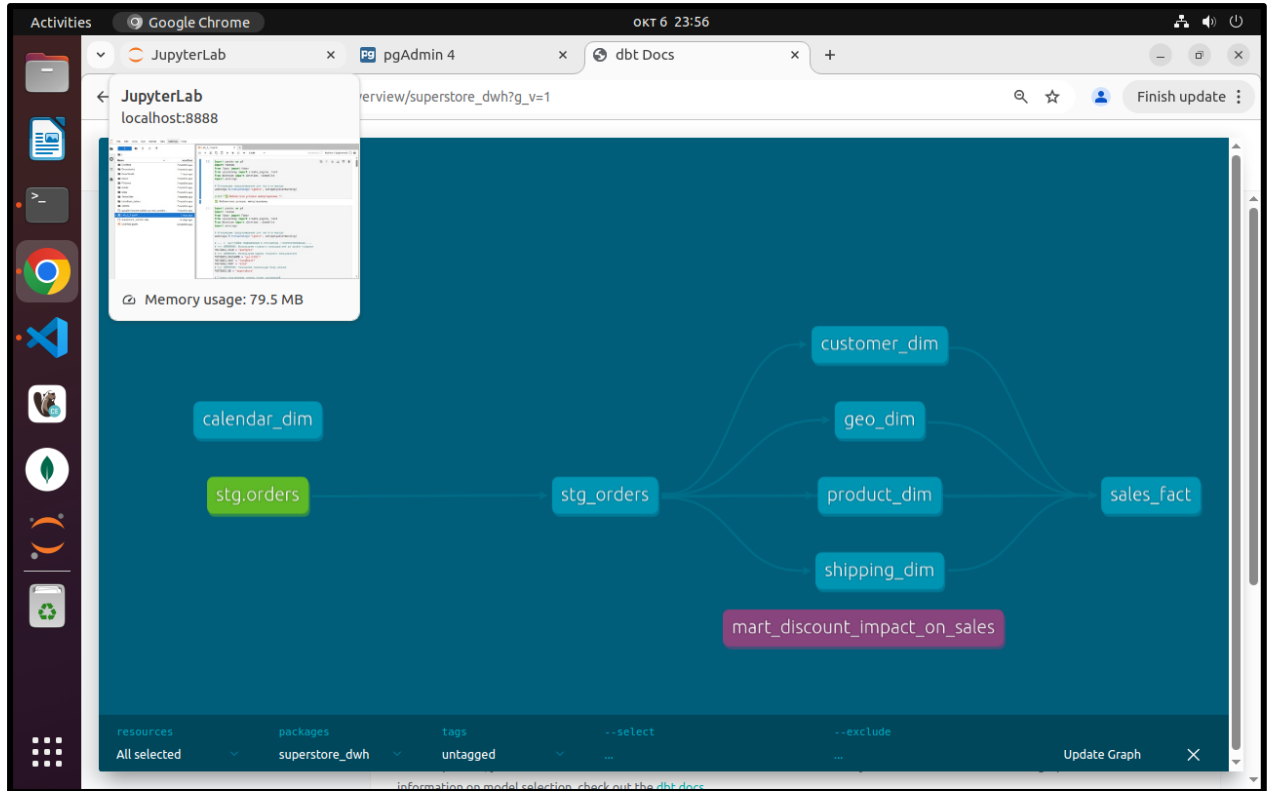


Практическая работа 2.1 Разработка и тестирование dbt-моделей
для бизнес-логики

Работу выполнил студент МГПУ

Войт Иван Иванович

Архитектура DWH



Код модели stg_orders.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

```
-- 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_discount_impact_on_sales.sql

```
-- models/marts/mart_discount_impact_on_sales.sql
SELECT
CASE
WHEN discount = 0 THEN 'Без скидки'
WHEN discount > 0 AND discount <= 0.2 THEN 'Малая скидка (1-20%)'
WHEN discount > 0.2 AND discount <= 0.4 THEN 'Средняя скидка (21-40%)'
WHEN discount > 0.4 AND discount <= 0.6 THEN 'Высокая скидка (41-60%)'
ELSE 'Очень высокая скидка (>60%)'
END AS discount_segment,
COUNT(DISTINCT order_id) AS order_count,
SUM(sales) AS total_sales,
SUM(quantity) AS total_quantity,
SUM(profit) AS total_profit,
AVG(sales) AS avg_sales_per_order,
AVG(quantity) AS avg_quantity_per_order,
AVG(profit) AS avg_profit_per_order,
CASE
WHEN SUM(sales) = 0 THEN 0
ELSE (SUM(profit) / SUM(sales)) * 100
END AS profit_margin_percent,
COUNT(*) AS transaction_count,
ROUND((SUM(sales) / (SELECT SUM(sales) FROM {{ ref('sales_fact') }})) *
100, 2) AS sales_percentage_of_total
FROM {{ ref('sales_fact') }}
GROUP BY
CASE
WHEN discount = 0 THEN 'Без скидки'
WHEN discount > 0 AND discount <= 0.2 THEN 'Малая скидка (1-20%)'
WHEN discount > 0.2 AND discount <= 0.4 THEN 'Средняя скидка (21-40%)'
WHEN discount > 0.4 AND discount <= 0.6 THEN 'Высокая скидка (41-60%)'
ELSE 'Очень высокая скидка (>60%)'
END
ORDER BY
CASE
WHEN 'Без скидки' THEN 1
WHEN 'Малая скидка (1-20%)' THEN 2
WHEN 'Средняя скидка (21-40%)' THEN 3
WHEN 'Высокая скидка (41-60%)' THEN 4
ELSE 5
END
```

Объяснение логики модели mart_discount_impact_on_sales.sql:

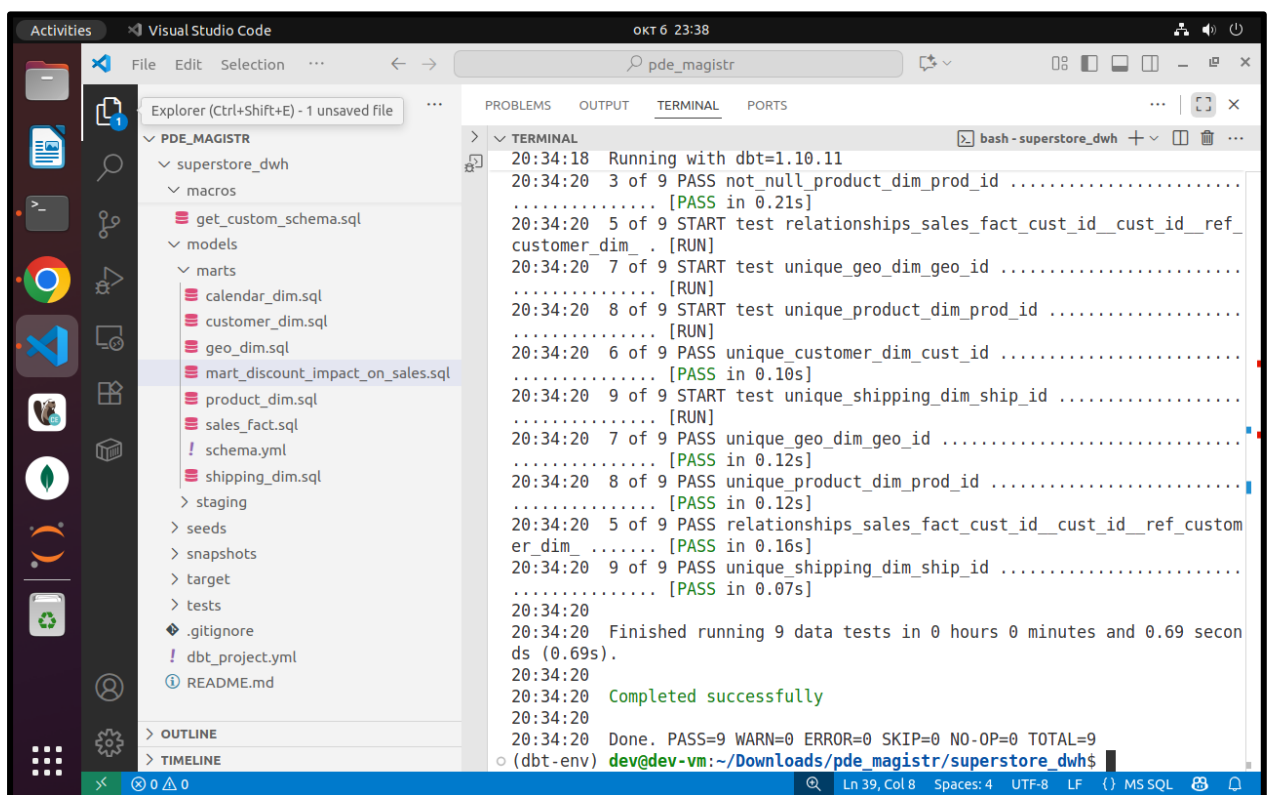
Сегментация Скидок:

- **Без скидки (0%)** - базовый уровень для сравнения
- **Малая скидка (1-20%)** - стандартные промо-акции
- **Средняя скидка (21-40%)** - сезонные распродажи
- **Высокая скидка (41-60%)** - акции по очистке склада
- **Очень высокая скидка (>60%)** - экстремальные распродажи

Ключевые метрики:

- **Order Count** - количество заказов для оценки популярности
- **Total Sales** - общий объем продаж
- **Profit Margin** - маржинальность для анализа рентабельности
- **Sales Percentage** - доля в общем объеме продаж

Выполнение dbt test:



```
Activities Visual Studio Code окт 6 23:38
File Edit Selection ... pde_magistr
Explorer (Ctrl+Shift+E) - 1 unsaved file
  PDE_MAGISTR
    superstore_dwh
      macros
        get_custom_schema.sql
      models
        marts
          calendar_dim.sql
          customer_dim.sql
          geo_dim.sql
          mart_discount_impact_on_sales.sql
          product_dim.sql
          sales_fact.sql
          schema.yml
          shipping_dim.sql
        staging
        seeds
        snapshots
        target
        tests
      .gitignore
      dbt_project.yml
      README.md
  OUTLINE
  TIMELINE
PROBLEMS OUTPUT TERMINAL PORTS
  TERMINAL
    20:34:18 Running with dbt=1.10.11
    20:34:20 3 of 9 PASS not_null_product_dim_prod_id ..... [PASS in 0.21s]
    20:34:20 5 of 9 START test relationships_sales_fact_cust_id_cust_id_ref_customer_dim_ . [RUN]
    20:34:20 7 of 9 START test unique_geo_dim_geo_id ..... [RUN]
    20:34:20 8 of 9 START test unique_product_dim_prod_id ..... [RUN]
    20:34:20 6 of 9 PASS unique_customer_dim_cust_id ..... [PASS in 0.10s]
    20:34:20 9 of 9 START test unique_shipping_dim_ship_id ..... [RUN]
    20:34:20 7 of 9 PASS unique_geo_dim_geo_id ..... [PASS in 0.12s]
    20:34:20 8 of 9 PASS unique_product_dim_prod_id ..... [PASS in 0.12s]
    20:34:20 5 of 9 PASS relationships_sales_fact_cust_id_cust_id_ref_custom er_dim_ ..... [PASS in 0.16s]
    20:34:20 9 of 9 PASS unique_shipping_dim_ship_id ..... [PASS in 0.07s]
    20:34:20 Finished running 9 data tests in 0 hours 0 minutes and 0.69 seconds (0.69s).
    20:34:20 Completed successfully
    20:34:20 Done. PASS=9 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=9
    (dbt-env) dev@dev-vm:~/Downloads/pde_magistr/superstore_dwh$
```

Ключевые преимущества dbt по сравнению с ручным написанием DDL/DML скриптов:

1. Автоматизация и стандартизация

- **dbt** автоматически генерирует DDL для всех моделей, исключая человеческие ошибки
- **Ручные скрипты** требуют ручного написания каждого CREATE TABLE, что подвержено ошибкам и несогласованности

2. Тестирование данных

- **dbt** предоставляет встроенную систему тестирования (unique, not_null, relationships)
- **Ручные скрипты** требуют написания кастомных проверок данных, которые часто игнорируются

3. Документирование

- **dbt** автоматически генерирует документацию и граф зависимостей
- **Ручные скрипты** требуют отдельного ведения документации, которая быстро устаревает

4. Версионирование и сотрудничество

- **dbt** проекты хранятся в Git, обеспечивая контроль версий и совместную работу
- **Ручные скрипты** сложно версионировать и координировать между разработчиками

5. Повторное использование кода

- **dbt** позволяет использовать макросы и дженерики для повторяющейся логики
- **Ручные скрипты** ведут к дублированию кода и несогласованности

6. Моделирование как код

- **dbt** рассматривает преобразования данных как код с полным циклом разработки
- **Ручные скрипты** часто являются одноразовыми решениями без должной архитектуры

На примере нашего проекта:

Автоматическое создание всех измерений и фактов через dbt run

Тестирование целостности связей между таблицами через dbt test

Прозрачность зависимостей - визуализация через dbt docs serve

Быстрое развертывание в разные схемы (dw_test, dw_student) без изменения кода

Легкое сопровождение - все преобразования в одном репозитории с историей изменений

Итог: dbt превращает процесс построения DWH из набора разрозненных скриптов в управляемый, тестируемый и документированный инженерный процесс, что критически важно для создания надежного "единого источника истины" в компании.