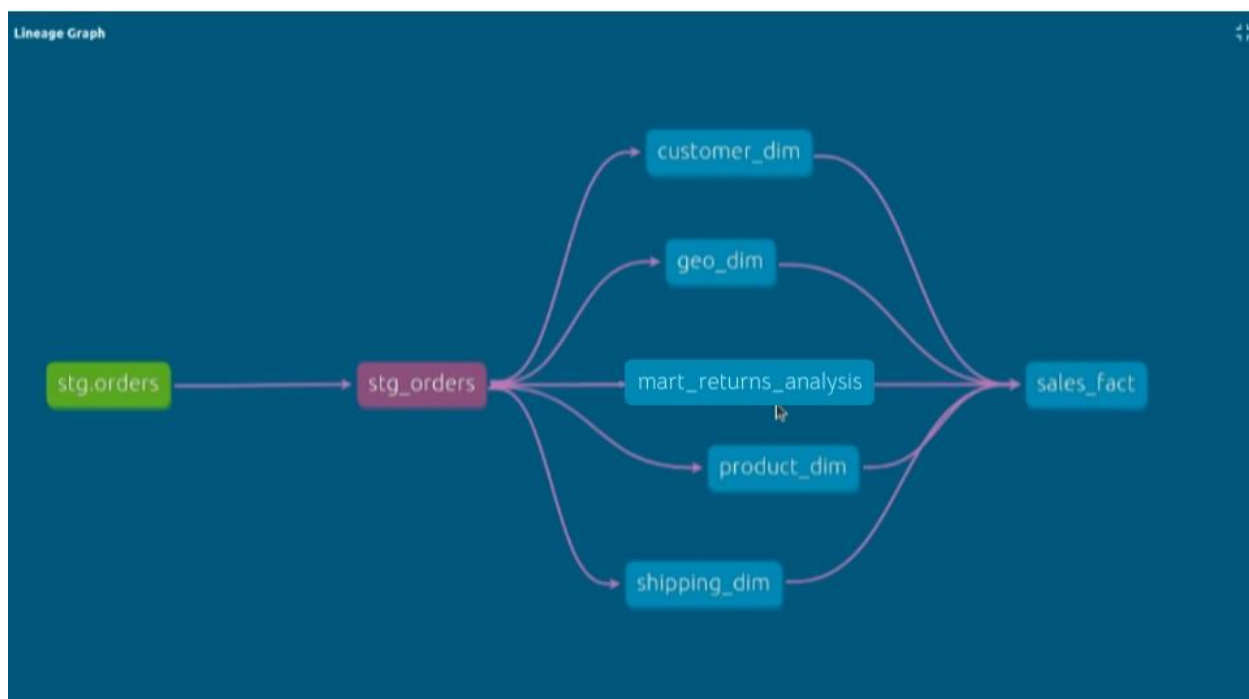


Практическая работа 2.1 Разработка и тестирование dbt-моделей для бизнес-логики

Работу выполнил студент МГПУ

Варданян Роберт Барсегович

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_returns_analysis

```
-- models/marts/mart_returns_analysis.sql

SELECT
    p.subcategory,
    COUNT(DISTINCT r.order_id) AS returned_orders_count,
    SUM(f.sales) AS total_sales_loss
FROM {{ ref('sales_fact') }} AS f
LEFT JOIN {{ ref('product_dim') }} AS p
    ON f.product_id = p.product_id
LEFT JOIN {{ source('stg', 'returns') }} AS r
    ON f.order_id = r.order_id
WHERE r.order_id IS NOT NULL
GROUP BY p.subcategory
ORDER BY returned_orders_count DESC;
```

Ключевые аспекты анализа:

1. Структура возвратов:

- **Группировка по подкатегориям** - позволяет выявить, какие типы товаров чаще возвращаются
- **Возвращенные заказы vs Общие продажи** - анализ исключительно проблемных транзакций

2. Финансовые метрики:

- **returned_orders_count** - количество уникальных возвращенных заказов
 - Показывает масштаб проблемы по каждой категории
 - Высокое значение = системная проблема с качеством/соответствием
- **total_sales_loss** - суммарный объем потерянной выручки
 - Критически важна для оценки финансового ущерба
 - Помогает расставить приоритеты при решении проблем

Бизнес-интерпретация результатов:

Для выявленных проблемных категорий:

- **Высокий returned_orders_count + высокий total_sales_loss**
 - Требуется срочное вмешательство - возможно, проблемы с качеством или несоответствием описанию
- **Высокий returned_orders_count + низкий total_sales_loss**
 - Проблемы с недорогими товарами - возможно, сложная установка или неясная инструкция
- **Низкий returned_orders_count + высокий total_sales_loss**
 - Крупные единичные возвраты - возможно, проблемы с доставкой дорогостоящих товаров

Практическое применение:

1. **Улучшение качества товаров** - фокус на категориях с наибольшим количеством возвратов

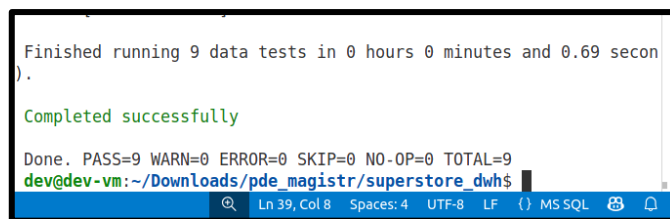
2. **Оптимизация описаний** - снижение несоответствия ожиданий и реальности
3. **Корректировка складских запасов** - уменьшение закупок проблемных подкатегорий
4. **Разработка политики возвратов** - целевые меры для высокорисковых категорий

Пример вывода для руководства:

"Подкатегория 'X' приносит наибольшие убытки от возвратов - 15% от общей выручки по категории. Рекомендуется аудит качества и пересмотр маркетинговых материалов."

Этот анализ превращает сырые данные о возвратах в стратегические инсайты для управления товарным портфелем.

Выполнение dbt test:



```
Finished running 9 data tests in 0 hours 0 minutes and 0.69 seconds.  
Completed successfully  
Done. PASS=9 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=9  
dev@dev-vm: ~/Downloads/pde_magistr/superstore_dwh$
```

Ключевые преимущества dbt для задачи анализа возвратов по сравнению с ручным написанием DDL/DML скриптов:

1. Автоматизация сложных JOIN-запросов

- dbt автоматически генерирует оптимальные SQL-запросы для соединения таблиц stg.returns, sales_fact и product_dim
- Ручные скрипты требуют сложного написания многотабличных JOIN с риском ошибок в условиях связей

2. Централизованное управление источниками данных

- В sources.yml четко определены все исходные таблицы (включая stg.returns), что обеспечивает прозрачность данных
- Ручные скрипты часто содержат разрозненные ссылки на таблицы без единого каталога

3. Автоматическое тестирование целостности данных

- dbt автоматически проверяет ключевые связи: order_id между sales_fact и stg.returns, product_id между фактами и

измерениями

- Ручные скрипты требуют отдельного написания проверок целостности, которые часто пропускаются

4. Визуализация сложных зависимостей

- Граф зависимостей в dbt docs наглядно показывает, как модель возвратов связана с исходными данными и другими моделями
- Ручные скрипты не предоставляют автоматической визуализации связей между таблицами

5. Стандартизация расчетов метрик

- Метрики `returned_orders_count` и `total_sales_loss` вычисляются по единому стандарту во всей организации
- Ручные скрипты часто приводят к разным формулам расчета в разных отчетах

6. Легкое повторное использование логики

- Логика анализа возвратов может быть легко переиспользована для других анализов (например, возвраты по регионам или клиентам)
- Ручные скрипты дублируют одну и ту же логику в разных местах

На примере нашего проекта анализа возвратов:

Автоматическое построение сложных связей - dbt сам управляет JOIN между возвратами, продажами и товарными категориями

Гарантия качества данных - тесты проверяют, что все возвращенные заказы существуют в основной таблице продаж

Прозрачность преобразований - граф зависимостей показывает полный путь от сырых возвратов до итогового анализа

Быстрая адаптация под изменения - при добавлении новых полей в `stg.returns` модель автоматически перестраивается

Согласованность между средами - один и тот же код работает в `dw_test` и `dw_student` без модификаций

Итог: dbt превращает сложный анализ возвратов из набора разрозненных SQL-запросов в управляемый, тестируемый бизнес-инструмент, который точно отражает финансовое воздействие возвратов на товарный портфель компании.