Потоковый конвейер данных интернет-магазина

Цель

Построить end-to-end конвейер потоковой обработки данных для интернет-магазина: от PostgreSQL через CDC и Spark Structured Streaming до аналитических витрин с инкрементальными dbt-моделями. Нужно реализовать полный Kappa-конвейер, охватывающий все ключевые темы лекции: Event Time, Watermarks, Windowing, гарантии доставки, CDC, табличные форматы и compaction.

Исходные данные

- **Генератор данных** (data_generator/) — эмулятор интернет-магазина с PostgreSQL и JSONL-логами
- Docker Compose с PostgreSQL 16, pgAdmin, MinIO (S3-совместимое хранилище)
- **Генератор запускается в режиме Streaming** (python main.py --sleep-seconds 1), продвигая виртуальное время на 1 час за секунду реального времени

Схема БД описана в data_generator/database.md.

Задание 1a. Развёртывание инфраструктуры потокового конвейера

Что нужно сделать

Добавить в docker-compose.yml недостающие компоненты потоковой архитектуры: **Redpanda** (message broker) и **Debezium** (CDC-движок). Настроить CDC-коннекторы Debezium для захвата изменений из всех таблиц схем ecommerce и wms PostgreSQL.

Что проверить

- **Запустить генератор в streaming-режиме** (python main.py --sleep-seconds 1)
- **Убедиться, что в топиках Redpanda появляются сообщения:** rpk topic consume ecom.ecommerce.orders
- Убедиться, что CDC-сообщения содержат поля before и after (для UPDATE — оба, для INSERT — только after, для DELETE — только before)
- **Проверить через** rpk topic list наличие всех ожидаемых топиков

Ожидаемые результаты

- Файл docker-compose.yml с Redpanda и Debezium
- Конфигурационные файлы коннекторов Debezium (JSON) — отдельно или скрипт register_connectors.sh
- **Скриншот/лог:** rpk topic consume ecom.ecommerce.orders показывает CDC-сообщения с изменяющимися статусами заказов

Ссылка на сделанную преподавателем работу:
https://owncloud.nano.sfedu.ru/index.php/s/xRffJAL975Po6tL

Задание 1b. Стриминг файловых логов в Redpanda

Модифицируйте генератор так, чтобы он стримил файловые логи clickstream и crm в Redpanda.

Задание 2. Spark Structured Streaming — чтение CDC и запись в Delta (Bronze Layer)

Что нужно сделать

Написать PySpark-приложение, которое читает CDC-сообщения из Redpanda, парсит JSON-структуру Debezium (поля before, after, op, source) и записывает их в Delta-таблицы в MinIO. Реализовать bronze-слой: сырые CDC-записи сохраняются «как есть» для возможности полного пересчёта истории (replayability в духе Kappa-архитектуры).

Что проверить

- **Delta-таблицы создаются в MinIO по пути** s3a://datalake/bronze/
- Запустить генератор, подождать 10-15 шагов, остановить Spark-стрим, перечитать таблицу в batch-режиме: записи соответствуют тому, что есть в PostgreSQL
- CDC-сообщения типа u (update) корректно сохраняют и before, и after
- **Event Time из** source.ts_ms корректно преобразован в timestamp

Ссылка на сделанную преподавателем работу:
https://owncloud.nano.sfedu.ru/index.php/s/xRffJAL975Po6tL

Задание 3а. Accumulating Snapshot — жизненный цикл заказа (Silver Layer)

Мотивация: почему batch здесь бессилен

В оперативной БД статус заказа обновляется на месте (UPDATE orders SET status = ...). Batch-запрос видит только последнее состояние строки. Узнать, когда заказ был оплачен, сколько времени заняла сборка на складе или когда он был передан в доставку, невозможно — история переходов затёрта. CDC-поток сохраняет каждое изменение как отдельное событие с полями before и after. Streaming-обработка способна собрать эти разрозненные события в полную картину жизненного цикла заказа. Это задача, где streaming не просто «быстрее», а принципиально необходим.

Что нужно сделать

В bronze-слое каждая смена статуса заказа — отдельная CDC-запись (строки с op = 'c' для создания и op = 'u' для каждого обновления статуса). Задача: собрать их в **Accumulating Snapshot** — одну строку на заказ с временáми всех переходов и длительностями каждого этапа. Указание: использовать **stateful-обработку** в Spark Structured Streaming (flatMapGroupsWithState или mapGroupsWithState).

Что проверить

- **time_to_pay + time_to_ship + time_to_deliver = delivered_at − created_at** (с точностью до секунды)
- **Все заказы из bronze присутствуют в silver:** число уникальных order_id в bronze с op = 'c' = число строк в order_lifecycle
- Отсутствуют «зависшие» заказы в состоянии (статус не DELIVERED и не CANCELLED после остановки генератора)
- **Среднее время доставки — разумное число** (в генераторе SHIPPED_TO_DELIVERED от 1 до 5 виртуальных дней)

Задание 3b. Оконные агрегации с Event Time и Watermarks (Silver Layer)

Что нужно сделать

Поверх bronze-слоя построить silver-агрегаты с корректной обработкой Event Time, окон и watermark'ов. Реализовать:

- **Tumbling-окна:** количество заказов каждые 5 минут по Event Time
- **Sliding-окна:** скользящая сумма выручки за 30 минут с шагом 5 минут
- **Session-окна:** сессии пользователей из clickstream (разрыв > 15 минут)

Что проверить

- **Tumbling:** Запустить генератор на 30 виртуальных дней (в режиме bulk или streaming). Проверить, что количество заказов в каждом 5-минутном окне корректно — нет отрицательных значений, разумные числа
- **Sliding:** Визуализировать скользящую выручку (например, через DuckDB + matplotlib). График должен быть сглаженным (перекрытие окон сглаживает пики)
- **Session:** Проверить, что сессии корректно закрываются после gap > 15 минут. Сравнить общее число checkout'ов по сессиям с числом созданных заказов в orders
- **Watermark:** Специально задержать несколько событий (например, вручную отправить в Redpanda события с Event Time на 20 минут назад) и убедиться, что watermark их отбросил (проверить через лог Spark'a или side output)

Задание 4. At-Least-Once, Deduplication и Exactly-Once семантика

Что нужно сделать

При сбое Spark-стрима и перезапуске с checkpoint'a некоторые микробатчи могут быть перечитаны из Redpanda повторно. Без дедупликации в bronze-слое появятся дубликаты. Нужно реализовать защиту от дубликатов в bronze-слое и обеспечить exactly-once семантику при записи в silver-слой. Продемонстрировать разницу между at-least-once (без дедупликации) и exactly-once (с дедупликацией).

Что проверить

- После перезапуска стрима число уникальных строк в дедуплицированной таблице = числу строк в не-дедуплицированной минус дубликаты
- Delta-таблица с дедупликацией при повторном запуске не меняет количество строк (идемпотентность)
- **DESCRIBE HISTORY delta.**bronze_orders_dedup — видно, что повторные микробатчи создают транзакции, но не добавляют новых строк (пустые коммиты)

Задание 5. dbt — инкрементальные модели, SCD Type 2 и MERGE

Что нужно сделать

Настроить dbt-проект поверх Delta-таблиц bronze- и silver-слоя, реализовать инкрементальные модели с MERGE-стратегией и построить Slowly Changing Dimension

Type 2 для уровня лояльности клиентов. Трансформировать данные в две аналитические витрины:

- ежедневные продажи с накоплением для каждого клиента
- операционные метрики жизненного цикла заказов

Что проверить

- Запустить dbt run несколько раз подряд — инкрементальные модели не пересчитывают уже обработанные данные
- **SCD Type 2:** для клиента, повысившего лояльность с BRONZE до SILVER, в таблице ровно 2 строки (старая закрыта, новая текущая), valid_to старой = valid_from новой
- **dbt test** — тесты на уникальность unique_key, not-null колонок
- **Сравнить SELECT COUNT(*) в bronze и silver** — числа должны совпадать (с точностью до дедупликации)
- **Витрина 2:** avg_total_hours совпадает с прямым запросом к order_lifecycle
- **Витрина 2:** sla_breach_pct в диапазоне 0–100%

Задание 6. Compaction и аналитика DuckDB

Что нужно сделать

Обнаружить проблему мелких файлов в Delta-таблицах, сгенерированных Spark Structured Streaming (каждый микробатч создаёт новые Parquet-файлы). Реализовать compaction через OPTIMIZE и продемонстрировать улучшение производительности запросов. Построить аналитический слой на DuckDB:

- График 1: количество заказов по дням (из ежедневной витрины)
- График 2: скользящая выручка за 30 минут (из sliding window silver)
- График 3: распределение длительности сессий (гистограмма)
- График 4: среднее время доставки по дням + доля SLA-нарушений (из операционной витрины)

Что проверить

- **OPTIMIZE сокращает numFiles кратно** (например, с 700 до 7 файлов)
- Время выполнения тестового запроса уменьшается после compaction (цель — минимум в 2-3 раза)
- **После VACUUM старые файлы физически удалены из MinIO** (проверить через mc ls или через MinIO browser на порту 9001)
- Графики DuckDB строятся без ошибок
- Time-travel всё ещё работает для недавних версий (после VACUUM с retention 1 час старые версии старше часа должны быть недоступны)