Презентация результата: Промышленный ETL-пайплайн с верификацией данных

1. **Архитектура решения и обзор**

Цель: Построение отказоустойчивого и проверяемого ETL-процесса для загрузки данных из разнородных Excel-файлов в нормализованное хранилище данных с полной историчностью.

**Технологический стек:**

· Orchestrator: Apache Airflow

· Обработка: Python (Pandas, SQLAlchemy, hashlib)

· Хранилище: PostgreSQL

· Верификация: Контрольные суммы MD5, проверка количества строк

**Архитектура данных:**

text

[Excel-файлы] -> [STG-слой (PostgreSQL)] -> [ODS-слой (SCD2)] -> [DDS-слой (Снежинка)]

· STG (Staging): Зеркало источника. Данные загружаются с проверкой хэша для гарантии идентичности.

· ODS (Operational Data Store): Слой историчности. Данные версионируются по алгоритму SCD Type 2.

· DDS (Data Distribution Store): Нормализованная аналитическая схема типа "снежинка". Готова для построения отчетов.

**DAG 1: stg\_load\_with\_verification**

Задачи DAG:

- Загружает Excel-файлы из /opt/airflow/data/input/ в таблицы STG.

Файлы:

- Товары.xlsx → stg\_products

- Транзакции.xlsx → stg\_transactions

- Торговые Точки.xlsx → stg\_stores

- Контрагенты.xlsx → stg\_counterparties

- Договор.xls → stg\_contracts

Для каждой загрузки:

- Чтение Excel (через openpyxl или xlrd).

- Подсчёт MD5-хэша (нормализованные строки).

- Очистка STG-таблицы. - Загрузка данных через pandas.to\_sql.

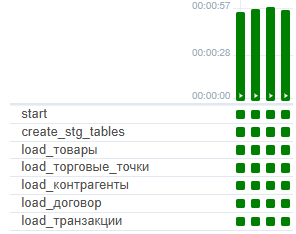
- Повторный подсчёт хэша.

- Проверка совпадения хэшей источника и STG.

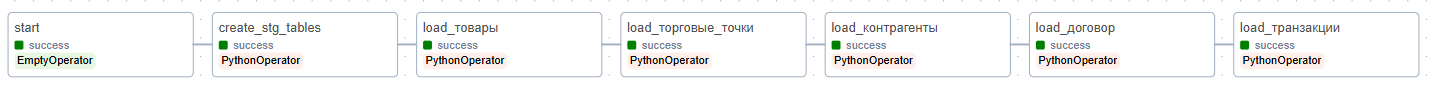
- Последовательность задач: Товары → Торговые Точки → Контрагенты → Договор → Транзакции.

- После успешной загрузки срабатывает TriggerDagRunOperator, запускающий DAG ods\_scd2\_load.

**Выполнение DAG:**



**Graph:**



Пример успешно выполненного лога:



**DAG 2: ODS слой и реализация SCD2**

Задача: Преобразовать плоские данные из STG в версионированные сущности с историчностью.

Ключевые особенности:

* Алгоритм SCD Type 2: Для каждой записи отслеживается период ее действительности (effective\_from, effective\_to, is\_current).
* Обработка изменений:
  + Новые записи: Вставляются с is\_current=TRUE.
  + Обновленные записи: Старая версия закрывается (is\_current=FALSE, effective\_to=NOW()), новая – вставляется.
  + Удаленные записи: Помечаются как неактуальные (закрывается effective\_to).
* Параллельная обработка: Все ODS-таблицы обновляются параллельно для скорости.

Задачи DAG:

* Загружает данные из STG в ODS-таблицы с реализацией SCD2.

Соответствия:

* + stg\_products → ods\_products (ключ: КодТовара)
  + stg\_transactions → ods\_transactions (ключ: №строки)
  + stg\_stores → ods\_stores (ключ: КодТорговойТочки)
  + stg\_counterparties → ods\_counterparties (ключ: КодКонтрагента)
  + stg\_contracts → ods\_contracts (ключ: КодТорговойТочки)

Алгоритм:

1. Чтение данных из STG.

2. Чтение текущих записей (is\_current = TRUE) из ODS.

3. Определение новых и удалённых ключей.

4. Пометка удалённых записей: effective\_to = NOW(), is\_current = FALSE.

5. Добавление новых записей: генерация surrogate key, выставление effective\_from = NOW(), effective\_to = 9999-12-31, is\_current = TRUE.

Все таблицы обрабатываются параллельно.

SQL-создание ODS-таблиц (create\_ods\_tables.sql):

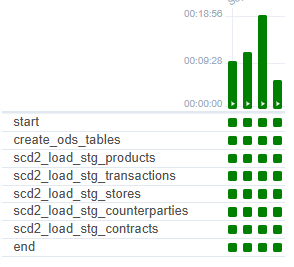
* + ods\_products: товары с SCD2.
  + ods\_transactions: транзакции с SCD2.
  + ods\_stores: торговые точки с SCD2.
  + ods\_counterparties: контрагенты с SCD2.
  + ods\_contracts: договоры с SCD2.

Для каждой таблицы добавлены индексы по ключам и is\_current.

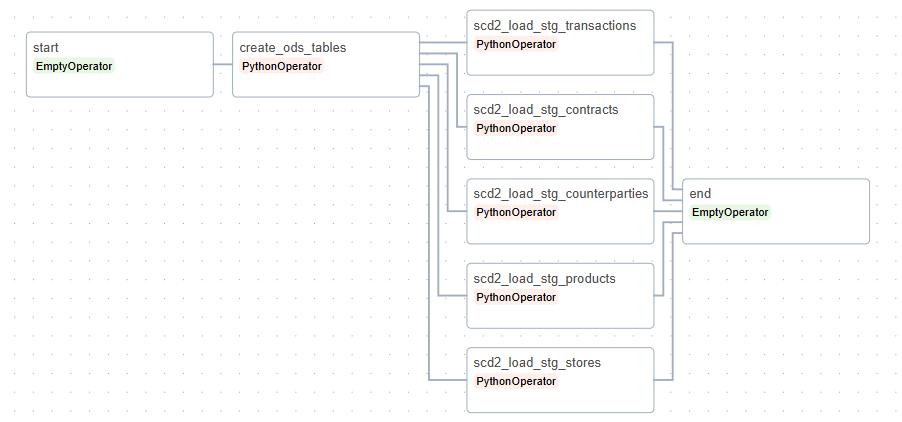
**Составные бизнес-ключи**

Для некоторых таблиц ODS были использованы составные бизнес-ключи, объединяющие несколько колонок для однозначной идентификации записей. Это позволило корректно переносить данные из STG, отслеживать изменения и избегать дублирования.

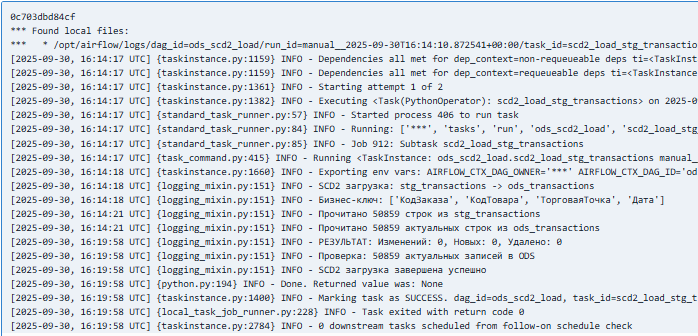
**Выполнение DAG:**



**Graph:**

****

**Пример успешно выполненного лога (повторно):**



**DAG 3 - DDS слой (нормализованная "снежинка")**

Задача: Построить оптимальную схему для аналитических запросов.

Ключевые особенности:

* Нормализация: Данные разбиты на логические сущности (измерения): product, store, counterparty и т.д.
* Фактовая таблица: transaction содержит только ключи измерений и числовые показатели (выручка, себестоимость, кол-во).
* Последовательная загрузка: Сначала загружаются независимые измерения (manufacturer), затем зависящие от них (brand, product\_group), и в конце – факты.

Задачи DAG:

Создание DDS-таблиц (PostgresOperator, create\_dds\_tables.sql).

Проверка создания таблиц (dds\_check\_tables).

Последовательная загрузка измерений и фактов (через SQL-скрипты):

* load\_manufacturer.sql
* load\_brand.sql
* load\_product\_group.sql
* load\_product\_line.sql
* load\_product.sql
* load\_city.sql
* load\_counterparty.sql
* load\_contract.sql
* load\_store.sql
* load\_supplier.sql
* load\_branch.sql
* load\_manager.sql
* load\_deal\_type.sql
* load\_delivery\_type.sql
* load\_transaction\_type.sql
* load\_orders.sql
* load\_transactions.sql

SQL-создание DDS-таблиц (create\_dds\_tables.sql):

Измерения:

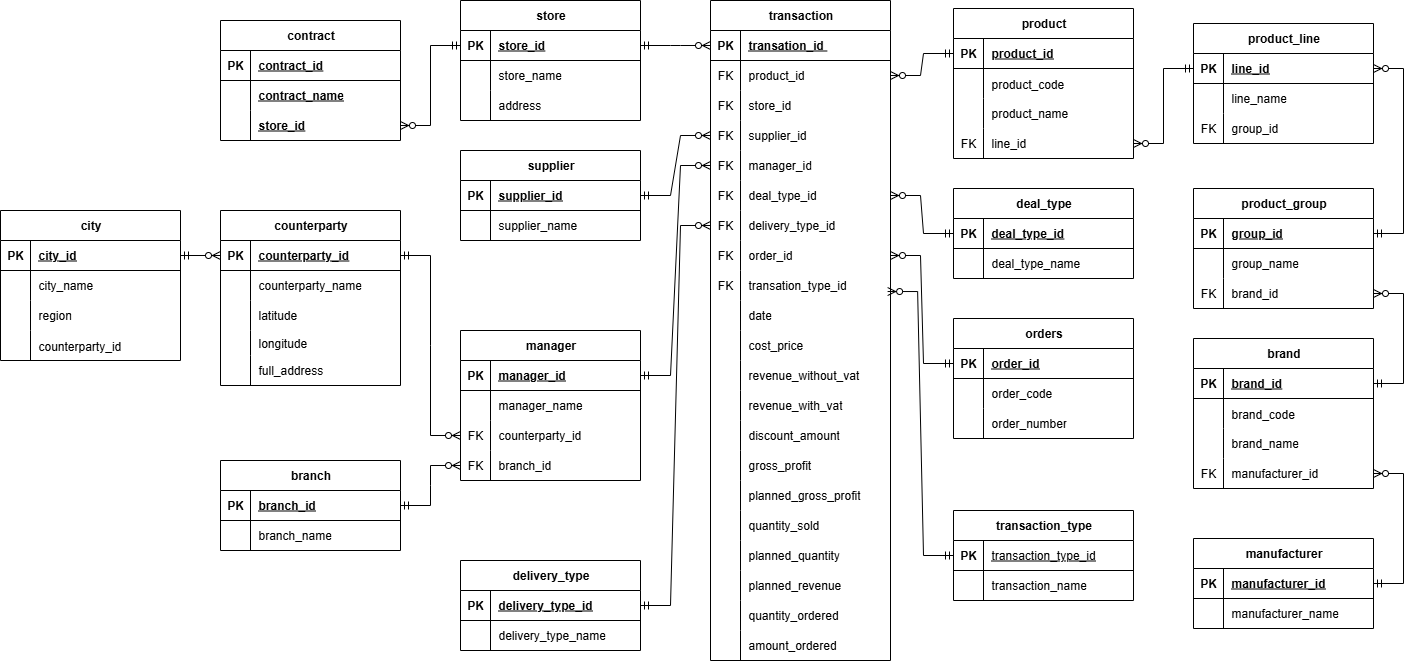
manufacturer, brand, product\_group, product\_line, product, city, counterparty, branch, store, supplier, manager, deal\_type, delivery\_type, transaction\_type, orders, contract

Факт:

transactions — содержит связи по surrogate keys на измерения и показатели: себестоимость, выручка, скидка, валовый доход и т.п.

Оптимизация: добавлены индексы по бизнес-ключам и связям для ускорения аналитических запросов.

Схема связей:



Общая структура

Схема представляет собой нормализованную модель транзакций розничной торговли, соответствующую требованиям третьей нормальной формы (3НФ). Она обеспечивает структурированное хранение данных с разделением сущностей и минимизацией дублирования, что упрощает сопровождение, расширение и построение аналитических витрин.

**Ключевые связи** (и типы связей)

**Один-ко-многим** (1:N)

* product → transaction: один товар может участвовать в многих транзакциях.
* store → transaction: одна точка продаж → много транзакций.
* supplier → transaction: один поставщик может быть источником многих транзакций.
* manager → transaction: один менеджер может вести несколько транзакций.
* deal\_type → transaction: один тип сделки — множество применений.
* delivery\_type → transaction: одна доставка — много применений.
* order → transaction: один заказ → несколько строк транзакции.
* transaction\_type → transaction: один тип транзакции — много записей.

**Иерархии**

* product → product\_line → product\_group → brand → manufacturer

Эта цепочка описывает иерархическую классификацию товара: от конкретного товара до производителя.

* manager → counterparty и branch

Менеджер относится к определённой ветке организации и контрагенту.

* city → region (через поле region)

**Нормализация**

* Факты (transaction) отделены от справочников (product, store, supplier и т.д.)
* Нет избыточности в хранении текстовых значений: используются surrogate keys

**Возможности аналитики**

На основе схемы можно строить:

* Выручку и прибыль по:
  + товарам, группам, брендам, производителям
  + магазинам, регионам
  + менеджерам, веткам, контрагентам
  + типам сделок и доставок
* Анализ исполнения заказов (сравнение фактических и плановых значений)
* Кросс-фильтрацию: например, как контрагенты в разных городах работают по разным контрактам

**Пример аналитического запроса:**

SELECT

b.brand\_name,

SUM(t.revenue\_with\_vat) AS total\_revenue,

SUM(t.discount\_amount) AS total\_discount,

SUM(t.gross\_profit) AS total\_profit

FROM transaction t

JOIN product p ON t.product\_id = p.product\_id

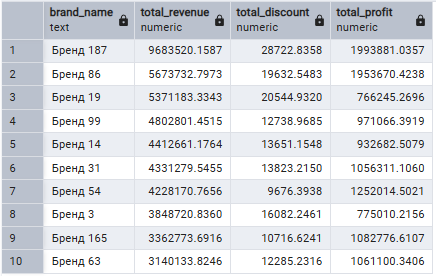
JOIN product\_group pg ON p.line\_id = pg.group\_id

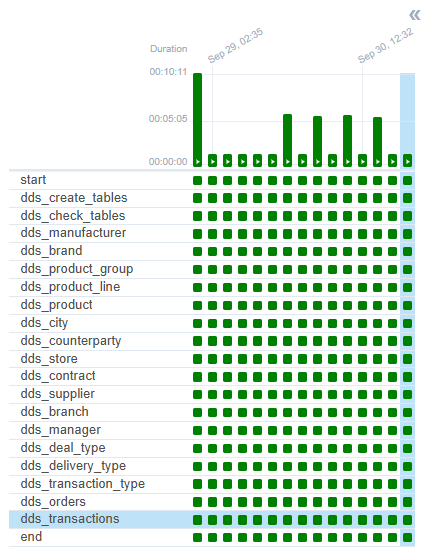
JOIN brand b ON pg.brand\_id = b.brand\_id

GROUP BY b.brand\_name

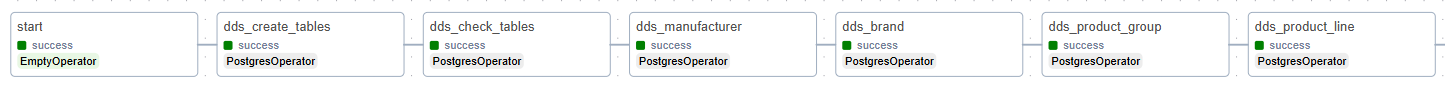
ORDER BY total\_revenue DESC

LIMIT 10;

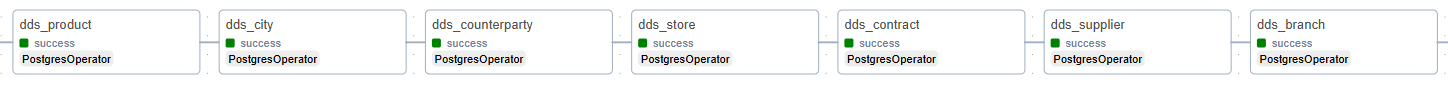
**Результат запроса:  
**

**Выполнение третьего DAG:  
**

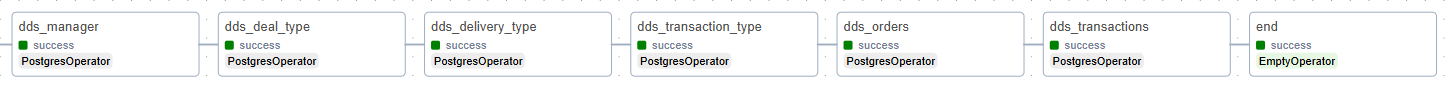
**Graph:**

****

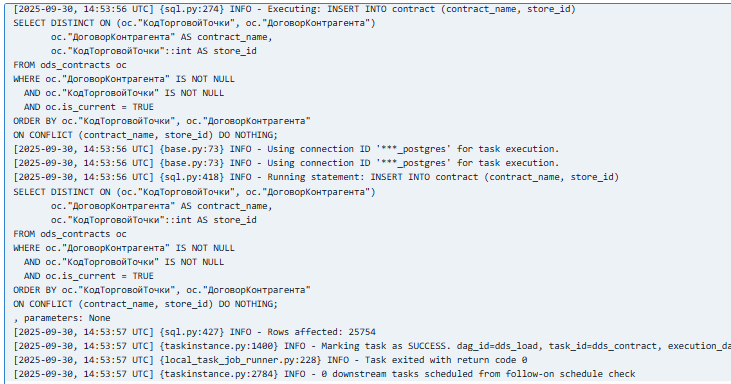
**→ → →**

****

**→ → →**

****

**Пример успешно выполненного лога:**

****

Тест-кейс 1: Верификация целостности при STG-загрузке

Результат: ТЕСТ ПРОЙДЕН УСПЕШНО

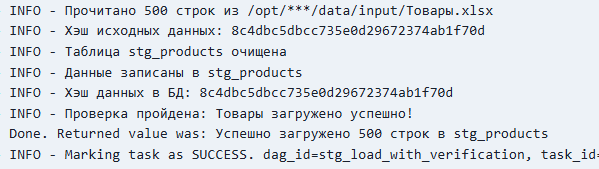
Цель тестирования:

Убедиться, что данные не искажаются при переносе из Excel в STG-слой БД и механизм проверки хэшей работает корректно.

Выполненные действия:

1. Запущен DAG stg\_load\_with\_verification

→Получен исходный хэш данных: 8c4dbc5dbcc735e0d29672374ab1f70d

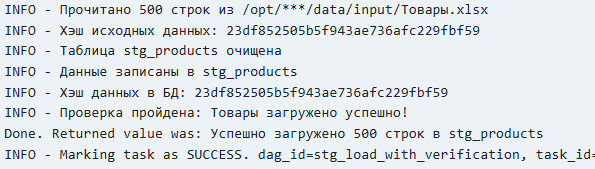


1. Изменение исходных данных

В файле Товары.xlsx изменено значение ячейки "ТЕСТ\_ДЛЯ\_ОШИБКИ\_ХЭША" на "Бренд 1"

1. Запущен DAG с измененным файлом

→Получен новый хэш данных: 23df852505b5f943ae736afc229fbf59



Анализ результатов

Что подтверждено тестом:

* Целостность данных - данные переносятся без искажений
* Реакция на изменения - система обнаруживает изменения в источнике
* Работоспособность DAG - процесс загрузки выполняется корректно
* Механизм хэширования - хэш-функция чувствительна к изменениям

Особенности реализации:

* STG-слой полностью перезаписывается при каждой загрузке
* Проверка выполняется сравнением хэшей исходного файла и загруженных данных
* При изменении источника система корректно обновляет STG-слой

Оформление отчета по тест-кейсу 2: Проверка историчности SCD2 в ODS-слое

Тест-кейс 2: ПРОВЕРКА ИСТОРИЧНОСТИ SCD2

Цель тестирования:

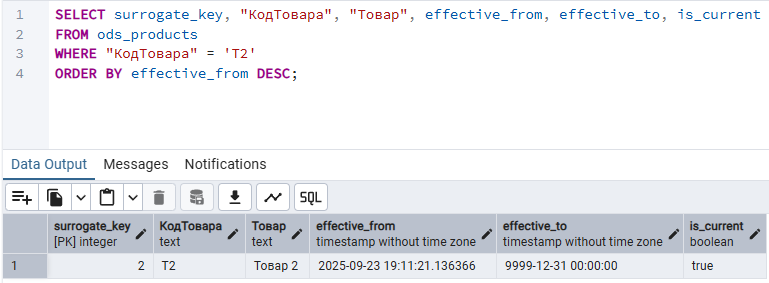
Подтвердить стабильность работы механизма SCD2 при изменении данных в источнике на примере товара T2, убедиться в корректном сохранении историчности данных.

Выполненные действия:

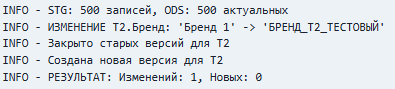
1. Подготовка тестовой среды
   * Обновлен STG-слой через DAG stg\_load\_with\_verification
   * Проверено исходное состояние товара T2 в ODS-слое
   * Убедились в отсутствии дублирования актуальных записей
2. Модификация тестовых данных
   * Выбран товар с КодТовара = 'T2' для тестирования
   * Обновлено название товара: 'Товар 2' → 'ТОВАР\_T2\_ИСТОРИЧНЫЙ\_v2'
   * Изменен бренд товара: 'Бренд 2' → 'БРЕНД\_T2\_ТЕСТОВЫЙ'
3. Запуск SCD2-процесса
   * Выполнен DAG ods\_scd2\_load
   * Система обнаружила изменения атрибутов товара
   * Корректно применила механизм историчности SCD2

Исходное состояние ODS

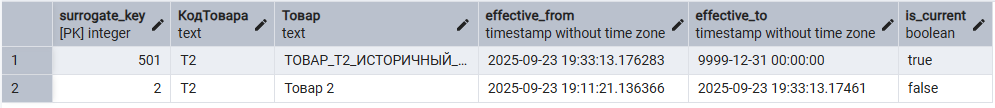
(до изменений - дублирование записей)



SCD2 загрузка: stg\_products -> ods\_products



Конечное состояние ODS



Анализ результатов

Что подтверждено тестом:

* Стабильность механизма SCD2 - система корректно обрабатывает изменения для разных записей
* Историчность сохраняется - старые версии данных сохраняются с правильными временными метками
* Корректность актуальности - только одна запись помечена is\_current = TRUE
* Работа с атрибутами - система обнаруживает изменения в полях данных (не только в ключах)

Вывод

Требования выполнены:

* Данные на ODS хранят историчность SCD2 для различных сценариев изменений
* При изменениях атрибутов создаются новые версии записей
* Старые версии корректно помечаются как неактуальные
* Актуальные данные соответствуют источнику после изменений

Тест-кейс 3: Верификация нормализации в DDS-слое (снежинка-модель)

ЦЕЛЬ: Проверить корректность загрузки данных в DDS и работу снежинка-модели

1. Проверка создания всех таблиц DDS

Пишем SQL запрос, в котором узнаем, создались ли нужные нам таблицы:

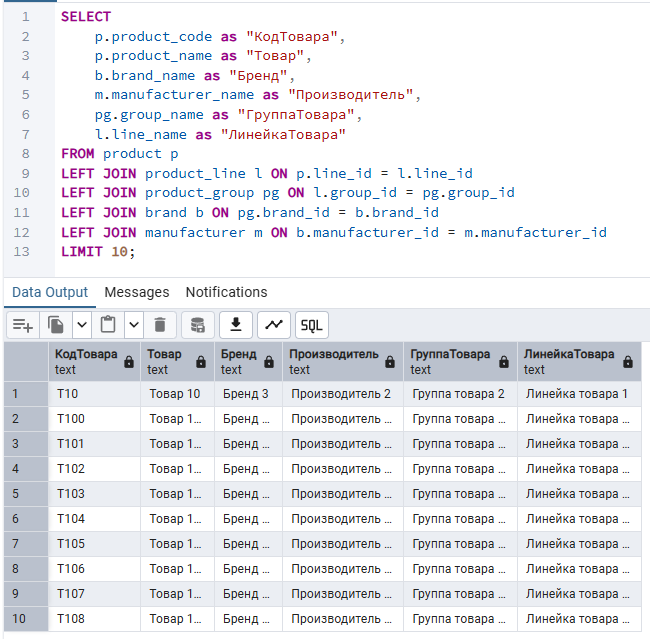
Получаем результат:



Результат: созданы все 17 таблиц для нашей модели.

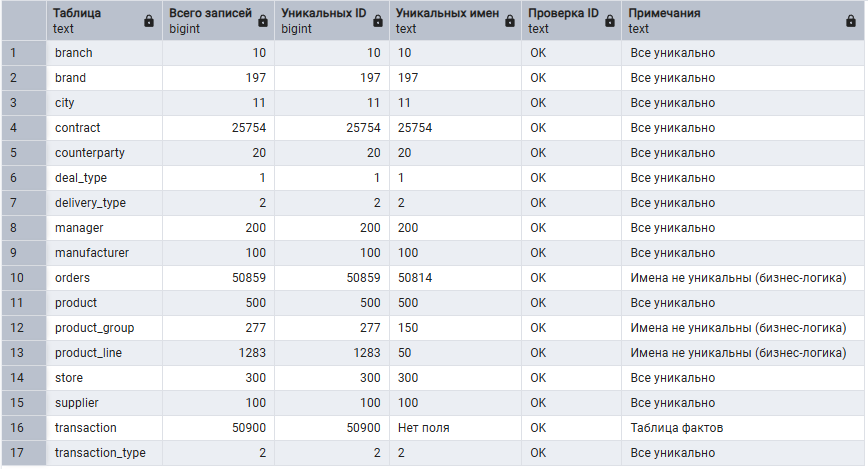
1. Проверка нормализации товаров (снежинка)

Пишем SQL запрос для проверки нормализации товаров со множественными джойнами:



Результат: данные нормализованы правильно.

1. Проверка целостности данных и отсутствия дубликатов



Результат:

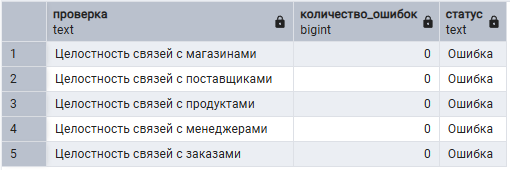
Дубликаты отсутствуют, все поля сверки совпадают, за исключением таблицы фактов transaction, у которой нет поля с наименованием транзакций, а также product\_line и product\_group, но это обусловлено бизнес-логикой, поскольку у каждого бренда могут быть одинаковые линии и группы товаров.   
При проверке уникальности кодов заказов (order\_code) выявлены дубли: один и тот же код может встречаться несколько раз с разными внутренними идентификаторами (order\_id). Это связано с тем, что один заказ может состоять из нескольких частей или обрабатываться в несколько этапов.

1. Проверка корректности построения DDS (Data Delivery System) слоя и целостности бизнес-связей между сущностями "contract" и "store".

## Результаты тестирования:

* DDS слой построен корректно
* Все связи между сущностями работают правильно
* Данные загружены полностью и без ошибок
* Бизнес-логика отражена адекватно

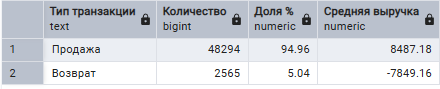
1. Проверка фактовой таблицы transaction
2. Проверка целостности связей со справочниками



Результат:

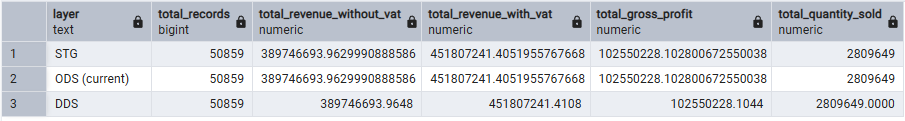
Все связи фактовой таблицы с справочниками корректны. Отсутствуют битые ссылки и нарушения целостности данных.

1. Проверка распределения по типам операций (отдельным запросом)



* Полная целостность данных - все внешние ключи работают корректно
* Качественные связи с продуктами, магазинами, поставщиками, менеджерами и заказами
* Отсутствие битых ссылок в фактовой таблице

Тест-кейс 4: Сквозное тестирование всего ETL-пайплайна  
  
1. Результаты теста консистентности данных:



Результат:

1. Количество записей совпадает во всех слоях — 50 859.
2. Агрегированные показатели выручки и валового дохода отличаются только на доли единиц после округления (до 4 знаков после запятой). Эти расхождения несущественны и вызваны округлением чисел после конвертации из текстового формата с запятой в числовой тип.
3. Объем продаж совпадает полностью.

Итог:  
 Данные корректно переносились и агрегируются через все слои ETL. Отличия минимальны и находятся в пределах допустимого округления. Система готова к использованию для аналитики.

## **ЗАКЛЮЧЕНИЕ**

### **Достигнутые результаты**

1. Полностью автоматизированный промышленный ETL-пайплайн

* Реализован отказоустойчивый трехслойный процесс загрузки данных
* Обеспечена полная историчность данных через SCD Type 2
* Построена нормализованная аналитическая модель "снежинка"

2. Гарантированная целостность данных

* Внедрена система верификации на основе MD5-хэшей
* Реализованы проверки консистентности между слоями
* Обеспечено отсутствие потерь и искажений данных

3. Промышленная надежность

* Обработка ошибок и повторные попытки выполнения
* Параллельная обработка таблиц для оптимизации производительности
* Мониторинг качества данных на каждом этапе

4. Масштабируемая архитектура

* Четкое разделение ответственности между слоями (STG/ODS/DDS)
* Возможность легкого добавления новых источников данных
* Гибкая модель данных для будущего расширения