Лабораторная работа 2. Этап 2: Dimensional Modeling и магия Star Schema

Введение

На первом этапе вы прочувствовали боль аналитиков, пытающихся извлечь инсайты напрямую из сырых нормализованных таблиц (3NF) и логов. Сложные джойны, оконные функции для поиска актуальных состояний (Point-in-Time) и запутанная бизнес-логика делают такие запросы медленными и подверженными ошибкам.

На этом этапе мы применим методологию **Dimensional Modeling (моделирование Кимбалла)**. Мы перенесем сложность вычислений с этапа *чтения* (когда аналитик пишет `SELECT`) на этап *записи* (когда дата-инженер строит ETL-пайплайн).

Архитектура этапа

Мы построим классическую схему "Звезда" (Star Schema), состоящую из измерений (Dimensions) и таблиц фактов (Fact Tables).

Задание

Задача 1: Измерение со Slowly Changing Dimensions (SCD Type 2)

Создайте модель `dim_customer`. Это самое сложное измерение, так как оно должно объединить данные из двух разных систем:

- `ecommerce.customers` (имя, фамилия, телефон)
- `crm.loyalty_updates` (уровень лояльности, баланс баллов)

Требования:

- Связь между системами осуществляется только по полю `email` (это наш бизнес-ключ).
- Измерение должно поддерживать версионирование (SCD Type 2). При любом изменении уровня лояльности в логах CRM, старая запись должна закрываться (`valid_to`), а новая открываться (`valid_from`, `is_current = TRUE`).
- У каждого клиента должен появиться суррогатный ключ (`customer_sk`).

Задача 2: Transaction Fact Table

Создайте транзакционную таблицу фактов `fact_order_items`.

- **Зерно (Grain):** 1 строка = 1 товар в заказе.
- **Ключи:** `product_id, order_id (измерений product, order пока нет, поэтому нужно использовать натуральные ключи), customer_sk .`

- **Важно:** При джойне с `dim_customer` необходимо использовать суррогатный ключ клиента *именно на момент совершения заказа* (Point-in-Time join переносится в ETL!).
- **Метрики:** `quantity`, `base_price`, `discount_amount`, `final_price`.

Задача 3: Accumulating Snapshot Fact Table

Создайте таблицу фактов с накапливающимся снимком `fact_order_fulfillment`.

- **Зерно (Grain):** 1 строка = 1 заказ.
- **Ключи:** `customer_sk`, `date_created_sk`, `date_paid_sk`, `date_shipped_sk`, `date_delivered_sk`.
- **Метрики:** Время (в часах) между этапами: `lag_created_to_paid`, `lag_paid_to_shipped`, `lag_shipped_to_delivered`.
- **Суть:** Эта таблица обновляется (UPDATE/MERGE) по мере того, как заказ проходит свой жизненный цикл.
- **Подвох**: в учебных данных отсутствует реальная история изменений статусов заказов в PostgreSQL. Таблица `orders` содержит только текущее состояние каждого заказа без истории изменений. В production-среде accumulation snapshot таблицы обычно строятся на основе CDC или логов. **Как с этим быть - неизвестно!**

Задача 4: Periodic Snapshot Fact Table

Создайте таблицу фактов периодического снимка `fact_daily_inventory`.

- **Зерно (Grain):** 1 строка = 1 товар на 1 складе на конец каждого дня.
- **Источник:** Append-only таблица `wms.inventory_transactions`.
- **Суть:** Вам нужно свернуть историю транзакций (приходы и расходы) в ежедневные остатки. Если в какой-то день транзакций не было, остаток должен переноситься с предыдущего дня (forward fill).

Задача 5: Анализ и рефакторинг

1. **Перепишите бизнес-запросы из Этапа 1**, используя новые таблицы фактов и измерения.
2. **Сравните код:** Насколько проще стали запросы? Исчезли ли оконные функции из аналитических запросов?
3. Проанализируйте логику SCD2 и генерацию календаря для Periodic Snapshot.