ООО «Нетология»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к дипломному проекту

по профессии «Инженер данных»

Студент: Тиунов Арсений Юрьевич

Группа: DEG-36

2025г.

Содержание

1. **Введение**  
   1.1. Актуальность и цель работы  
   1.2. Задачи и этапы исследования  
   1.3. Краткое описание инструментов и технологий
2. **Описание исходного датасета**  
   2.1. Структура CSV: поля, типы данных  
   2.2. Анализ полноты и качества данных: пропуски, дубликаты, некорректные значения, форматы дат
   1. Проверка хэша cvs-файла на изменение и решение о его загрузке
3. **Проектирование нормализованного хранилища (NDS)**  
   3.1. Выбор ключевых сущностей и атрибутов NDS  
   3.2. Логическая ER-диаграмма DDS
4. **Схема звезды (DDS) и витрины**  
   4.1. Выбор ключевых сущностей и атрибутов DDS  
   4.2. Логическая ER-диаграмма DDS
5. **Оркестрация ETL-процессов**
6. **Процессы загрузки и трансформации данных**
7. **Построение дашбордов в Tableau**
8. **Выводы по дипломному проекту**

**Приложения:**  
1. DAG-файлы  
2. Пояснительная записка к дашбордам Sales Data Mart (Tableau)  
3. ER-диаграммы и таблицы хранилища данных

# 1. Введение

## 1.1. Актуальность и цель работы

Считаю, что в условиях стремительного роста объемов данных и необходимости оперативного принятия решений - эффективная организация процессов ETL (Extract, Transform, Load) становится ключевым фактором успешной работы аналитических систем. Целью данной работы является разработка и документирование процессов ETL на основе предложенного датасета «sales.csv» для построения надежного и масштабируемого хранилища данных.

## 1.2. Задачи и этапы исследования

В рамках дипломного проекта поставлены следующие задачи:  
- Обработка и предварительный анализ качества исходных данных (пропуски, дубликаты, формат).  
- Проектирование нормализованного хранилища NDS и схемы звезда DDS.  
- Реализация ETL-процессов для заливки данных в NDS и формирования витрин.  
- Оркестрация процессов с помощью Apache Airflow и контейнеризация через Docker.  
- Формулирование выводов на основе полученных результатов.

## 1.3. Краткое описание инструментов и технологий

Для выполнения проекта использовались следующие инструменты и технологии:  
- Draw.io для построения ER-диаграмм и визуализации схемы данных. - Python (pandas, SQL) для обработки и трансформации данных;  
- ClickHouse в качестве аналитической базы данных для хранения NDS и DDS;  
- Apache Airflow для оркестрации ETL-джобов;  
- Docker для контейнеризации окружения и обеспечения воспроизводимости;  
- DBeaver для управления базой данных и проверки корректности выполнения SQL-запросов;  
- Tableau для создания интерактивных дашбордов и отчетности;

# 2. Описание исходного датасета

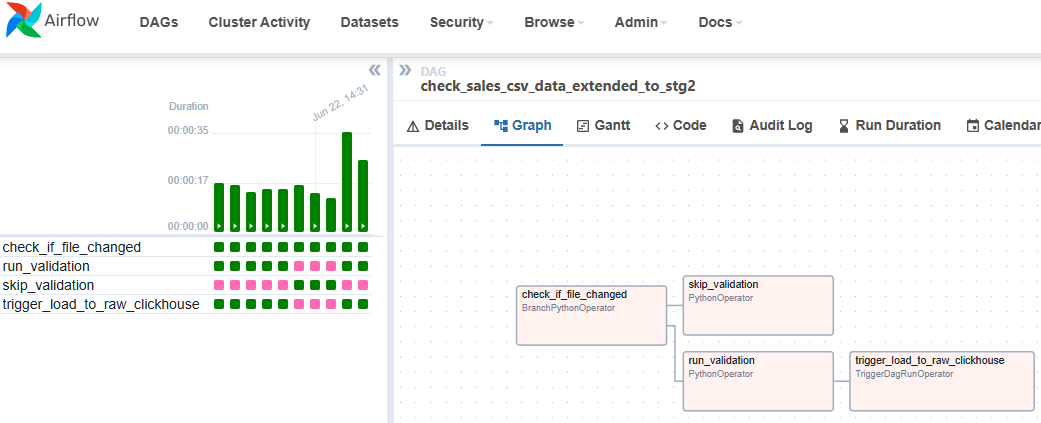
## 2.1. Структура CSV-файла «sales»

Исходный файл представлен в формате CSV и содержит следующие поля:

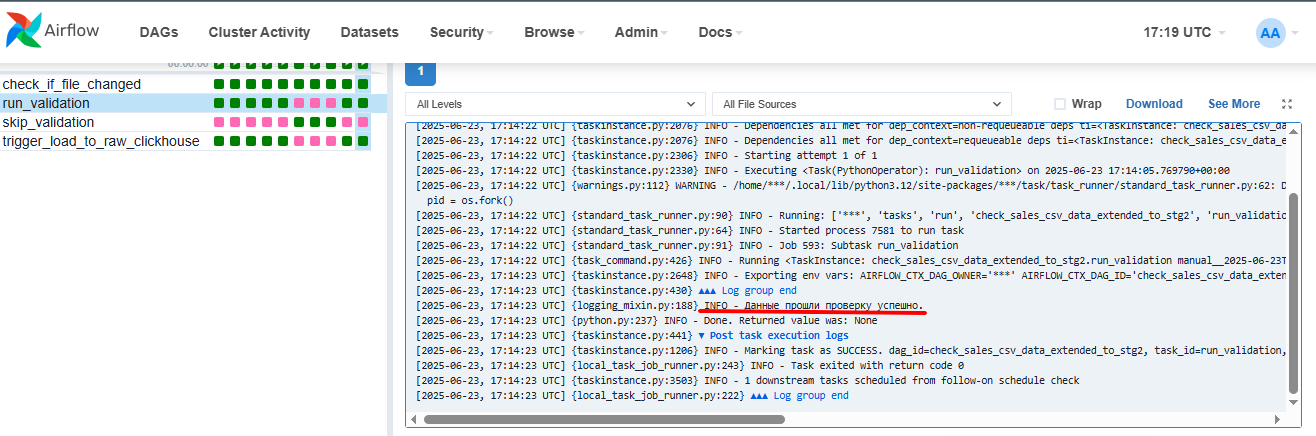
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Система-источник** | **Таблица** | **Атрибут** | **Тип атрибута** | **Является первичным ключом** | **Комментарий к атрибуту** |
| sales.csv | sales | Invoice ID | string | Да | Уникальный идентификатор чека (продажи) |
| sales.csv | sales | Branch | string | Нет | Обозначение филиала: A, B, C |
| sales.csv | sales | City | string | Нет | Город, где находится филиал (напр. Yangon, Mandalay) |
| sales.csv | sales | Customer type | string | Нет | Тип клиента: Member (постоянный) или Normal (разовый) |
| sales.csv | sales | Gender | string | Нет | Пол клиента: Male или Female |
| sales.csv | sales | Product line | string | Нет | Категория товара (например: Health and beauty, Electronic accessories) |
| sales.csv | sales | Unit price | float | Нет | Цена за единицу товара |
| sales.csv | sales | Quantity | integer | Нет | Количество приобретённого товара |
| sales.csv | sales | Tax 5% | float | Нет | Налог 5% от суммы без учёта налога |
| sales.csv | sales | Total | float | Нет | Общая сумма покупки (включая налог) |
| sales.csv | sales | Date | date | Нет | Дата покупки (в формате mm/dd/yyyy в источнике) |
| sales.csv | sales | Time | time (string) | Нет | Время покупки |
| sales.csv | sales | Payment | string | Нет | Способ оплаты: Cash, Credit card, Ewallet |
| sales.csv | sales | cogs | float | Нет | Себестоимость проданных товаров (Cost Of Goods Sold) |
| sales.csv | sales | gross margin percentage | float | Нет | Валовая маржа в процентах (всегда 4.7619 в данном датасете) |
| sales.csv | sales | gross income | float | Нет | Валовая прибыль — разница между Total и cogs |
| sales.csv | sales | Rating | float | Нет | Оценка обслуживания клиентом по шкале от 1 до 10 |

## 2.2. Анализ полноты и качества данных в датасете:

Проверка качества данных происходит в DAG (можно отследить в логах) **«check\_sales\_csv\_data\_extended\_to\_stg2**» (см. Приложение 1).



См. фразу о том, что «данные прошли проверку удачно»:

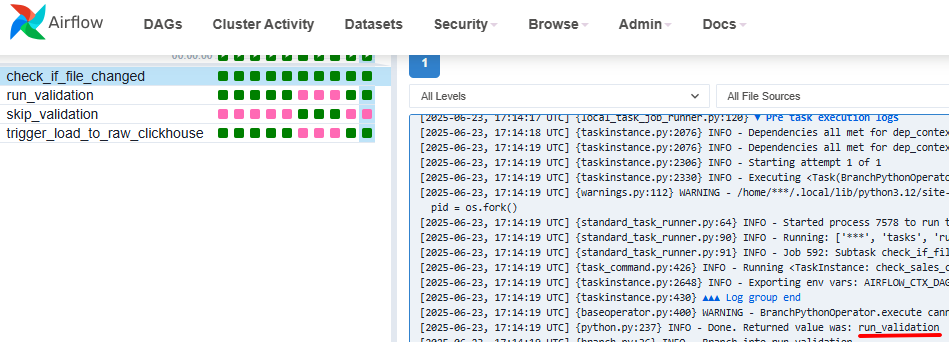
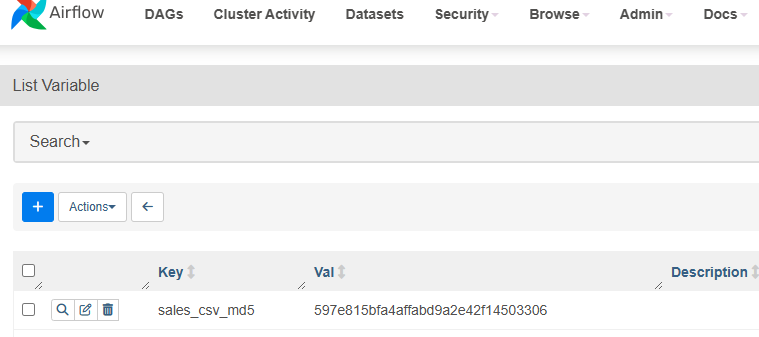


Проверка (task «**run\_validation**») выявила:

- пропусков в столбцах не было;  
- дубликатов не было;  
- отрицательных значений не выявлено;

-ошибок преобразования дат продаж – также не было.

**2.3. Проверка хэша cvs-файла на изменение и решение о его загрузке:**

Необходимость повторной загрузки cvs проверяется сравнением хэша файла со значением, сохраненным в перемен.«**sales\_cvs\_md5**», переход к валид. кач-ва данных:  

# 3. Проектирование нормализованного хранилища (NDS)

## 3.1. Выбор ключевых сущностей и атрибутов NDS

При проектировании NDS была проведена декомпозиция исходных данных на основные сущности: Продажи (Sales), Продукт (product), Филиал (store), Тип покупателя (customer type), Пол покупателя (gender), Типы платежей (payment)

|  |  |  |
| --- | --- | --- |
| **Таблица** | **Логика формирования данных из Primary Data Layer** | **SQL** |
| customer\_type | Берём данные из sales\_raw\_layer, выделяя уникальные customer\_type | core\_customer\_type.sql |
| gender | Берём данные из sales\_raw\_layer, выделяя уникальные gender | core\_gender.sql |
| store | Берём данные из sales\_raw\_layer, выделяя уникальные branch и city | core\_store.sql |
| product | Берём данные из sales\_raw\_layer, выделяя уникальные product\_line и unit\_price | core\_product.sql |
| payment | Берём данные из sales\_raw\_layer, выделяя уникальные payment | core\_payment.sql |
| sales | Формируем записи факта продаж, связывая внешние ключи из таблиц измерений | core\_sales.sql |

Для каждой новой таблицы были сгенерированы ID с помощью конструкции Clickhouse «**SELECT row\_number() OVER ()**». Это позволяет обеспечить целостность данных при переносе их в слой CORE.

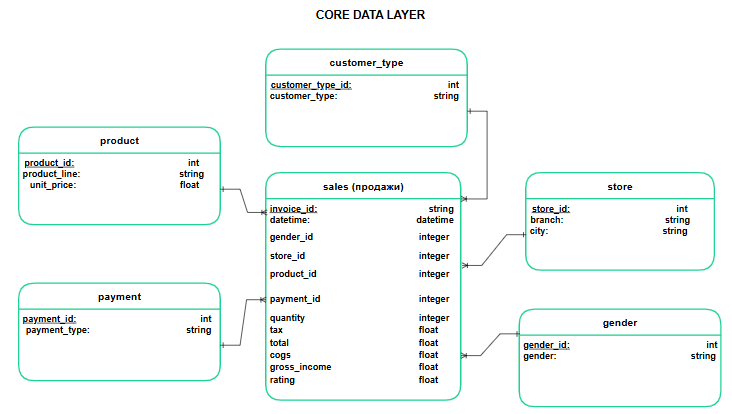
Для каждой сущности определены атрибуты:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Таблица** | **Атрибут** | **Тип атрибута** | **Первичный ключ** | **Внешний ключ** | **Атрибут из SQL (с листа «Таблицы»)** | **Комментарий к атрибуту** |
| customer\_type | customer\_type\_id | integer | Да | Нет | customer\_type\_id | Идентификатор типа клиента |
| customer\_type | customer\_type | string | Нет | Нет | customer\_type | Тип клиента (Member / Normal) |
| gender | gender\_id | integer | Да | Нет | gender\_id | Идентификатор пола |
| gender | gender | string | Нет | Нет | gender | Пол клиента |
| store | store\_id | integer | Да | Нет | store\_id | Идентификатор магазина |
| store | branch | string | Нет | Нет | branch | Филиал магазина (A, B, C) |
| store | city | string | Нет | Нет | city | Город магазина |
| product | product\_id | integer | Да | Нет | product\_id | Идентификатор продукта |
| product | product\_line | string | Нет | Нет | product\_line | Категория товара |
| product | unit\_price | float | Нет | Нет | unit\_price | Цена за единицу товара |
| payment | payment\_id | integer | Да | Нет | payment\_id | Идентификатор способа оплаты |
| payment | payment\_type | string | Нет | Нет | payment\_type | Способ оплаты (Cash / Credit card / Ewallet) |
| sales | invoice\_id | string | Да | Нет | invoice\_id | Номер чека |
| sales | customer\_type\_id | integer | Нет | Да | customer\_type\_id | Связь с таблицей customer\_type |
| sales | gender\_id | integer | Нет | Да | gender\_id | Связь с таблицей gender |
| sales | store\_id | integer | Нет | Да | store\_id | Связь с таблицей store |
| sales | product\_id | integer | Нет | Да | product\_id | Связь с таблицей product |
| sales | payment\_id | integer | Нет | Да | payment\_id | Связь с таблицей payment |
| sales | datetime | datetime | Нет | Нет | datetime | Дата и время покупки |
| sales | quantity | integer | Нет | Нет | quantity | Количество товаров |
| sales | tax | float | Нет | Нет | tax | Налог 5% |
| sales | total | float | Нет | Нет | total | Общая сумма чека |
| sales | cogs | float | Нет | Нет | cogs | Себестоимость |
| sales | gross\_income | float | Нет | Нет | gross\_income | Валовая прибыль |
| sales | rating | float | Нет | Нет | rating | Оценка клиента |

## 3.2. Логическая ER-диаграмма NDS

Для визуализации взаимосвязей сущностей использована ER-диаграмма, построенная в Draw.io. Основные связи:  
- Один Product может встречаться во многих записях Sales (1:N);  
- Каждая продажа связана с одним Store и одним Customer (N:1);  
- Оплата (Payment) связана с конкретной продажей через поле payment\_id (1:N).  
Диаграмма отражает первичные и внешние ключи, зоны ответственности и позволяет оценить структуру будущего хранилища.

Null в связях новых таблиц с таблицей sales при такой архитектуре быть не должно, т.к. внешние ключи это гарантируют.



# 4. Схема звезды (DDS) и витрины

В данном разделе описывается разработка схемы «звезда» для слоя datamart аналитических витрин на основе предыдущего этапа с нормализованным слоем NDS..

Для построения DDS были определены одна таблица фактов и шесть таблиц измерений.

## 4.1. Выбор ключевых сущностей и атрибутов DDS

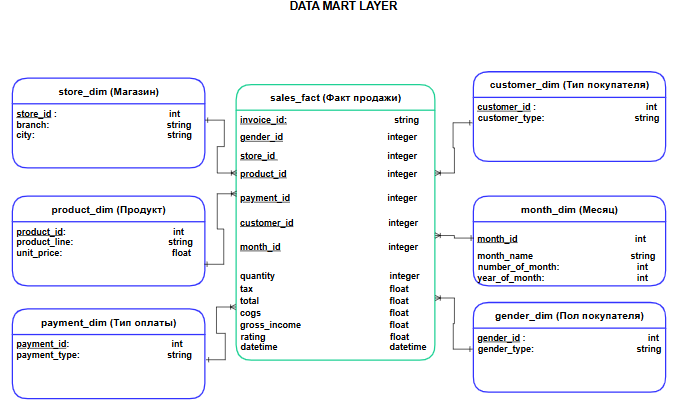
Каждая таблица измерений загружается перед таблицами фактов, чтобы обеспечить ссылочную целостность.

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица** | **Логика формирования данных из Core Data Layer** | **SQL** | **Комментарий к таблице** |
| sales\_fact | Берём данные из таблицы core\_layer.sales генерируем month\_id на основе столбца datetime | core\_to\_datamart\_sales\_fact.sql |  |
| customer\_dim | Берём из таблицы customer\_type слоя core\_layer | core\_to\_datamart\_customer.sql |  |
| gender\_dim | Берём из таблицы gender слоя core\_layer | core\_to\_datamart\_gender.sql |  |
| store\_dim | Берём из таблицы store слоя core\_layer | core\_to\_datamart\_store.sql |  |
| product\_dim | Берём из таблицы product слоя core\_layer | core\_to\_datamart\_product.sql |  |
| payment\_dim | Берём из таблицы payment слоя core\_layer | core\_to\_datamart\_payment.sql |  |
| month\_dim | Генерируем с помощью ClickHouse функций дат | generate\_month\_dim.sql |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Таблица** | **Атрибут** | **Тип атрибута** | **Первичный ключ** | **Внешний ключ** | **Атрибут из SQL (с листа «Таблицы»)** | **Комментарий к атрибуту** |
| sales\_fact | invoice\_id | string | Да | Нет | invoice\_id | Идентификатор чека продажи |
| sales\_fact | customer\_type\_id | integer | Нет | Да | customer\_type\_id | Внешний ключ на customer\_dim |
| sales\_fact | gender\_id | integer | Нет | Да | gender\_id | Внешний ключ на gender\_dim |
| sales\_fact | store\_id | integer | Нет | Да | store\_id | Внешний ключ на store\_dim |
| sales\_fact | product\_id | integer | Нет | Да | product\_id | Внешний ключ на product\_dim |
| sales\_fact | payment\_id | integer | Нет | Да | payment\_id | Внешний ключ на payment\_dim |
| sales\_fact | month\_id | integer | Нет | Да | month\_id | Внешний ключ на month\_dim |
| sales\_fact | datetime | datetime | Нет | Нет | datetime | Дата и время покупки |
| sales\_fact | quantity | integer | Нет | Нет | quantity | Количество товаров |
| sales\_fact | tax | float | Нет | Нет | tax | Налог |
| sales\_fact | total | float | Нет | Нет | total | Общая сумма продажи |
| sales\_fact | cogs | float | Нет | Нет | cogs | Себестоимость |
| sales\_fact | gross\_income | float | Нет | Нет | gross\_income | Валовая прибыль |
| sales\_fact | rating | float | Нет | Нет | rating | Оценка сервиса |
| month\_dim | month\_id | integer | Да | Нет | month\_id | YYYYMM идентификатор месяца |
| month\_dim | month\_name | string | Нет | Нет | month\_name | Название месяца |
| month\_dim | number\_of\_month | integer | Нет | Нет | number\_of\_month | Номер месяца |
| month\_dim | year\_of\_month | integer | Нет | Нет | year\_of\_month | Год |
| customer\_dim | customer\_type\_id | integer | Да | Нет | customer\_type\_id | Тип покупателя |
| customer\_dim | customer\_type | string | Нет | Нет | customer\_type | Название типа |
| gender\_dim | gender\_id | integer | Да | Нет | gender\_id | Пол |
| gender\_dim | gender | string | Нет | Нет | gender | Значение пола |
| store\_dim | store\_id | integer | Да | Нет | store\_id | Идентификатор магазина |
| store\_dim | branch | string | Нет | Нет | branch | Филиал |
| store\_dim | city | string | Нет | Нет | city | Город |
| product\_dim | product\_id | integer | Да | Нет | product\_id | Идентификатор продукта |
| product\_dim | product\_line | string | Нет | Нет | product\_line | Линейка продукта |
| product\_dim | unit\_price | float | Нет | Нет | unit\_price | Цена за единицу |
| payment\_dim | payment\_id | integer | Да | Нет | payment\_id | Идентификатор оплаты |
| payment\_dim | payment\_type | string | Нет | Нет | payment\_type | Тип оплаты |

## 4.2. Логическая ER-диаграмма DDS

Схема «звезда» позволяет быстрое выполнения OLAP-запросов и формирование отчетов, обеспечивая высокую скорость агрегации и простоту понимания структуры



# 5. Оркестрация ETL-процессов

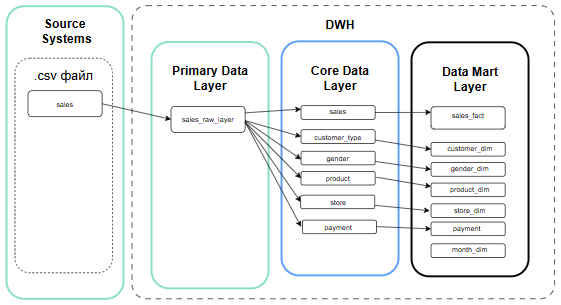
Для обеспечения надёжного выполнения ETL-процессов в дипломном проекте я использовал Apache Airflow — система управления рабочими процессами (workflow orchestration). Каждый этап загрузки и преобразования данных оформлен в виде DAG (Directed Acyclic Graph), где задачи последовательно или параллельно выполняются в соответствии с зависимостями. Использование Airflow позволило централизованно управлять расписанием, отслеживать статус джобов и автоматически повторять неудавшиеся задачи по заданным правилам retry.

Контейнеризация компонентов существенно ускорила мою реализацию задания и была осуществлена с помощью Docker-Desktop. Это позволило упростить деплой без сбоев и возможных конфликтов зависимостей компонентов моего решения.

Расписание выполнения ETL задаётся через параметры DAG: ежедневный запуск в 03:00 по локальному времени обеспечивает загрузку свежих данных за предыдущий день. Мониторинг процессов реализован с помощью встроенного Airflow UI по порту 8080. Такой подход обеспечивает прозрачность и возможность в реальном времени реагировать на сбои в ETL-пайплайне и корректировать логику ETL-процесса на этапе разработки и в дальнейшем жизненном цикле моего архитектурного решения.

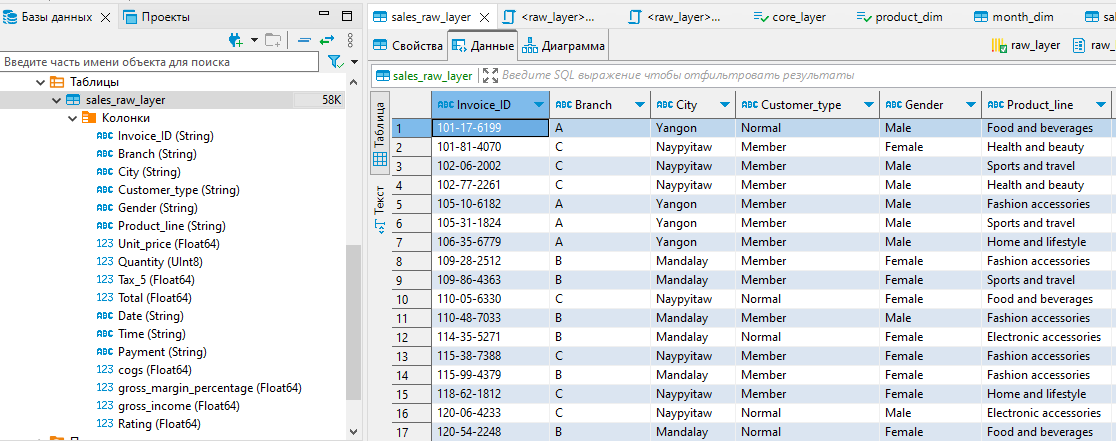
# 6. Процессы загрузки и трансформации данных

Для удобства управления и контроля загруженных данных используется структура слоёв: «raw\_layer» → «core\_layer» → «datamart\_layer», что позволяет изолировать влияние изменений и повторно использовать одни и те же скрипты при переработке или актуализации данных.



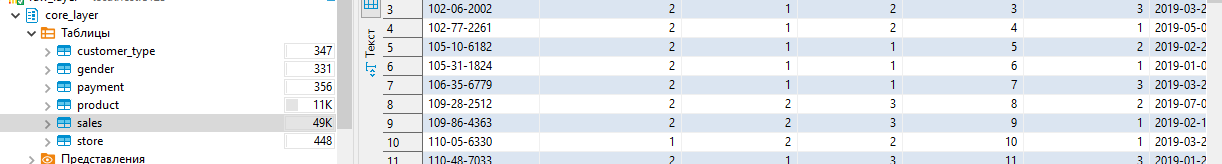
В данном разделе приведено описание ключевых этапов извлечения, трансформации и загрузки данных ETL-пайплайна, обеспечивающих корректность и готовность информации для аналитики. Процессы реализованы с помощью Python-скриптов и SQL-запросов, выполняемых в dag-ах внутри Apache Airflow.

Например, этап «Extract» выполняется в DAG: «**load\_to\_raw\_clickhouse**» (см. Приложение 1)

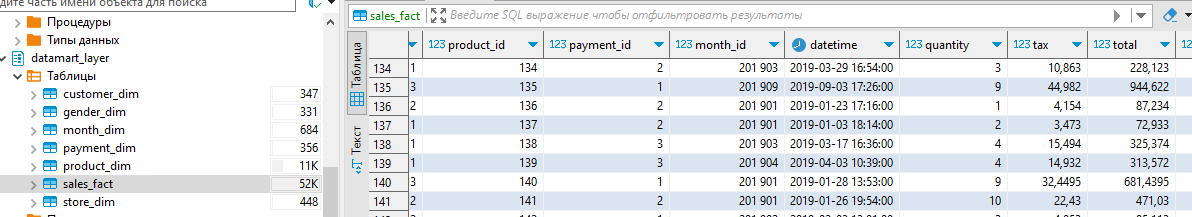
В нем данные из CSV-файлов загружаются в исходном виде (как были в cvs) в RAW-слой в таблицу «**raw\_layer.sales\_raw\_layer**» в ClickHouse. 

Следующий этап выполняется в DAG: «[**load\_nds\_from\_raw**](http://localhost:8080/dags/load_nds_from_raw/grid)» (см. Приложение 1)

На этапе происходит «Transform» - данные перемещаются в слой «**core\_layer**», где происходит нормализация: записи разбиваются на отдельные сущности (продукты, филиалы, клиенты), создаются внешние ключи (2-я нормальная форма) на таблицу «core\_layer.sales». Все преобразования описаны в SQL-запросах, запускаемых из Airflow DAG.

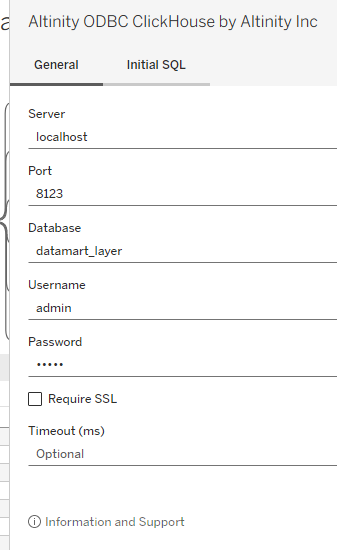
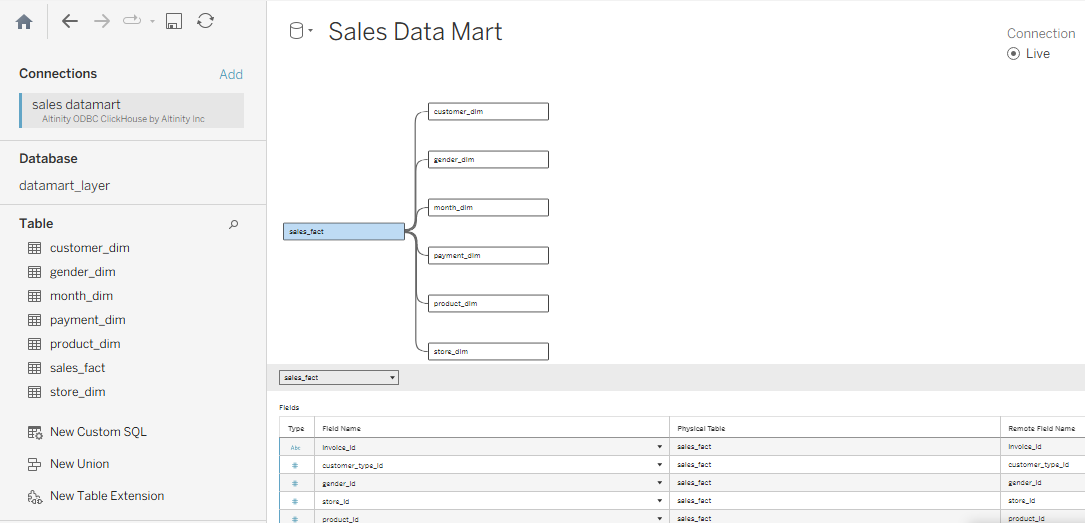


### Далее следует этап «Load» и DAG «load\_dds\_from\_core» (Приложение 1): нормализованные данные переносятся в слой datamart\_layer (в витрины DDS). Загрузка в таблицы «dim\_\*» и «fact\_\*» осуществляется пакетно через ClickHouse INSERT SELECT, что позволяет выполнять массовые обновления минимальными транзакциями.



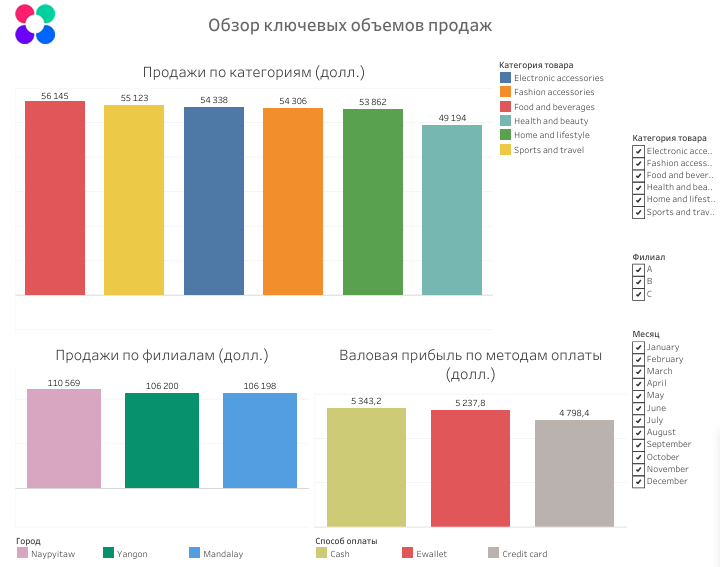
# 7. Построение дашбордов в Tableau

В данном разделе описывается процесс разработки и реализации интерактивных дашбордов в Tableau, которые служат основным инструментом визуализации ключевых бизнес-метрик и поддержки принятия решений. Разработка дашбордов строилась на основании требований к аналитике: обеспечение наглядности, гибкость фильтрации и удобство взаимодействия для конечного пользователя.

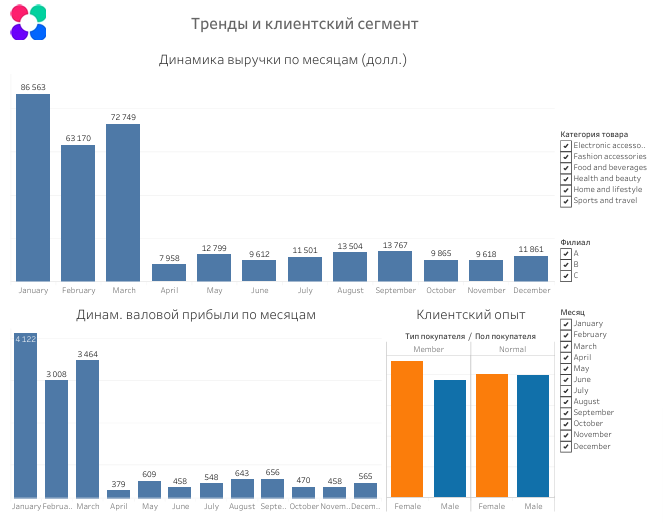
Подключение к Clickhouse осуществил с помощью ODBC-драйвера к Clickhouse из состава триал (платной) версии Tableau: 

Первым этапом была постановка задач и выбор основных показателей для отображения. Были выделены ключевых метрики: общая выручка и валовая прибыль. Для каждой метрики определены сценарии использования: мониторинг динамики, сравнение между филиалами и категориями товаров, а также оценка эффективности методов оплаты и сегментов клиентов.

Конструкция дашбордов состоит из двух экранов. Экран «Обзор ключевых объёмов продаж» содержит три столбчатых диаграммы, позволяющие быстро увидеть распределение продаж по категориям, филиалам и методам оплаты.



Экран «Тренды и клиентский сегмент» показывает временную динамику выручки и прибыли, а также гистограмму, демонстрирующую распределение клиентского опыта по типу покупателя и его пола. Оба экрана снабжены единой панелью фильтров слева, обеспечивающей консистентность и интуитивность интерфейса.



Для каждого листа настроены механизмы взаимодействия пользователя: выбор значений в фильтрах автоматически пересчитывает и обновляет все визуализации, наведение курсора на элементы графиков отображает подробные тултипы с числовыми и процентными данными.

Результаты работы представлены в виде Tableau-воркбука в формате .twb, включающего все настройки листов и дашбордов. Дашборды протестированы на корретность фильтрации и отзывчивость интерфейса при различных комбинациях параметров. Для обеспечения удобства распространения отчетности предусмотрен экспорт текущего состояния дашбордов в PDF или Word, что позволяет оперативно делиться аналитикой с коллегами и руководством.

Аналитика на основе построенных дашбордов приведена в Приложении 2: «Пояснительная записка к дашбордам Sales Data Mart (Tableau)»

# 8. Выводы по дипломному проекту

В рамках дипломного проекта я выстроил сквозной ETL-конвейер от предварительной обработки исходного CSV-файла до формирования витринных таблиц в ClickHouse. Оркестрацию сделал через Apache Airflow, что позволило автоматизировать последовательность задач, обеспечив надёжность и прозрачность выполнения каждого этапа.

Весомым, на мой взгляд, преимуществом выбранного архитектурного решения стала контейнеризация с помощью Docker, которая обеспечила мне лёгкость развертывания и повторяемости результатов.

Процесс извлечения данных (Extract) на базе Python-скриптов и pandas выявил необходимость строгой валидации входных файлов. В ходе работы над проектом мной использовались механизмы логирования и обработки ошибок, позволяющие ускорить процесс ревизии python-кода dag-ов Airflow .

Этап трансформации (Transform) продемонстрировал эффективность разделения логики на SQL-джобы в ClickHouse и Python-таски в Airflow. Не смотря на кажущееся преимущество параллельного выполнение задач нормализации и генерации ключей - я выбрал последовательное выполнение задач исходя из практики загрузок данных и необходимости в стабильной загрузке данных. Т.е. выполнение вначале предварительных тасок, а уже потом итоговой загрузки данных. Это помогло мне осуществлять контроль загрузок и получить требуемый результат. В высоконагруженных решениях возможно параллельное выполнение было бы более предпочтительным. Я пришел к выводу, что нужно индивидуально смотреть в разрезе архитектуры и инфраструктуры решения под поставленную задачу – индивидуально. В случае дипломного проекта - моя архитектура незначительно увеличила общее время конвейера, однако чёткое разделение слоёв raw\_layer, core\_layer и datamart\_layer обеспечило модульность и удобство отладки.

Загрузка (Load) в конечные таблицы Fact и Dim через INSERT SELECT операций в ClickHouse показала высокую пропускную способность движка MergeTree. Оптимизация ORDER BY и использование пакетных вставок позволили достичь низкой латентности при обновлении витрин.

Входной контроль качества был данных внедрён мной на уровне Airflow: выполняются автоматические pandas-тесты на отсутствие дубликатов, корректность типов и соответствие бизнес-правилам.

В результате была создана надёжная инфраструктура ETL, обладающая следующими характеристиками:  
- Полная автоматизация процессов с гибкой настройкой расписаний в Airflow;  
- Воспроизводимость окружения за счёт Docker;  
- Масштабируемость и модульность благодаря архитектуре слоёв хранилища данных;  
- Прозрачный мониторинг и отладка через Airflow UI и логи;  
- Высокая производительность загрузки и агрегации в ClickHouse.

В качестве дальнейших шагов по развитию решения я размещу файлы проекта в системе контроля версий (Git) для DAG-файлов.

Из того, что можно было бы сделать – добавить в хранилище данных - слой метаданные и отслеживать количества, время выполнения задач и качество каждого прохождения данных.

**Дипломный проект позволил систематизировать те знания, которые я получил на курсе «Инженер данных» и получить бесценный опыт и веру в то, что в этой перспективной профессии – можно развиваться и успешно решать интересные задачи!**

**Приложения:**

1. DAG-и  
2. Пояснительная записка к дашбордам Sales Data Mart (Tableau)  
3. ER-диаграммы и таблицы хранилища данных

**DAG-и**

1. **Check\_sales\_csv\_data\_extended.py**

*from airflow import DAG*

*from airflow.operators.python import PythonOperator, BranchPythonOperator*

*from airflow.operators.trigger\_dagrun import TriggerDagRunOperator*

*from airflow.models import Variable*

*from datetime import datetime*

*import hashlib*

*import pandas as pd*

*FILE\_PATH = '/opt/airflow/dags/files/sales.csv'*

*HASH\_VAR\_KEY = 'sales\_csv\_md5'*

*STAGE\_DONE\_VAR = 'stage\_num\_1\_done'*

*STAGE2\_DONE\_VAR ='stage\_num\_2\_done'*

*STAGE3\_DONE\_VAR ='stage\_num\_3\_done'*

*STAGE4\_DONE\_VAR ='stage\_num\_4\_done'*

*Variable.set(STAGE\_DONE\_VAR, 0)*

*Variable.set(STAGE2\_DONE\_VAR, 0)*

*Variable.set(STAGE3\_DONE\_VAR, 0)*

*Variable.set(STAGE4\_DONE\_VAR, 0)*

*def calculate\_md5(file\_path):*

*with open(file\_path, 'rb') as f:*

*return hashlib.md5(f.read()).hexdigest()*

*def check\_if\_file\_changed(\*\*kwargs):*

*current\_hash = calculate\_md5(FILE\_PATH)*

*stored\_hash = Variable.get(HASH\_VAR\_KEY, default\_var=None)*

*if stored\_hash == current\_hash:*

*print("Файл не изменился — загрузка не требуется.")*

*return 'skip\_validation'*

*else:*

*Variable.set(HASH\_VAR\_KEY, current\_hash)*

*return 'run\_validation'*

*def validate\_data\_and\_proceed(\*\*kwargs):*

*df = pd.read\_csv(FILE\_PATH)*

*errors = []*

*if df.isnull().sum().sum() > 0:*

*errors.append("Присутствуют пропущенные значения")*

*if df.duplicated().sum() > 0:*

*errors.append("Присутствуют дубликаты строк")*

*if (df.select\_dtypes(include='number') < 0).any().any():*

*errors.append("Есть отрицательные значения")*

*try:*

*pd.to\_datetime(df['Date'])*

*except Exception:*

*errors.append("Ошибка преобразования даты")*

*if errors:*

*Variable.set(STAGE\_DONE\_VAR, 0)*

*raise ValueError("Ошибки в данных:\n" + "\n".join(errors))*

*else:*

*print("Данные прошли проверку успешно.")*

*Variable.set(STAGE\_DONE\_VAR, 1)*

*def skip\_validation():*

*print("Пропуск валидации, файл не изменился.")*

*with DAG(*

*dag\_id='check\_sales\_csv\_data\_extended\_to\_stg2',*

*schedule\_interval='@daily',*

*start\_date=datetime(2019, 1, 1),*

*catchup=False,*

*tags=['validation', 'sales']*

*) as dag:*

*check\_file = BranchPythonOperator(*

*task\_id='check\_if\_file\_changed',*

*python\_callable=check\_if\_file\_changed,*

*provide\_context=True*

*)*

*validate = PythonOperator(*

*task\_id='run\_validation',*

*python\_callable=validate\_data\_and\_proceed*

*)*

*skip = PythonOperator(*

*task\_id='skip\_validation',*

*python\_callable=skip\_validation*

*)*

*trigger\_load\_raw = TriggerDagRunOperator(*

*task\_id='trigger\_load\_to\_raw\_clickhouse',*

*trigger\_dag\_id='load\_to\_raw\_clickhouse',*

*wait\_for\_completion=False,*

*reset\_dag\_run=True*

*)*

*check\_file >> [validate, skip]*

*validate >> trigger\_load\_raw*

1. **load\_to\_raw\_clickhouse.py**

*from airflow import DAG*

*from airflow.providers.http.operators.http import HttpOperator*

*from airflow.operators.python import PythonOperator*

*from airflow.operators.trigger\_dagrun import TriggerDagRunOperator*

*from airflow.models import Variable*

*from airflow.utils.dates import days\_ago*

*import pandas as pd*

*STAGE\_VAR = 'stage\_num\_2\_done'*

*FILE\_PATH = '/opt/airflow/dags/files/sales.csv'*

*TABLE\_NAME = 'raw\_layer.sales\_raw\_layer'*

*default\_args = {*

*'owner': 'airflow',*

*'retries': 1,*

*}*

*with DAG(*

*dag\_id='load\_to\_raw\_clickhouse',*

*default\_args=default\_args,*

*schedule\_interval=None,*

*start\_date=days\_ago(1),*

*catchup=False,*

*tags=['clickhouse', 'raw']*

*) as dag:*

*drop\_table = HttpOperator(*

*task\_id='drop\_sales\_raw\_table',*

*http\_conn\_id='clickhouse\_http\_default',*

*endpoint='',*

*method='POST',*

*data=f"DROP TABLE IF EXISTS {TABLE\_NAME}",*

*headers={"Content-Type": "application/sql"},*

*log\_response=True,*

*)*

*create\_table = HttpOperator(*

*task\_id='create\_sales\_raw\_table',*

*http\_conn\_id='clickhouse\_http\_default',*

*endpoint='',*

*method='POST',*

*data=f"""*

*CREATE TABLE {TABLE\_NAME} (*

*Invoice\_ID String,*

*Branch String,*

*City String,*

*Customer\_type String,*

*Gender String,*

*Product\_line String,*

*Unit\_price Float64,*

*Quantity UInt8,*

*Tax\_5 Float64,*

*Total Float64,*

*Date String,*

*Time String,*

*Payment String,*

*cogs Float64,*

*gross\_margin\_percentage Float64,*

*gross\_income Float64,*

*Rating Float64*

*) ENGINE = MergeTree()*

*ORDER BY Invoice\_ID*

*""",*

*headers={"Content-Type": "application/sql"},*

*log\_response=True,*

*)*

*def insert\_sales\_data():*

*df = pd.read\_csv(FILE\_PATH)*

*df.columns = [c.replace(" ", "\_").replace("%", "") for c in df.columns]*

*values = []*

*for \_, row in df.iterrows():*

*val = "(" + ", ".join([*

*f"'{str(row['Invoice\_ID'])}'",*

*f"'{row['Branch']}'",*

*f"'{row['City']}'",*

*f"'{row['Customer\_type']}'",*

*f"'{row['Gender']}'",*

*f"'{row['Product\_line']}'",*

*f"{row['Unit\_price']}",*

*f"{int(row['Quantity'])}",*

*f"{row['Tax\_5']}",*

*f"{row['Total']}",*

*f"'{row['Date']}'",*

*f"'{row['Time']}'",*

*f"'{row['Payment']}'",*

*f"{row['cogs']}",*

*f"{row['gross\_margin\_percentage']}",*

*f"{row['gross\_income']}",*

*f"{row['Rating']}"*

*]) + ")"*

*values.append(val)*

*query = f"INSERT INTO {TABLE\_NAME} VALUES " + ",\n".join(values)*

*from airflow.providers.http.hooks.http import HttpHook*

*hook = HttpHook(method='POST', http\_conn\_id='clickhouse\_http\_default')*

*hook.run(endpoint="", data=query, headers={"Content-Type": "application/sql"})*

*Variable.set(STAGE\_VAR, 1)*

*print("Данные успешно загружены в ClickHouse.")*

*insert\_task = PythonOperator(*

*task\_id='insert\_sales\_data',*

*python\_callable=insert\_sales\_data*

*)*

*trigger\_nds\_dag = TriggerDagRunOperator(*

*task\_id='trigger\_load\_nds\_from\_raw',*

*trigger\_dag\_id='load\_nds\_from\_raw',*

*wait\_for\_completion=False,*

*reset\_dag\_run=True*

*)*

*drop\_table >> create\_table >> insert\_task >> trigger\_nds\_dag*

1. **load\_nds\_from\_raw.py**

*from airflow import DAG*

*from airflow.providers.http.operators.http import SimpleHttpOperator*

*from airflow.operators.python import PythonOperator*

*from airflow.operators.trigger\_dagrun import TriggerDagRunOperator*

*from airflow.models import Variable*

*from airflow.utils.dates import days\_ago*

*STAGE\_VAR = 'stage\_core\_loaded'*

*default\_args = {*

*'owner': 'airflow',*

*'retries': 1,*

*}*

*with DAG(*

*dag\_id='load\_nds\_from\_raw',*

*default\_args=default\_args,*

*schedule\_interval=None,*

*start\_date=days\_ago(1),*

*catchup=False,*

*tags=['clickhouse', 'core', '3nf']*

*) as dag:*

*def build\_sql\_task(task\_id, sql):*

*return SimpleHttpOperator(*

*task\_id=task\_id,*

*http\_conn\_id='clickhouse\_http\_default',*

*method='POST',*

*data=sql,*

*headers={"Content-Type": "application/sql"},*

*log\_response=True,*

*)*

*task\_sequence = []*

*sql\_task\_defs = [*

*("core\_customer\_type\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.customer\_type (*

*customer\_type\_id UInt32,*

*customer\_type String*

*) ENGINE = MergeTree() ORDER BY customer\_type\_id;'''),*

*("core\_customer\_type\_truncate", "TRUNCATE TABLE core\_layer.customer\_type;"),*

*("core\_customer\_type", '''INSERT INTO core\_layer.customer\_type*

*SELECT row\_number() OVER () AS customer\_type\_id, "Customer\_type" AS customer\_type*

*FROM (SELECT DISTINCT "Customer\_type" FROM raw\_layer.sales\_raw\_layer);'''),*

*("core\_gender\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.gender (*

*gender\_id UInt32,*

*gender String*

*) ENGINE = MergeTree() ORDER BY gender\_id;'''),*

*("core\_gender\_truncate", "TRUNCATE TABLE core\_layer.gender;"),*

*("core\_gender", '''INSERT INTO core\_layer.gender*

*SELECT row\_number() OVER () AS gender\_id, "Gender" AS gender*

*FROM (SELECT DISTINCT "Gender" FROM raw\_layer.sales\_raw\_layer);'''),*

*("core\_store\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.store (*

*store\_id UInt32,*

*branch String,*

*city String*

*) ENGINE = MergeTree() ORDER BY store\_id;'''),*

*("core\_store\_truncate", "TRUNCATE TABLE core\_layer.store;"),*

*("core\_store", '''INSERT INTO core\_layer.store*

*SELECT row\_number() OVER () AS store\_id, "Branch", "City"*

*FROM (SELECT DISTINCT "Branch", "City" FROM raw\_layer.sales\_raw\_layer);'''),*

*("core\_product\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.product (*

*product\_id UInt32,*

*product\_line String,*

*unit\_price Float64*

*) ENGINE = MergeTree() ORDER BY product\_id;'''),*

*("core\_product\_truncate", "TRUNCATE TABLE core\_layer.product;"),*

*("core\_product", '''INSERT INTO core\_layer.product*

*SELECT row\_number() OVER () AS product\_id, "Product\_line", "Unit\_price"*

*FROM (SELECT DISTINCT "Product\_line", "Unit\_price" FROM raw\_layer.sales\_raw\_layer);'''),*

*("core\_payment\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.payment (*

*payment\_id UInt32,*

*payment\_type String*

*) ENGINE = MergeTree() ORDER BY payment\_id;'''),*

*("core\_payment\_truncate", "TRUNCATE TABLE core\_layer.payment;"),*

*("core\_payment", '''INSERT INTO core\_layer.payment*

*SELECT row\_number() OVER () AS payment\_id, "Payment"*

*FROM (SELECT DISTINCT "Payment" FROM raw\_layer.sales\_raw\_layer);'''),*

*("core\_sales\_create", '''CREATE TABLE IF NOT EXISTS core\_layer.sales (*

*invoice\_id String,*

*customer\_type\_id UInt32,*

*gender\_id UInt32,*

*store\_id UInt32,*

*product\_id UInt32,*

*payment\_id UInt32,*

*datetime DateTime,*

*quantity UInt8,*

*tax Float64,*

*total Float64,*

*cogs Float64,*

*gross\_income Float64,*

*rating Float64*

*) ENGINE = MergeTree() ORDER BY invoice\_id;'''),*

*("core\_sales\_truncate", "TRUNCATE TABLE core\_layer.sales;"),*

*("core\_sales", '''INSERT INTO core\_layer.sales*

*SELECT*

*s."Invoice\_ID",*

*ct.customer\_type\_id,*

*g.gender\_id,*

*st.store\_id,*

*p.product\_id,*

*pay.payment\_id,*

*parseDateTimeBestEffort(concat(s."Date", ' ', s."Time")) AS datetime,*

*s."Quantity",*

*s."Tax\_5" AS tax,*

*s."Total",*

*s."cogs",*

*s."gross\_income",*

*s."Rating"*

*FROM raw\_layer.sales\_raw\_layer s*

*JOIN core\_layer.customer\_type ct ON s."Customer\_type" = ct.customer\_type*

*JOIN core\_layer.gender g ON s."Gender" = g.gender*

*JOIN core\_layer.store st ON s."Branch" = st.branch AND s."City" = st.city*

*JOIN core\_layer.product p ON s."Product\_line" = p.product\_line AND s."Unit\_price" = p.unit\_price*

*JOIN core\_layer.payment pay ON s."Payment" = pay.payment\_type;'''),*

*]*

*last\_task = None*

*for task\_id, sql in sql\_task\_defs:*

*task = build\_sql\_task(task\_id, sql)*

*if last\_task:*

*last\_task >> task*

*last\_task = task*

*set\_stage\_done = PythonOperator(*

*task\_id='set\_stage\_core\_loaded',*

*python\_callable=lambda: Variable.set(STAGE\_VAR, 1),*

*)*

*trigger\_next = TriggerDagRunOperator(*

*task\_id='trigger\_load\_dds\_from\_core',*

*trigger\_dag\_id='load\_dds\_from\_core',*

*wait\_for\_completion=False,*

*reset\_dag\_run=True,*

*)*

*last\_task >> set\_stage\_done >> trigger\_next*

1. **load\_dds\_from\_core.py**

*from airflow import DAG*

*from airflow.providers.http.operators.http import SimpleHttpOperator*

*from airflow.operators.python import PythonOperator*

*from airflow.utils.dates import days\_ago*

*from airflow.models import Variable*

*STAGE\_VAR = 'stage\_dds\_loaded'*

*default\_args = {*

*'owner': 'airflow',*

*'retries': 1,*

*}*

*def build\_sql\_task(task\_id, sql):*

*return SimpleHttpOperator(*

*task\_id=task\_id,*

*http\_conn\_id='clickhouse\_http\_default',*

*method='POST',*

*data=sql,*

*headers={"Content-Type": "application/sql"},*

*log\_response=True,*

*)*

*with DAG(*

*dag\_id='load\_dds\_from\_core',*

*default\_args=default\_args,*

*schedule\_interval=None,*

*start\_date=days\_ago(1),*

*catchup=False,*

*tags=['clickhouse', 'datamart', 'star\_schema']*

*) as dag:*

*# customer\_dim*

*customer\_create = build\_sql\_task("create\_customer\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.customer\_dim (*

*customer\_type\_id UInt32,*

*customer\_type String*

*) ENGINE = MergeTree() ORDER BY customer\_type\_id;*

*''')*

*customer\_truncate = build\_sql\_task("truncate\_customer\_dim", "TRUNCATE TABLE datamart\_layer.customer\_dim;")*

*customer\_insert = build\_sql\_task("insert\_customer\_dim", '''*

*INSERT INTO datamart\_layer.customer\_dim*

*SELECT \* FROM core\_layer.customer\_type;*

*''')*

*# gender\_dim*

*gender\_create = build\_sql\_task("create\_gender\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.gender\_dim (*

*gender\_id UInt32,*

*gender String*

*) ENGINE = MergeTree() ORDER BY gender\_id;*

*''')*

*gender\_truncate = build\_sql\_task("truncate\_gender\_dim", "TRUNCATE TABLE datamart\_layer.gender\_dim;")*

*gender\_insert = build\_sql\_task("insert\_gender\_dim", '''*

*INSERT INTO datamart\_layer.gender\_dim*

*SELECT \* FROM core\_layer.gender;*

*''')*

*# store\_dim*

*store\_create = build\_sql\_task("create\_store\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.store\_dim (*

*store\_id UInt32,*

*branch String,*

*city String*

*) ENGINE = MergeTree() ORDER BY store\_id;*

*''')*

*store\_truncate = build\_sql\_task("truncate\_store\_dim", "TRUNCATE TABLE datamart\_layer.store\_dim;")*

*store\_insert = build\_sql\_task("insert\_store\_dim", '''*

*INSERT INTO datamart\_layer.store\_dim*

*SELECT \* FROM core\_layer.store;*

*''')*

*# product\_dim*

*product\_create = build\_sql\_task("create\_product\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.product\_dim (*

*product\_id UInt32,*

*product\_line String,*

*unit\_price Float64*

*) ENGINE = MergeTree() ORDER BY product\_id;*

*''')*

*product\_truncate = build\_sql\_task("truncate\_product\_dim", "TRUNCATE TABLE datamart\_layer.product\_dim;")*

*product\_insert = build\_sql\_task("insert\_product\_dim", '''*

*INSERT INTO datamart\_layer.product\_dim*

*SELECT \* FROM core\_layer.product;*

*''')*

*# payment\_dim*

*payment\_create = build\_sql\_task("create\_payment\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.payment\_dim (*

*payment\_id UInt32,*

*payment\_type String*

*) ENGINE = MergeTree() ORDER BY payment\_id;*

*''')*

*payment\_truncate = build\_sql\_task("truncate\_payment\_dim", "TRUNCATE TABLE datamart\_layer.payment\_dim;")*

*payment\_insert = build\_sql\_task("insert\_payment\_dim", '''*

*INSERT INTO datamart\_layer.payment\_dim*

*SELECT \* FROM core\_layer.payment;*

*''')*

*# month\_dim*

*month\_create = build\_sql\_task("create\_month\_dim", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.month\_dim (*

*month\_id UInt32,*

*month\_name String,*

*number\_of\_month UInt8,*

*year\_of\_month UInt32*

*) ENGINE = MergeTree() ORDER BY month\_id;*

*''')*

*month\_truncate = build\_sql\_task("truncate\_month\_dim", "TRUNCATE TABLE datamart\_layer.month\_dim;")*

*month\_insert = build\_sql\_task("insert\_month\_dim", '''*

*INSERT INTO datamart\_layer.month\_dim*

*SELECT DISTINCT*

*toYYYYMM(datetime) AS month\_id,*

*dateName('month', datetime) AS month\_name,*

*toMonth(datetime) AS number\_of\_month,*

*toYear(datetime) AS year\_of\_month*

*from (*

*select*

*DateAdd(day, n.number, toDate('2019-01-01')) as datetime*

*from*

*numbers(0, 365 \* 1) as n*

*);*

*''')*

*# sales\_fact*

*sales\_create = build\_sql\_task("create\_sales\_fact", '''*

*CREATE TABLE IF NOT EXISTS datamart\_layer.sales\_fact (*

*invoice\_id String,*

*customer\_type\_id UInt32,*

*gender\_id UInt32,*

*store\_id UInt32,*

*product\_id UInt32,*

*payment\_id UInt32,*

*month\_id UInt32,*

*datetime DateTime,*

*quantity UInt8,*

*tax Float64,*

*total Float64,*

*cogs Float64,*

*gross\_income Float64,*

*rating Float64*

*) ENGINE = MergeTree() ORDER BY invoice\_id;*

*''')*

*sales\_truncate = build\_sql\_task("truncate\_sales\_fact", "TRUNCATE TABLE datamart\_layer.sales\_fact;")*

*sales\_insert = build\_sql\_task("insert\_sales\_fact", '''*

*INSERT INTO datamart\_layer.sales\_fact*

*SELECT*

*s.invoice\_id,*

*s.customer\_type\_id,*

*s.gender\_id,*

*s.store\_id,*

*s.product\_id,*

*s.payment\_id,*

*toYYYYMM(s.datetime) AS month\_id,*

*s.datetime,*

*s.quantity,*

*s.tax,*

*s.total,*

*s.cogs,*

*s.gross\_income,*

*s.rating*

*FROM core\_layer.sales s;*

*''')*

*finalize = PythonOperator(*

*task\_id='set\_stage\_dds\_loaded',*

*python\_callable=lambda: Variable.set(STAGE\_VAR, 1),*

*)*

*(*

*customer\_create >> customer\_truncate >> customer\_insert >>*

*gender\_create >> gender\_truncate >> gender\_insert >>*

*store\_create >> store\_truncate >> store\_insert >>*

*product\_create >> product\_truncate >> product\_insert >>*

*payment\_create >> payment\_truncate >> payment\_insert >>*

*month\_create >> month\_truncate >> month\_insert >>*

*sales\_create >> sales\_truncate >> sales\_insert >>*

*finalize*

*)*

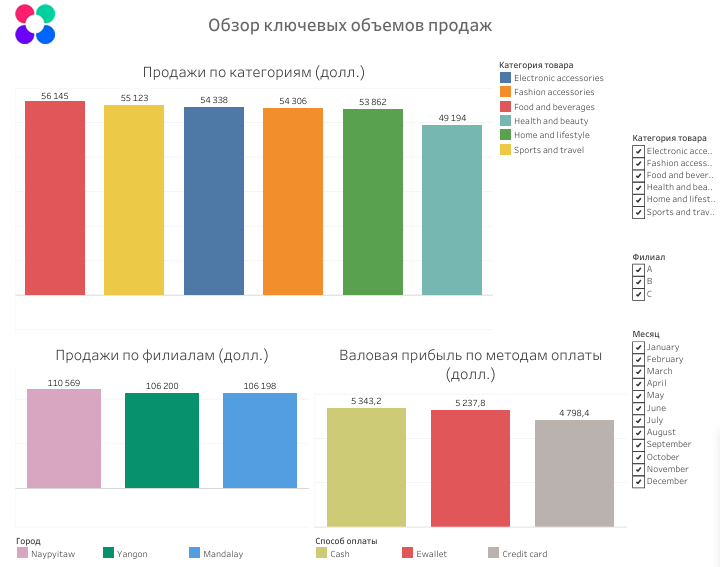
**Пояснительная записка к дашбордам Sales Datamart (Tableau)**

# 1. Введение

В этом отчете представлены два экрана дашборда Tableau, обеспечивающие обзор ключевых показателей продаж и детальный анализ трендов и сегментов.

Даны примеры аналитики на основе текущих данных из датасета.

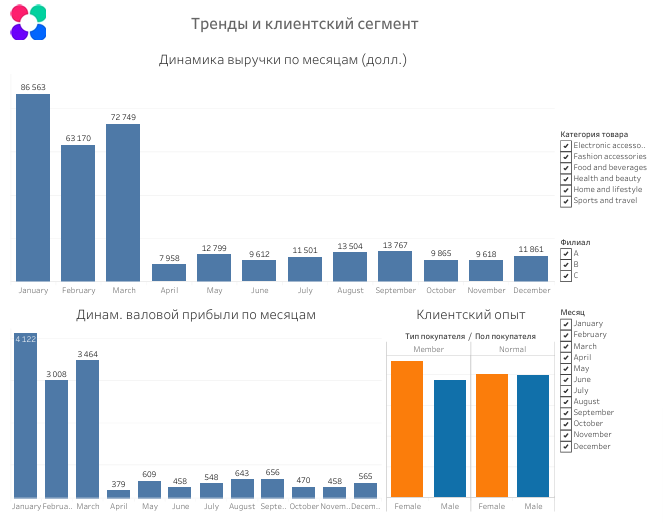
# 2. Дашборд 1: «Обзор ключевых объёмов продаж»



Сценарий работы: пользователь выбирает период, филиал, категорию товара в правой панели фильтров, после чего видит обновленные ключевые метрики диаграмм. Наведение на столбец отображает тултип с деталями, клик по фильтру включает или скрывает категории.

Аналитика: продажи по категориям сбалансированы — Food and beverages лидирует с $56 145, а Health and beauty отстает на $6 951. Филиалы Mandalay ($110 569), Naypyitaw ($106 200) и Yangon ($106 198) практически равны, Cash обеспечивает большую часть валовой прибыли, а Credit card — меньшую (есть необходимость провести анализ, почему данный способ оплаты отстает).

# 3. Дашборд 2: «Тренды и клиентский сегмент»



Сценарий работы: пользователь выбирает категорию товара и филиал, период в месяцах (только для клиентского опыта, т.к. остальные диаграммы уже отражаются в разрезе по месяцам) в правой панели фильтров, после чего получает две диаграммы динамики выручки и прибыли по месяцам и гистограмму клиентского опыта. Наведение на точку или сегмент показывает всплывающие подсказки с подробностями.

Аналитика: выручка и валовая прибыль достигли пика в январе и упали в апреле-июне ниже $500 в прибыли и $8 000 в выручке, затем восстановились в июле-сентябре. Участники программы лояльности (Member) совершают больше покупок, чем обычные покупатели. При этом женщины лидируют по числу покупок и среднему чеку(выручке) по сравнению с мужчинами.

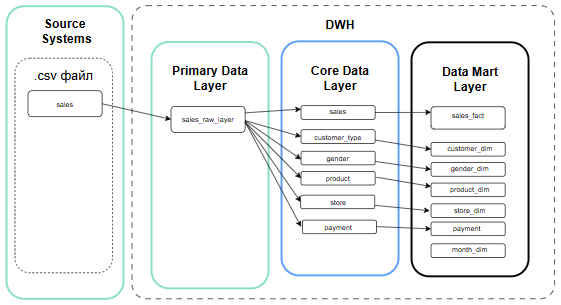
# 4. Выводы и рекомендации

- Использовать дашборд «Обзор ключевых объёмов продаж» для оперативного мониторинга объёмов и сравнения филиалов, категорий и методов оплаты.  
- Применять дашборд «Тренды и клиентский сегмент» для анализа сезонных трендов и сегментации клиентской базы с целью разработки акций, направленный на определенную аудиторию покупателей.  
- Провести дополнительный анализ причин общего апрельского спада, а также кардинального спада в феврале у мужской части клиентов и усилить работу с сегментом Health and beauty и клиентами не являющимися участниками программы лояльности.

**ER-диаграммы и таблицы хранилища данных**

1. PRIMARY DATA - слой
2. CORE DATA - слой
3. DATA MART - слой

***Схема хранилища данных***



1. PRIMARY DATA – слой (системы-источники)

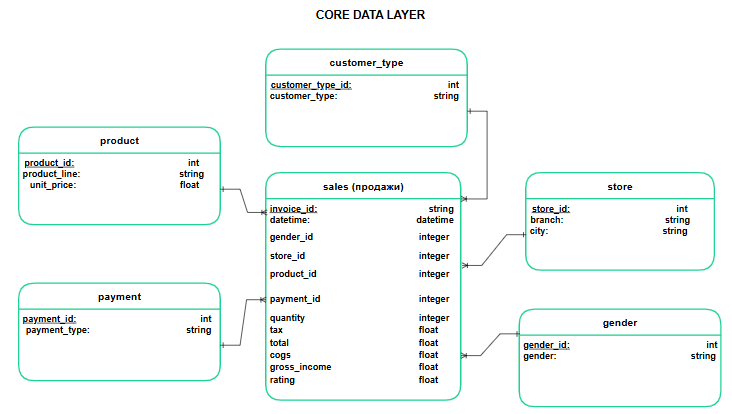
|  |  |  |  |
| --- | --- | --- | --- |
| **Система-источник** | **Тип источника** | **URL** | **Комментарий** |
| sales.csv | файл .csv |  | Данные о продажах |

1. PRIMARY DATA – слой (таблицы)

|  |  |  |  |
| --- | --- | --- | --- |
| **Система-источник** | **Таблица** | **Требования к забору данных из таблицы** | **Комментарий к таблице** |
| sales.cvs | sales | Загрузка должна быть каждый день |  |

1. PRIMARY DATA – слой (атрибуты)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Система-источник** | **Таблица** | **Атрибут** | **Тип атрибута** | **Является первичным ключом** | **Комментарий к атрибуту** |
| sales.csv | sales | Invoice ID | string | Да | Уникальный идентификатор чека (продажи) |
| sales.csv | sales | Branch | string | Нет | Обозначение филиала: A, B, C |
| sales.csv | sales | City | string | Нет | Город, где находится филиал (напр. Yangon, Mandalay) |
| sales.csv | sales | Customer type | string | Нет | Тип клиента: Member (постоянный) или Normal (разовый) |
| sales.csv | sales | Gender | string | Нет | Пол клиента: Male или Female |
| sales.csv | sales | Product line | string | Нет | Категория товара (например: Health and beauty, Electronic accessories) |
| sales.csv | sales | Unit price | float | Нет | Цена за единицу товара |
| sales.csv | sales | Quantity | integer | Нет | Количество приобретённого товара |
| sales.csv | sales | Tax 5% | float | Нет | Налог 5% от суммы без учёта налога |
| sales.csv | sales | Total | float | Нет | Общая сумма покупки (включая налог) |
| sales.csv | sales | Date | date | Нет | Дата покупки (в формате mm/dd/yyyy в источнике) |
| sales.csv | sales | Time | time (string) | Нет | Время покупки |
| sales.csv | sales | Payment | string | Нет | Способ оплаты: Cash, Credit card, Ewallet |
| sales.csv | sales | cogs | float | Нет | Себестоимость проданных товаров (Cost Of Goods Sold) |
| sales.csv | sales | gross margin percentage | float | Нет | Валовая маржа в процентах (всегда 4.7619 в данном датасете) |
| sales.csv | sales | gross income | float | Нет | Валовая прибыль — разница между Total и cogs |
| sales.csv | sales | Rating | float | Нет | Оценка обслуживания клиентом по шкале от 1 до 10 |

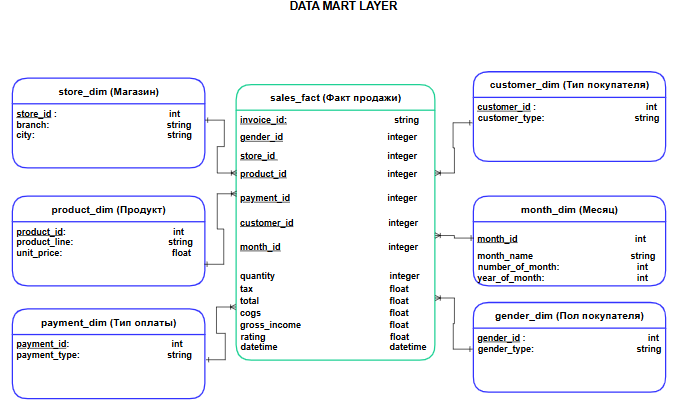


1. CORE DATA – слой (таблицы)

|  |  |  |
| --- | --- | --- |
| **Таблица** | **Логика формирования данных из Primary Data Layer** | **SQL** |
| customer\_type | Берём данные из sales\_raw\_layer, выделяя уникальные customer\_type | core\_customer\_type.sql |
| gender | Берём данные из sales\_raw\_layer, выделяя уникальные gender | core\_gender.sql |
| store | Берём данные из sales\_raw\_layer, выделяя уникальные branch и city | core\_store.sql |
| product | Берём данные из sales\_raw\_layer, выделяя уникальные product\_line и unit\_price | core\_product.sql |
| payment | Берём данные из sales\_raw\_layer, выделяя уникальные payment | core\_payment.sql |
| sales | Формируем записи факта продаж, связывая внешние ключи из таблиц измерений | core\_sales.sql |

1. CORE DATA – слой (атрибуты)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Таблица** | **Атрибут** | **Тип атрибута** | **Первичный ключ** | **Внешний ключ** | **Атрибут из SQL (с листа «Таблицы»)** | **Комментарий к атрибуту** |
| customer\_type | customer\_type\_id | integer | Да | Нет | customer\_type\_id | Идентификатор типа клиента |
| customer\_type | customer\_type | string | Нет | Нет | customer\_type | Тип клиента (Member / Normal) |
| gender | gender\_id | integer | Да | Нет | gender\_id | Идентификатор пола |
| gender | gender | string | Нет | Нет | gender | Пол клиента |
| store | store\_id | integer | Да | Нет | store\_id | Идентификатор магазина |
| store | branch | string | Нет | Нет | branch | Филиал магазина (A, B, C) |
| store | city | string | Нет | Нет | city | Город магазина |
| product | product\_id | integer | Да | Нет | product\_id | Идентификатор продукта |
| product | product\_line | string | Нет | Нет | product\_line | Категория товара |
| product | unit\_price | float | Нет | Нет | unit\_price | Цена за единицу товара |
| payment | payment\_id | integer | Да | Нет | payment\_id | Идентификатор способа оплаты |
| payment | payment\_type | string | Нет | Нет | payment\_type | Способ оплаты (Cash / Credit card / Ewallet) |
| sales | invoice\_id | string | Да | Нет | invoice\_id | Номер чека |
| sales | customer\_type\_id | integer | Нет | Да | customer\_type\_id | Связь с таблицей customer\_type |
| sales | gender\_id | integer | Нет | Да | gender\_id | Связь с таблицей gender |
| sales | store\_id | integer | Нет | Да | store\_id | Связь с таблицей store |
| sales | product\_id | integer | Нет | Да | product\_id | Связь с таблицей product |
| sales | payment\_id | integer | Нет | Да | payment\_id | Связь с таблицей payment |
| sales | datetime | datetime | Нет | Нет | datetime | Дата и время покупки |
| sales | quantity | integer | Нет | Нет | quantity | Количество товаров |
| sales | tax | float | Нет | Нет | tax | Налог 5% |
| sales | total | float | Нет | Нет | total | Общая сумма чека |
| sales | cogs | float | Нет | Нет | cogs | Себестоимость |
| sales | gross\_income | float | Нет | Нет | gross\_income | Валовая прибыль |
| sales | rating | float | Нет | Нет | rating | Оценка клиента |



1. DATA MART – слой (таблицы)

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица** | **Логика формирования данных из Core Data Layer** | **SQL** | **Комментарий к таблице** |
| sales\_fact | Берём данные из таблицы core\_layer.sales генерируем month\_id на основе столбца datetime | core\_to\_datamart\_sales\_fact.sql |  |
| customer\_dim | Берём из таблицы customer\_type слоя core\_layer | core\_to\_datamart\_customer.sql |  |
| gender\_dim | Берём из таблицы gender слоя core\_layer | core\_to\_datamart\_gender.sql |  |
| store\_dim | Берём из таблицы store слоя core\_layer | core\_to\_datamart\_store.sql |  |
| product\_dim | Берём из таблицы product слоя core\_layer | core\_to\_datamart\_product.sql |  |
| payment\_dim | Берём из таблицы payment слоя core\_layer | core\_to\_datamart\_payment.sql |  |
| month\_dim | Генерируем с помощью ClickHouse функций дат | generate\_month\_dim.sql |  |

1. DATA MART – слой (атрибуты)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Таблица** | **Атрибут** | **Тип атрибута** | **Первичный ключ** | **Внешний ключ** | **Атрибут из SQL (с листа «Таблицы»)** | **Комментарий к атрибуту** |
| sales\_fact | invoice\_id | string | Да | Нет | invoice\_id | Идентификатор чека продажи |
| sales\_fact | customer\_type\_id | integer | Нет | Да | customer\_type\_id | Внешний ключ на customer\_dim |
| sales\_fact | gender\_id | integer | Нет | Да | gender\_id | Внешний ключ на gender\_dim |
| sales\_fact | store\_id | integer | Нет | Да | store\_id | Внешний ключ на store\_dim |
| sales\_fact | product\_id | integer | Нет | Да | product\_id | Внешний ключ на product\_dim |
| sales\_fact | payment\_id | integer | Нет | Да | payment\_id | Внешний ключ на payment\_dim |
| sales\_fact | month\_id | integer | Нет | Да | month\_id | Внешний ключ на month\_dim |
| sales\_fact | datetime | datetime | Нет | Нет | datetime | Дата и время покупки |
| sales\_fact | quantity | integer | Нет | Нет | quantity | Количество товаров |
| sales\_fact | tax | float | Нет | Нет | tax | Налог |
| sales\_fact | total | float | Нет | Нет | total | Общая сумма продажи |
| sales\_fact | cogs | float | Нет | Нет | cogs | Себестоимость |
| sales\_fact | gross\_income | float | Нет | Нет | gross\_income | Валовая прибыль |
| sales\_fact | rating | float | Нет | Нет | rating | Оценка сервиса |
| month\_dim | month\_id | integer | Да | Нет | month\_id | YYYYMM идентификатор месяца |
| month\_dim | month\_name | string | Нет | Нет | month\_name | Название месяца |
| month\_dim | number\_of\_month | integer | Нет | Нет | number\_of\_month | Номер месяца |
| month\_dim | year\_of\_month | integer | Нет | Нет | year\_of\_month | Год |
| customer\_dim | customer\_type\_id | integer | Да | Нет | customer\_type\_id | Тип покупателя |
| customer\_dim | customer\_type | string | Нет | Нет | customer\_type | Название типа |
| gender\_dim | gender\_id | integer | Да | Нет | gender\_id | Пол |
| gender\_dim | gender | string | Нет | Нет | gender | Значение пола |
| store\_dim | store\_id | integer | Да | Нет | store\_id | Идентификатор магазина |
| store\_dim | branch | string | Нет | Нет | branch | Филиал |
| store\_dim | city | string | Нет | Нет | city | Город |
| product\_dim | product\_id | integer | Да | Нет | product\_id | Идентификатор продукта |
| product\_dim | product\_line | string | Нет | Нет | product\_line | Линейка продукта |
| product\_dim | unit\_price | float | Нет | Нет | unit\_price | Цена за единицу |
| payment\_dim | payment\_id | integer | Да | Нет | payment\_id | Идентификатор оплаты |
| payment\_dim | payment\_type | string | Нет | Нет | payment\_type | Тип оплаты |

*Спасибо, что ознакомились с моим дипломным проектом!*