



30.3.2025

Дипломная работа по курсу "Дата - инженер"



Дмитрий Назарьянц
DIPLOM-DEG-34

Оглавление

1. Введение	2
2. Анализ данных.....	2
3. Формирование состава таблиц, построение ER-диаграмм и DDL-запросов	7
4. Разработка ETL-процессов.....	10
5. Формирование набора метрик и визуализация данных.....	24

1. Введение

В современном информационном мире организации сталкиваются с колоссальными объемами данных, которые требуют систематизации, надежного хранения и глубокого анализа. С ростом объемов информации, её разнообразия и сложности, эффективное управление данными становится ключевым фактором успеха для предприятий. В этом контексте концепция Дата инженерии приобретает особую значимость, объединяя лучшие практики с принципами управления данными для обеспечения гибкости, автоматизации и согласованности процессов работы с данными.

Целью данной дипломной работы является разработка и документирование процессов ETL (Extract, Transform, Load) для создания и сопровождения хранилища данных. Хранилище состоит из двух основных уровней: нормализованного хранилища данных (NDS) и схемы звезда (DDS). Реализация этих процессов позволит обеспечить эффективное хранение, структурирование и обработку данных, создавая прочную основу для анализа и принятия обоснованных бизнес-решений.

На заключительном этапе работы, на основе данных из схемы звезда (DDS), был разработан интерактивный дашборд, предоставляющий наглядное представление ключевых аспектов бизнес-деятельности. Этот инструмент визуализации позволяет оперативно анализировать данные, выявлять закономерности и принимать стратегические решения на основе объективной информации.

Для реализации дипломной работы использовались:

- Дистрибутив Anaconda (Jupyter Lab) - для анализа данных
- Сервис dbdiagram.io - для построения ER-диаграмм
- ClickHouse и PostgreSQL (Docker) - для хранения данных
- Apache Airflow (Docker) - для ETL-процессов
- Power BI Desktop - для построения дашборда

2. Анализ данных

Контекст датасета: датасет Supermarket Sales представляет из себя срез исторических данных о продажах товаров в 3 филиалах компании за 3 месяца.

Атрибуты:

1. Invoice ID: программно-генерируемый идентификационный номер счета-фактуры
2. Branch: название филиала компании
3. City: местонахождение филиала (город)
4. Customer Type: тип покупателя (наличие клубной карты)
5. Gender: пол покупателя
6. Product Line: продуктовая линейка
7. Unit Price: цена единицы товара в долларах
8. Quantity: количество проданных товаров
9. Tax: сумма взимаемого налога с продажи (5%)
10. Total: общая стоимость продажи, включая налоги
11. Date: дата продажи
12. Time: время продажи
13. Payment: метод оплаты

14. COGS: себестоимость проданных товаров
15. Gross Profit Percentage: процент прибыли
16. Gross Revenue: прибыль с продажи
17. Rating: рейтинг покупки от покупателя (по шкале от 1 до 10)

Для анализа данных воспользуемся библиотеками pandas, matplotlib и seaborn.

Загрузка данных в Pandas DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('supermarket_sales - Sheet1.csv', delimiter=',')
print(df.head(3))
```

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	

	Product line	Unit price	Quantity	Tax 5%	Total	Date	\
0	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	
1	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	
2	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	

	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	10:29	Cash	76.40	4.761905	3.8200	9.6
2	13:23	Credit card	324.31	4.761905	16.2155	7.4

Данные успешно загружены.

Проверка типов данных, количества строк и Null значений:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Invoice ID             1000 non-null  object
1   Branch                 1000 non-null  object
2   City                   1000 non-null  object
3   Customer type          1000 non-null  object
4   Gender                 1000 non-null  object
5   Product line           1000 non-null  object
6   Unit price             1000 non-null  float64
7   Quantity               1000 non-null  int64
8   Tax 5%                 1000 non-null  float64
9   Total                  1000 non-null  float64
```

```
10 Date          1000 non-null object
11 Time          1000 non-null object
12 Payment       1000 non-null object
13 cogs          1000 non-null float64
14 gross margin percentage 1000 non-null float64
15 gross income  1000 non-null float64
16 Rating        1000 non-null float64
```

dtypes: float64(7), int64(1), object(9)

memory usage: 132.9+ KB

В датафрейме 1000 строк, Null значения отсутствуют.

Проверка строк на дубликаты:

```
df.duplicated().any()
```

False

Дубликаты отсутствуют.

Вывод списков представленных филиалов, городов, типов клиентов, методов оплаты и продуктовых линейках:

```
print(df['Branch'].unique())
```

```
print(df['City'].unique())
```

```
print(df['Customer type'].unique())
```

```
print(df['Gender'].unique())
```

```
print(df['Product line'].unique())
```

```
print(df['Payment'].unique())
```

```
['A' 'C' 'B']
```

```
['Yangon' 'Naypyitaw' 'Mandalay']
```

```
['Member' 'Normal']
```

```
['Female' 'Male']
```

```
['Health and beauty' 'Electronic accessories' 'Home and lifestyle'
```

```
'Sports and travel' 'Food and beverages' 'Fashion accessories']
```

```
['Ewallet' 'Cash' 'Credit card']
```

Категориальные переменные определены.

Вывод статистики по всем столбцам датафрейма:

```
df.describe(include='all')
```

Из полученной статистики узнаем:

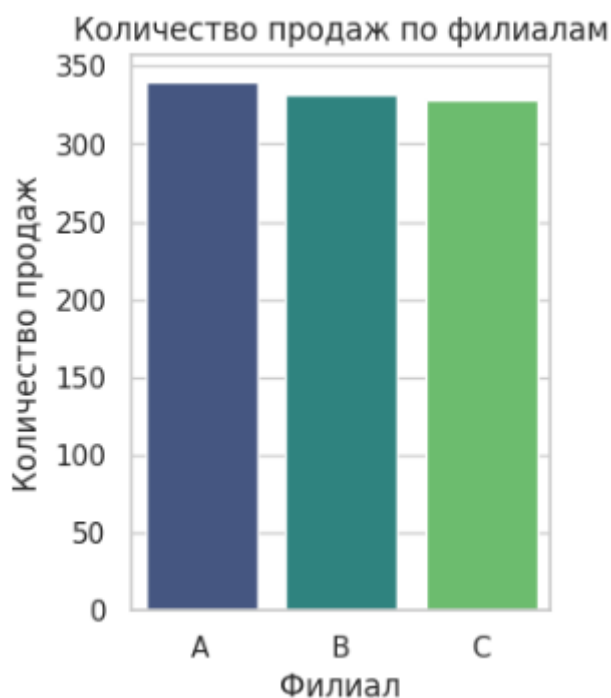
1. В данных почти равное распределение как по типу клиента (*Member / Normal*), так и по полу (*Female / Male*). В обоих случаях это 501 на 499.
2. Наибольшей популярностью пользуется категория *Fashion accessories*. На нее приходится 178 продаж.
3. Средняя стоимость единицы товара \$55.67.
4. Среднее количество товаров в покупке 5.51.
5. Средний налог с продажи \$15.38.
6. Средний доход с продажи \$322.97.
7. Средняя прибыль с продажи \$15.38.

8. Средний рейтинг покупки 6.97 баллов.

Построение столбчатой диаграммы по распределению продаж на филиалы:

```
# группировка данных
branch_stat = df.groupby('Branch').nunique().reset_index()
# построение графика
plt.figure(figsize=(3, 4))
sns.set(style="whitegrid")
sns.barplot(x='Branch', y='Invoice ID', data=branch_stat, palette="viridis")

plt.title('Количество продаж по филиалам')
plt.xlabel('Филиал')
plt.ylabel('Количество продаж')
plt.show()
```



Из диаграммы видно, что больше всего продаж приходится на филиал А, но в целом распределение почти равное.

Подсчет средней разницы в количестве продаж между филиалами:

```
average_percentage_difference = (branch_stat['Invoice ID'].max() - branch_stat['Invoice ID'].min()) /
branch_stat['Invoice ID'].mean() * 100
print('Разница составляет: ', average_percentage_difference.round(3), '%')
Разница составляет: 3.6 %
```

Распределение продаж по продуктовым линейкам:

```
product_line_stat = df.groupby('Product line')['Invoice ID'].nunique().reset_index()

plt.figure(figsize=(4, 4))
sns.set(style="whitegrid")
```

```
sns.barplot(x='Product line', y='Invoice ID', data=product_line_stat, palette="viridis")
```

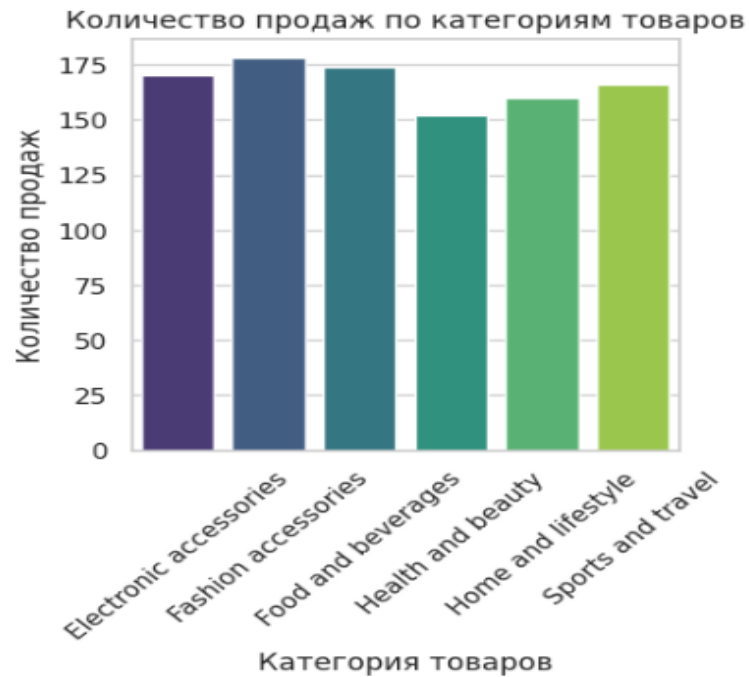
```
plt.title('Количество продаж по категориям товаров')
```

```
plt.xlabel('Категория товаров')
```

```
plt.ylabel('Количество продаж')
```

```
plt.xticks(rotation=45, ha='center')
```

```
plt.show()
```



Из полученной диаграммы видно, что наибольшее число продаж приходится на категорию *Fashion Accessories*, наименьшее на категорию *Health and beauty*.

Построение тепловой карты (город - метод оплаты):

```
sns.heatmap(pd.crosstab(df['City'], df['Payment']), cmap="YlGnBu")
```

```
plt.title('Тепловая карта по городам и методам оплаты')
```

```
plt.xlabel('Метод оплаты')
```

```
plt.ylabel('Город')
```

```
plt.show()
```



Из полученной карты можно сделать вывод о том, что:

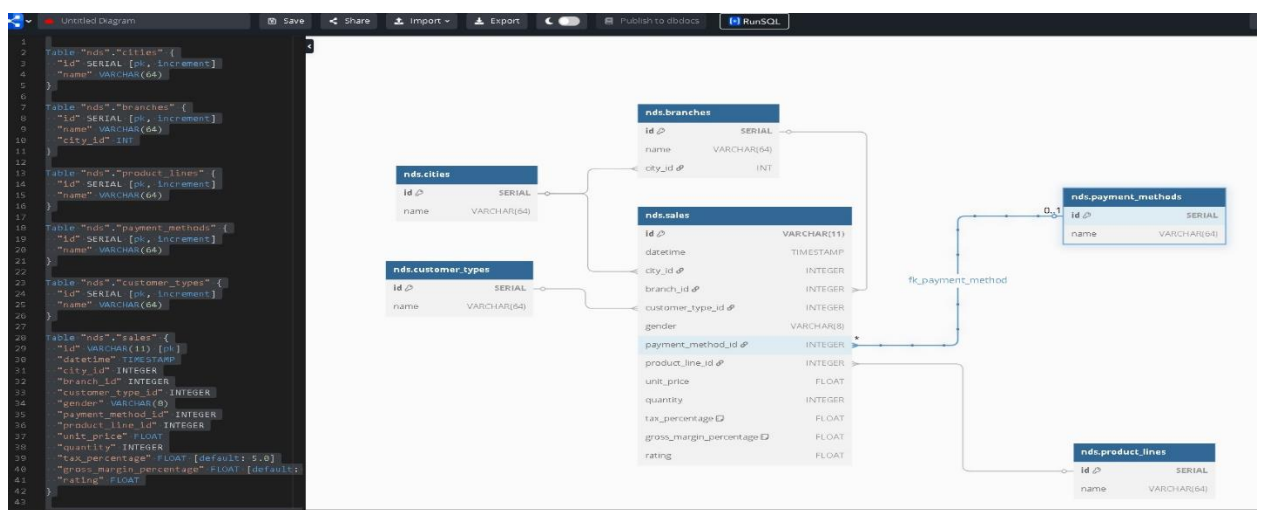
1. Метод оплаты *Cash* чаще всего используют в городе *Naypyitaw*
2. Метод оплаты *Credit Card* наиболее часто используют в городе *Mandalay*
3. Метод оплаты *Ewallet* наиболее часто используют в городе *Yangon*

Общие выводы:

По результатам анализа были определены основных характеристики, категориальные признаки, а также выполнена базовая проверка данных на ошибки и Null значения.

3. Формирование состава таблиц, построение ER-диаграмм и DDL-запросов

Нормализованная схема данных (NDS)



В процессе решения данной задачи был сформирован состав таблиц нормализованной схемы данных. Были исключены исчисляемые атрибуты (tax_amount, total, cogs, gross_income).

Нормализация данных помогает уменьшить избыточность информации и повысить эффективность хранения.

Нормализованная схема включает в себя следующие таблицы:

1. Города (nds.cities):
 - Таблица для хранения городов.
2. Филиалы (nds.branches):
3. Продуктовые линейки (nds.product_lines):
 - Таблица для хранения категорий товаров.
4. Способы оплаты (nds.payment_methods):
 - Таблица для хранения доступных методов оплаты.
5. Типы клиентов (nds.customer_types):
 - Таблица для хранения различных типов клиентов.
6. Продажи (nds.sales):
 - Основная таблица для хранения информации о продажах. Отдельные столбцы для даты, времени, цены, количества и других характеристик продажи. Внешние ключи связывают записи в этой таблице с соответствующими записями в таблицах городов, филиалов, категорий товаров, способов оплаты и типов клиентов.

Таблицы фактов и измерений по схеме звезда (DDS)

В процессе решения данной задачи был сформирован состав таблиц фактов и измерений по схеме звезда, предназначенной для анализа данных о покупках.

Схема звезда (Star Schema) представляет собой тип схемы "факты и измерения", где факты (данные о покупках) хранятся в одной основной таблице, а измерения (характеристики покупки) хранятся в отдельных таблицах, соединенных через внешние ключи. Данные в такой схеме денормализованы для увеличения эффективности при запросах.

Диаграмма:

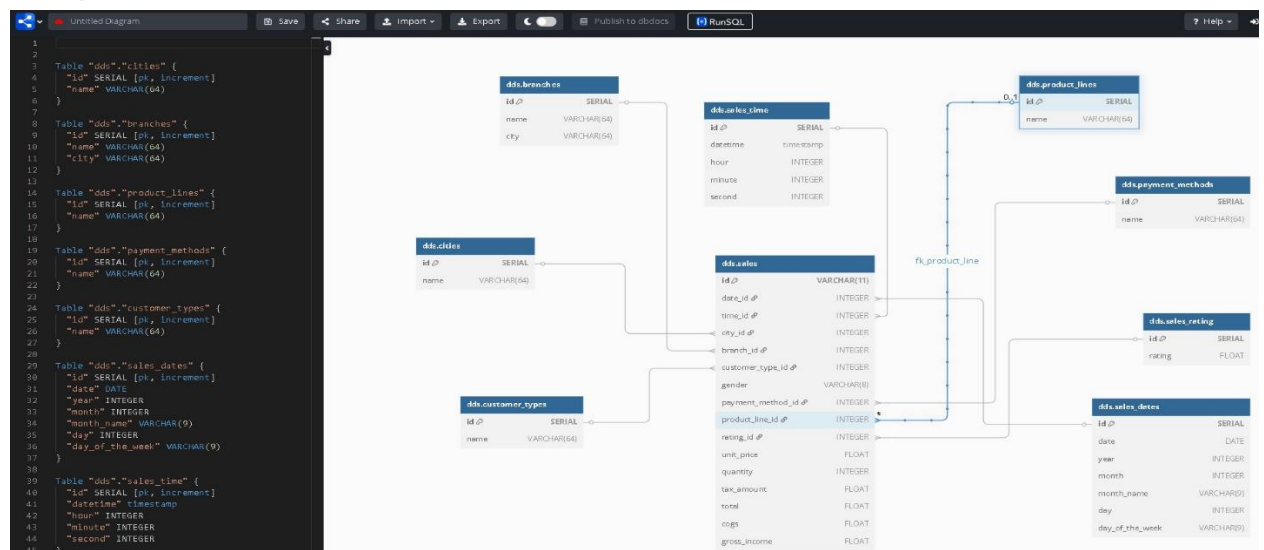


Схема звезды включает в себя следующие таблицы:

1. Города (dds.cities):
 - Таблица для хранения городов.
2. Филиалы (dds.branches):
 - Таблица для хранения информации о филиалах, ссылающаяся на таблицу городов.
3. Продуктовые линейки (dds.product_lines):
 - Таблица для хранения категорий товаров.
4. Способы оплаты (dds.payment_methods):
 - Таблица для хранения доступных методов оплаты.
5. Типы клиентов (dds.customer_types):
 - Таблица для хранения различных типов клиентов.
6. Дата продажи (dds.sales_dates):
 - Таблица для хранения информации о датах продажи, включая элементы дат.
7. Время продажи (dds.sales_time):
 - Таблица для хранения информации о времени продажи, включая элементы времени.
8. Рейтинг продаж (dds.sales_rating):
 - Таблица для хранения рейтингов продаж.
9. Продажи (dds.sales):
 - Основная таблица для хранения фактов о продажах. Содержит информацию о продажах, включая дату, время, место, тип клиента, продукт, стоимость, налог и другие характеристики. Связана с таблицами измерений через внешние ключи.

Структура и типы данных таблицы для хранения исходных данных

Диаграмма:

unprocessed_data	
id	varchar
datetime	datetime
city	varchar
branch	varchar
customer_type	varchar
gender	varchar
product_line	varchar
unit_price	float
quantity	integer
tax	float
total	float
payment_method	varchar
cogs	float
gross_margin_percentage	float
gross_income	float
rating	float
insert_time	datetime

 dbdiagram.io

В данной таблице изменены названия атрибутов, соединены время и дата (преобразованы в тип данных DateTime), а также добавлен столбец insert_time - фактическое время на момент insert'a строки в таблицу.

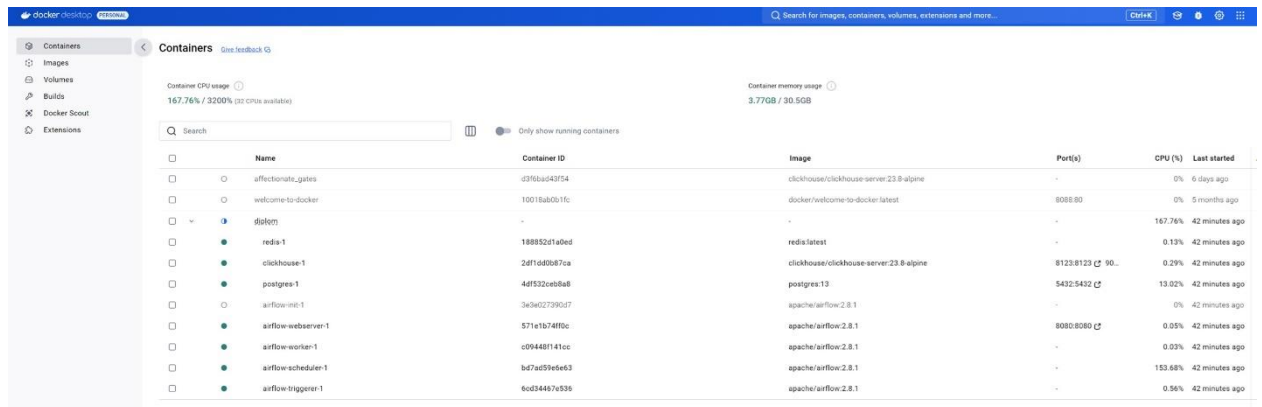
4. Разработка ETL-процессов

Для решения этой задачи было решено разделить процессы первичной выгрузки данных из источника и перемещение их в требуемые схемы на 2 DAG. Все окружение для реализации процессов поднято: в контейнеризированной среде через PowerShell

```
Windows PowerShell
Установите последнюю версию PowerShell для новых функций и улучшения! https://aka.ms/PSWindows

PS C:\Users\max_f> cd C:\Users\max_f\Diplom
PS C:\Users\max_f\Diplom> docker-compose up -d
time="2025-03-14T20:25:01+03:00" level=warning msg="The \"AIRFLOW_UID\" variable is not set. Defaulting to a blank string."
time="2025-03-14T20:25:01+03:00" level=warning msg="The \"AIRFLOW_UID\" variable is not set. Defaulting to a blank string."
[+] Running 59/8
  ✓ airflow-worker Pulled                                24.1s
  ✓ postgres Pulled                                     26.8s
  ✓ clickhouse Pulled                                    26.4s
  ✓ redis Pulled                                         26.4s
  ✓ airflow-webserver Pulled                             24.1s
  ✓ airflow-scheduler Pulled                             24.1s
  ✓ airflow-init Pulled                                 24.1s
  ✓ airflow-triggerer Pulled                             24.1s
[+] Running 11/11
  ✓ Network diplom_default                               0.0s
  ✓ Volume "diplom_postgres-db-volume"                  0.0s
  ✓ Volume "diplom_clickhouse-db-volume"               0.0s
  ✓ Container diplom-redis-1                            7.7s
  ✓ Container diplom-clickhouse-1                       1.4s
  ✓ Container diplom-postgres-1                         7.7s
  ✓ Container diplom-airflow-init-1                     14.5s
  ✓ Container diplom-airflow-worker-1                   15.3s
  ✓ Container diplom-airflow-scheduler-1                14.8s
  ✓ Container diplom-airflow-triggerer-1                15.4s
  ✓ Container diplom-airflow-webserver-1                15.4s
PS C:\Users\max_f\Diplom>
```

И через Docker передается в AIRFLOW DAGs



Name	Container ID	Image	Port(s)	CPU (%)	Last started
diplom-redis-1	18885261a0ed	redis:latest	-	0.13%	42 minutes ago
diplom-clickhouse-1	2d1f0d0b67ca	clickhouse/clickhouse-server:23.8-alpine	8123:8123	0.29%	42 minutes ago
diplom-postgres-1	4d1532ce08a8	postgres:13	5432:5432	13.02%	42 minutes ago
diplom-airflow-init-1	3e0e027390d7	apache/airflow:2.8.1	-	0%	42 minutes ago
diplom-airflow-worker-1	571e1b74f0c	apache/airflow:2.8.1	8080:8080	0.05%	42 minutes ago
diplom-airflow-scheduler-1	c094401f14cc	apache/airflow:2.8.1	-	0.03%	42 minutes ago
diplom-airflow-triggerer-1	b07a059f6e63	apache/airflow:2.8.1	-	153.68%	42 minutes ago
diplom-airflow-webserver-1	60d34497e536	apache/airflow:2.8.1	-	0.56%	42 minutes ago

Краткое описание: DAG выполняет выгрузку данных из источника (через Kaggle API), валидацию (проверка на Null значения, дубликаты строк, наличие валидных значений атрибутов, а также соответствие формата Invoice ID регулярному выражению) и преобразование типов (раздельные атрибуты даты и времени преобразовываются в один) и загрузку данных в ClickHouse.

Код Airflow DAG ([data_load_dag.py](#)):

```
from airflow import DAG
from airflow.operators.empty import EmptyOperator
from airflow.operators.python import PythonOperator
from datetime import datetime
import pandas as pd
from clickhouse_driver import Client
```

```

def create_table():
    client = Client(host='clickhouse', port=9000, database='default')
    client.execute("""
        CREATE TABLE IF NOT EXISTS unprocessed_data (
            id String,
            datetime DateTime,
            city String,
            branch String,
            customer_type String,
            gender String,
            product_line String,
            unit_price Decimal(10,2),
            quantity Int32,
            tax Decimal(10,2),
            total Decimal(10,2),
            payment_method String,
            cogs Decimal(10,2),
            gross_margin_percentage Float32,
            gross_income Decimal(10,2),
            rating Float32,
            insert_time DateTime DEFAULT now()
        ) ENGINE = MergeTree
        PARTITION BY toYYYYMM(datetime)
        ORDER BY (id, datetime)
    """)
    client.disconnect()

def load_to_clickhouse():
    df = pd.read_csv('/opt/airflow/data/supermarket_sales - Sheet1.csv')
    df = df.rename(columns={
        'Invoice ID': 'id',
        'City': 'city',
        'Branch': 'branch',
        'Customer type': 'customer_type',
        'Gender': 'gender',
        'Product line': 'product_line',
        'Unit price': 'unit_price',
        'Quantity': 'quantity',
        'Tax 5%': 'tax',
        'Total': 'total',
        'Payment': 'payment_method',
        'gross margin percentage': 'gross_margin_percentage',
        'gross income': 'gross_income',
        'Rating': 'rating'
    })

    # Создаем колонку datetime из Date и Time
    df['datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'])
    df.drop(columns=['Date', 'Time'], inplace=True)

    # Добавляем колонку insert_time
    df['insert_time'] = datetime.now()

    client = Client(host='clickhouse', port=9000, database='default')
    client.execute("INSERT INTO unprocessed_data VALUES", df.to_dict('records'))
    client.disconnect()

with DAG(
    'data_pipeline',

```

```

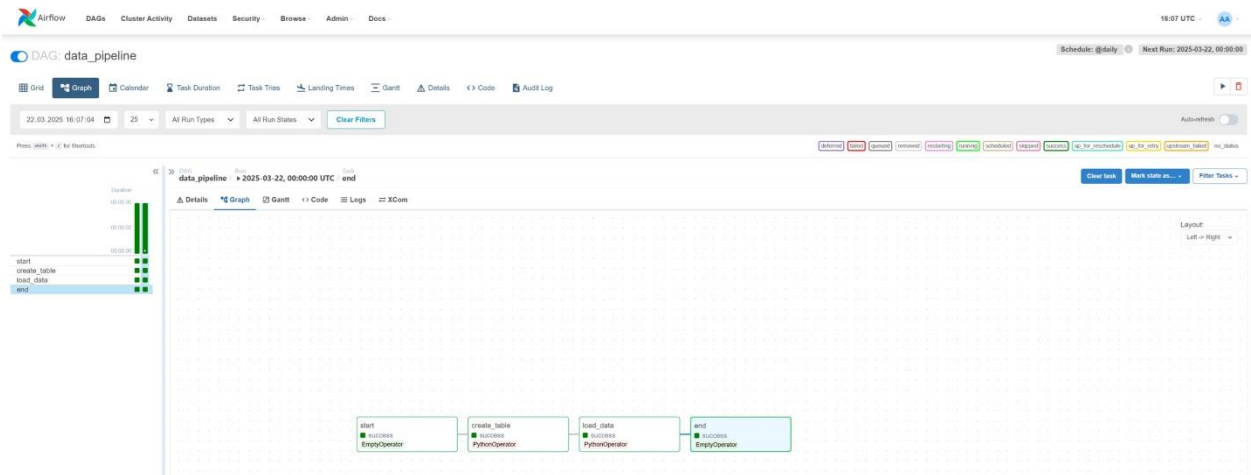
schedule_interval='@daily',
start_date=datetime(2024, 1, 1),
catchup=False
) as dag:

start = EmptyOperator(task_id='start')
create_table_task = PythonOperator(
    task_id='create_table',
    python_callable=create_table
)
load_task = PythonOperator(
    task_id='load_data',
    python_callable=load_to_clickhouse
)
end = EmptyOperator(task_id='end')

start >> create_table_task >> load_task >> end

```

Граф задач:



Обработка и загрузка данных в NDS и DDS, перемещение исходных данных

Краткое описание: DAG выполняет выгрузку данных из источника (таблица исходных данных в ClickHouse), маппинг значений (преобразование в id), загрузку данных в NDS и DDS, а также перемещение исходных данных из таблицы `unprocessed_data` в таблицу `processed_data`, очистку таблицы `unprocessed_data`.

Код Airflow DAG ([data_processing_dag.py](#)):

```

from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.empty import EmptyOperator
from datetime import datetime, timedelta
import pandas as pd
from clickhouse_driver import Client
import psycopg2
from psycopg2 import sql

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2024, 2, 1),
    'retries': 1,

```

```

'retry_delay': timedelta(minutes=5)
}

clickhouse_params = {
    'host': 'clickhouse',
    'port': 9000,
    'database': 'default'
}

postgres_params = {
    'host': 'postgres',
    'port': 5432,
    'user': 'airflow',
    'password': 'airflow',
    'database': 'postgres'
}

column_map = {
    'cities': 'city',
    'branches': 'branch',
    'product_lines': 'product_line',
    'payment_methods': 'payment_method',
    'customer_types': 'customer_type'
}

POSTGRES_TABLES = {
    'nds': [
        "CREATE SCHEMA IF NOT EXISTS nds",
        "CREATE TABLE IF NOT EXISTS nds.cities (id SERIAL PRIMARY KEY, name VARCHAR(64) UNIQUE)",
        "CREATE TABLE IF NOT EXISTS nds.branches (id SERIAL PRIMARY KEY, name VARCHAR(64) UNIQUE,
city_id INT REFERENCES nds.cities(id))",
        "CREATE TABLE IF NOT EXISTS nds.product_lines (id SERIAL PRIMARY KEY, name VARCHAR(64)
UNIQUE)",
        "CREATE TABLE IF NOT EXISTS nds.payment_methods (id SERIAL PRIMARY KEY, name VARCHAR(64)
UNIQUE)",
        "CREATE TABLE IF NOT EXISTS nds.customer_types (id SERIAL PRIMARY KEY, name VARCHAR(64)
UNIQUE)",
        """
        CREATE TABLE IF NOT EXISTS nds.sales (
            id VARCHAR(11) PRIMARY KEY,
            datetime TIMESTAMP,
            city_id INT REFERENCES nds.cities(id),
            branch_id INT REFERENCES nds.branches(id),
            customer_type_id INT REFERENCES nds.customer_types(id),
            gender VARCHAR(8),
            payment_method_id INT REFERENCES nds.payment_methods(id),
            product_line_id INT REFERENCES nds.product_lines(id),
            unit_price FLOAT,
            quantity INT,
            tax DECIMAL(10,2),
            total DECIMAL(10,2),
            rating FLOAT
        )
        """
    ],
    'dds': [
        "CREATE SCHEMA IF NOT EXISTS dds",
        "CREATE TABLE IF NOT EXISTS dds.cities (id SERIAL PRIMARY KEY, name VARCHAR(64))",

```

```

"CREATE TABLE IF NOT EXISTS dds.branches (id SERIAL PRIMARY KEY, name VARCHAR(64), city
VARCHAR(64))",
"CREATE TABLE IF NOT EXISTS dds.product_lines (id SERIAL PRIMARY KEY, name VARCHAR(64))",
"CREATE TABLE IF NOT EXISTS dds.payment_methods (id SERIAL PRIMARY KEY, name VARCHAR(64))",
"CREATE TABLE IF NOT EXISTS dds.customer_types (id SERIAL PRIMARY KEY, name VARCHAR(64))",
"CREATE TABLE IF NOT EXISTS dds.sales_dates (id SERIAL PRIMARY KEY, date DATE, year INT, month INT,
month_name VARCHAR(9), day INT, day_of_the_week VARCHAR(9))",
"CREATE TABLE IF NOT EXISTS dds.sales_time (id SERIAL PRIMARY KEY, datetime TIMESTAMP, hour INT,
minute INT, second INT)",
"CREATE TABLE IF NOT EXISTS dds.sales_rating (id SERIAL PRIMARY KEY, rating FLOAT)",
""""

CREATE TABLE IF NOT EXISTS dds.sales (
    id VARCHAR(11) PRIMARY KEY,
    date_id INT REFERENCES dds.sales_dates(id),
    time_id INT REFERENCES dds.sales_time(id),
    city VARCHAR(64),
    branch VARCHAR(64),
    customer_type VARCHAR(64),
    gender VARCHAR(8),
    payment_method VARCHAR(64),
    product_line VARCHAR(64),
    rating_id INT REFERENCES dds.sales_rating(id),
    unit_price FLOAT,
    quantity INT,
    tax DECIMAL(10,2),
    total DECIMAL(10,2),
    cogs DECIMAL(10,2),
    gross_income DECIMAL(10,2)
)
""""
]
}

REFERENCE_DATA = {
    'nds': {
        'cities': ['Yangon', 'Mandalay', 'Naypyitaw'],
        'branches': [('A', 1), ('B', 2), ('C', 3)],
        'product_lines': ['Sports and travel', 'Food and beverages', 'Health and beauty',
            'Fashion accessories', 'Electronic accessories', 'Home and lifestyle'],
        'customer_types': ['Normal', 'Member'],
        'payment_methods': ['Cash', 'Credit card', 'Ewallet']
    },
    'dds': {
        'cities': ['Yangon', 'Mandalay', 'Naypyitaw'],
        'branches': [('A', 'Yangon'), ('B', 'Mandalay'), ('C', 'Naypyitaw')],
        'product_lines': ['Sports and travel', 'Food and beverages', 'Health and beauty',
            'Fashion accessories', 'Electronic accessories', 'Home and lifestyle'],
        'customer_types': ['Normal', 'Member'],
        'payment_methods': ['Cash', 'Credit card', 'Ewallet']
    }
}

def create_postgres_tables(**kwargs):
    """Создание таблиц и заполнение справочников в PostgreSQL"""
    conn = psycopg2.connect(**kwargs['postgres_params'])
    cursor = conn.cursor()
    try:
        # Создание схем
        cursor.execute("CREATE SCHEMA IF NOT EXISTS nds")

```

```

cursor.execute("CREATE SCHEMA IF NOT EXISTS dds")

# Создание таблиц для NDS
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.cities (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) UNIQUE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.branches (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) UNIQUE,
        city_id INT REFERENCES nds.cities(id)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.product_lines (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) UNIQUE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.payment_methods (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) UNIQUE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.customer_types (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64) UNIQUE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS nds.sales (
        id VARCHAR(11) PRIMARY KEY,
        datetime TIMESTAMP,
        city_id INT REFERENCES nds.cities(id),
        branch_id INT REFERENCES nds.branches(id),
        customer_type_id INT REFERENCES nds.customer_types(id),
        gender VARCHAR(8),
        payment_method_id INT REFERENCES nds.payment_methods(id),
        product_line_id INT REFERENCES nds.product_lines(id),
        unit_price FLOAT,
        quantity INT,
        tax DECIMAL(10,2),
        total DECIMAL(10,2),
        rating FLOAT
    )
""")

# Создание таблиц для DDS
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.cities (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64)
    )
""")

```



```

cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.branches (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64),
        city VARCHAR(64)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.product_lines (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.payment_methods (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.customer_types (
        id SERIAL PRIMARY KEY,
        name VARCHAR(64)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.sales_dates (
        id SERIAL PRIMARY KEY,
        date DATE,
        year INT,
        month INT,
        month_name VARCHAR(9),
        day INT,
        day_of_the_week VARCHAR(9)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.sales_time (
        id SERIAL PRIMARY KEY,
        datetime TIMESTAMP,
        hour INT,
        minute INT,
        second INT
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.sales_rating (
        id SERIAL PRIMARY KEY,
        rating FLOAT
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS dds.sales (
        id VARCHAR(11) PRIMARY KEY,
        date_id INT REFERENCES dds.sales_dates(id),
        time_id INT REFERENCES dds.sales_time(id),
        city VARCHAR(64),
        branch VARCHAR(64),
        customer_type VARCHAR(64),

```

```

        gender VARCHAR(8),
        payment_method VARCHAR(64),
        product_line VARCHAR(64),
        rating_id INT REFERENCES dds.sales_rating(id),
        unit_price FLOAT,
        quantity INT,
        tax DECIMAL(10,2),
        total DECIMAL(10,2),
        cogs DECIMAL(10,2),
        gross_income DECIMAL(10,2)
    )
    """)

# Заполнение справочников
for city in REFERENCE_DATA['nds']['cities']:
    cursor.execute("INSERT INTO nds.cities (name) VALUES (%s) ON CONFLICT DO NOTHING", (city,))

for branch, city_id in REFERENCE_DATA['nds']['branches']:
    cursor.execute("INSERT INTO nds.branches (name, city_id) VALUES (%s, %s) ON CONFLICT DO
NOTHING",
                    (branch, city_id))

for name in REFERENCE_DATA['nds']['product_lines']:
    cursor.execute("INSERT INTO nds.product_lines (name) VALUES (%s) ON CONFLICT DO NOTHING",
                    (name,))

for name in REFERENCE_DATA['nds']['customer_types']:
    cursor.execute("INSERT INTO nds.customer_types (name) VALUES (%s) ON CONFLICT DO NOTHING",
                    (name,))

for name in REFERENCE_DATA['nds']['payment_methods']:
    cursor.execute("INSERT INTO nds.payment_methods (name) VALUES (%s) ON CONFLICT DO
NOTHING", (name,))

# Заполнение DDS справочников
for city in REFERENCE_DATA['dds']['cities']:
    cursor.execute("INSERT INTO dds.cities (name) VALUES (%s) ON CONFLICT DO NOTHING", (city,))

for branch, city in REFERENCE_DATA['dds']['branches']:
    cursor.execute("INSERT INTO dds.branches (name, city) VALUES (%s, %s) ON CONFLICT DO
NOTHING",
                    (branch, city))

for name in REFERENCE_DATA['dds']['product_lines']:
    cursor.execute("INSERT INTO dds.product_lines (name) VALUES (%s) ON CONFLICT DO NOTHING",
                    (name,))

for name in REFERENCE_DATA['dds']['customer_types']:
    cursor.execute("INSERT INTO dds.customer_types (name) VALUES (%s) ON CONFLICT DO NOTHING",
                    (name,))

for name in REFERENCE_DATA['dds']['payment_methods']:
    cursor.execute("INSERT INTO dds.payment_methods (name) VALUES (%s) ON CONFLICT DO
NOTHING", (name,))

conn.commit()
print("Таблицы и справочники в PostgreSQL созданы/обновлены")
finally:
    cursor.close()

```

```

conn.close()

def create_clickhouse_tables(**kwargs):
    client = Client(**kwargs['clickhouse_params'])
    try:
        client.execute("""
            CREATE TABLE IF NOT EXISTS unprocessed_data (
                id String,
                datetime DateTime,
                city String,
                branch String,
                customer_type String,
                gender String,
                product_line String,
                unit_price Decimal(10,2),
                quantity Int32,
                tax Decimal(10,2),
                total Decimal(10,2),
                payment_method String,
                cogs Decimal(10,2),
                gross_income Decimal(10,2),
                rating Float32
            ) ENGINE = MergeTree
            PARTITION BY toYYYYMM(datetime)
            ORDER BY (id, datetime)
        """)

        client.execute("""
            CREATE TABLE IF NOT EXISTS processed_data (
                id String,
                datetime DateTime,
                city String,
                branch String,
                customer_type String,
                gender String,
                product_line String,
                unit_price Decimal(10,2),
                quantity Int32,
                tax Decimal(10,2),
                total Decimal(10,2),
                payment_method String,
                cogs Decimal(10,2),
                gross_income Decimal(10,2),
                rating Float32
            ) ENGINE = MergeTree
            PARTITION BY toYYYYMM(datetime)
            ORDER BY (id, datetime)
        """)
    finally:
        client.disconnect()

def extract_unprocessed_data(**kwargs):
    client = Client(**kwargs['clickhouse_params'])
    df = pd.DataFrame()
    try:
        tables = client.execute("SHOW TABLES FROM default")
        if 'unprocessed_data' not in [t[0] for t in tables]:
            raise ValueError("Таблица unprocessed_data не найдена")

```

```

# ЯВНОЕ УКАЗАНИЕ КОЛОНКИ ЧЕРЕЗ SELECT
data = client.execute(
    """
    SELECT id, datetime, city, branch, customer_type, gender,
           product_line, unit_price, quantity, tax, total,
           payment_method, cogs, gross_income, rating
    FROM unprocessed_data
    """
)

columns = [
    'id', 'datetime', 'city', 'branch', 'customer_type',
    'gender', 'product_line', 'unit_price', 'quantity',
    'tax', 'total', 'payment_method', 'cogs', 'gross_income', 'rating'
]

df = pd.DataFrame(data, columns=columns)

# Дополнительная проверка
for col in ['insert_time', 'gross_margin_percentage']:
    if col in df.columns:
        df.drop(columns=[col], inplace=True)

print("Колонки после извлечения:", df.columns.tolist())
kwargs['ti'].xcom_push(key='clean_data', value=df)
finally:
    client.disconnect()
return df

def dataframe_values_mapping(postgres_params, schema, column_map, df):
    conn = psycopg2.connect(**postgres_params)
    cursor = conn.cursor()
    try:
        for table_name, column in column_map.items():
            cursor.execute(
                sql.SQL("SELECT EXISTS (SELECT FROM pg_tables WHERE schemaname = %s AND tablename = %s)"),
                (schema, table_name)
            )
            exists = cursor.fetchone()[0]
            if not exists:
                raise ValueError(f"Таблица {schema}.{table_name} не существует")

            cursor.execute(
                sql.SQL("SELECT name, id FROM {}.{}").format(
                    sql.Identifier(schema),
                    sql.Identifier(table_name)
                )
            )
            mapping = {name: id_ for id_, name in cursor.fetchall()}
            if not mapping:
                raise ValueError(f"Справочник {schema}.{table_name} пуст")

            df[column] = df[column].map(mapping).fillna(0).astype(int)
    return df
finally:
    cursor.close()
    conn.close()

```

```

def insert_dates(cursor, row, schema):
    cursor.execute(
        sql.SQL("""
            INSERT INTO {schema}.sales_dates (date, year, month, month_name, day, day_of_the_week)
            VALUES (%s, %s, %s, %s, %s, %s)
            RETURNING id
        """).format(schema=sql.Identifier(schema)),
        (
            row['datetime'].date(),
            row['datetime'].year,
            row['datetime'].month,
            row['datetime'].strftime('%B'),
            row['datetime'].day,
            row['datetime'].strftime('%A')
        )
    )
    return cursor.fetchone()[0]

def insert_time(cursor, row, schema):
    cursor.execute(
        sql.SQL("""
            INSERT INTO {schema}.sales_time (datetime, hour, minute, second)
            VALUES (%s, %s, %s, %s)
            RETURNING id
        """).format(schema=sql.Identifier(schema)),
        (
            row['datetime'],
            row['datetime'].hour,
            row['datetime'].minute,
            row['datetime'].second
        )
    )
    return cursor.fetchone()[0]

def insert_rating(cursor, row, schema):
    cursor.execute(
        sql.SQL("INSERT INTO {schema}.sales_rating (rating) VALUES (%s) RETURNING id").format(
            schema=sql.Identifier(schema),
            (row['rating'],)
        )
    )
    return cursor.fetchone()[0]

def load_to_nds(**kwargs):
    df = kwargs['ti'].xcom_pull(task_ids='extract_unprocessed_data', key='clean_data')
    mapped_df = dataframe_values_mapping(
        kwargs['postgres_params'],
        'nds',
        column_map,
        df.copy()
    )

    conn = psycopg2.connect(**kwargs['postgres_params'])
    cursor = conn.cursor()
    try:
        for _, row in mapped_df.iterrows():
            cursor.execute(
                sql.SQL("""
                    INSERT INTO nds.sales (
                        id, datetime, city_id, branch_id,

```

```

        customer_type_id, gender,
        payment_method_id, product_line_id,
        unit_price, quantity, tax, total, rating
    ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    ON CONFLICT DO NOTHING
    """),
    (
        row['id'], row['datetime'], row['city'], row['branch'],
        row['customer_type'], row['gender'],
        row['payment_method'], row['product_line'],
        row['unit_price'], row['quantity'],
        row['tax'], row['total'], row['rating']
    )
    )
    conn.commit()
finally:
    cursor.close()
    conn.close()

def load_to_dds(**kwargs):
    df = kwargs['ti'].xcom_pull(task_ids='extract_unprocessed_data', key='clean_data')
    mapped_df = dataframe_values_mapping(
        kwargs['postgres_params'],
        'dds',
        column_map,
        df.copy()
    )

    conn = psycopg2.connect(**kwargs['postgres_params'])
    cursor = conn.cursor()
    try:
        for _, row in mapped_df.iterrows():
            date_id = insert_dates(cursor, row, 'dds')
            time_id = insert_time(cursor, row, 'dds')
            rating_id = insert_rating(cursor, row, 'dds')

        cursor.execute(
            sql.SQL("""
                INSERT INTO dds.sales (
                    id, date_id, time_id, city, branch,
                    customer_type, gender, payment_method,
                    product_line, rating_id,
                    unit_price, quantity, tax, total,
                    cogs, gross_income
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT DO NOTHING
            """),
            (
                row['id'], date_id, time_id,
                row['city'], row['branch'],
                row['customer_type'], row['gender'],
                row['payment_method'], row['product_line'],
                rating_id,
                row['unit_price'], row['quantity'],
                row['tax'], row['total'],
                row['cogs'], row['gross_income']
            )
        )
    conn.commit()

```

```

finally:
    cursor.close()
    conn.close()

def transfer_of_processed_data(**kwargs):
    df = kwargs['ti'].xcom_pull(task_ids='extract_unprocessed_data', key='clean_data')

    # Удаление всех возможных проблемных колонок
    columns_to_drop = ['insert_time', 'gross_margin_percentage']
    for col in columns_to_drop:
        if col in df.columns:
            df.drop(columns=[col], inplace=True)

    # Фильтрация только нужных колонок
    expected_columns = [
        'id', 'datetime', 'city', 'branch', 'customer_type', 'gender',
        'product_line', 'unit_price', 'quantity', 'tax', 'total',
        'payment_method', 'cogs', 'gross_income', 'rating'
    ]
    df = df[expected_columns]

    # Проверка типов данных (пример для datetime)
    df['datetime'] = pd.to_datetime(df['datetime'])

    # Отладочный вывод
    print("Столбцы перед вставкой:", df.columns.tolist())
    print("Пример данных:", df.head(2).to_dict())

    # Вставка с явным указанием колонок
    client = Client(**kwargs['clickhouse_params'])
    try:
        client.execute(
            """
            INSERT INTO processed_data (
                id, datetime, city, branch, customer_type, gender,
                product_line, unit_price, quantity, tax, total,
                payment_method, cogs, gross_income, rating
            ) VALUES
            """
            + df.to_dict('records')
        )
        client.execute("TRUNCATE TABLE unprocessed_data")
    except Exception as e:
        print(f"Ошибка при вставке: {str(e)}")
        raise
    finally:
        client.disconnect()

with DAG(
    'data_processing_dag',
    default_args=default_args,
    schedule="@daily",
    catchup=False
) as dag:

    start = EmptyOperator(task_id='start')

    create_tables = PythonOperator(
        task_id='create_postgres_tables',

```

```

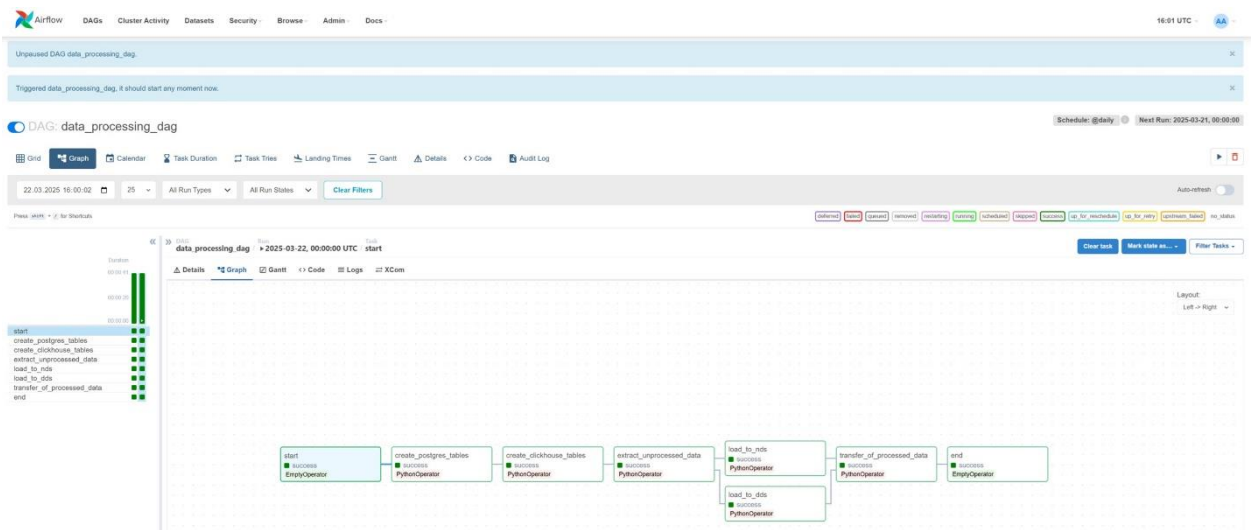
python_callable=create_postgres_tables,
op_kwargs={'postgres_params': postgres_params}
)

create_clickhouse = PythonOperator(
    task_id='create_clickhouse_tables',
    python_callable=create_clickhouse_tables,
    op_kwargs={'clickhouse_params': clickhouse_params}
)
extract_data = PythonOperator(
    task_id='extract_unprocessed_data',
    python_callable=extract_unprocessed_data,
    op_kwargs={'clickhouse_params': clickhouse_params}
)
load_nds = PythonOperator(
    task_id='load_to_nds',
    python_callable=load_to_nds,
    op_kwargs={
        'postgres_params': postgres_params,
        'column_map': column_map
    }
)
load_dds = PythonOperator(
    task_id='load_to_dds',
    python_callable=load_to_dds,
    op_kwargs={
        'postgres_params': postgres_params,
        'column_map': column_map
    }
)
transfer_data = PythonOperator(
    task_id='transfer_of_processed_data',
    python_callable=transfer_of_processed_data,
    op_kwargs={'clickhouse_params': clickhouse_params}
)
end = EmptyOperator(task_id='end')

start >> create_tables >> create_clickhouse >> extract_data
extract_data >> [load_nds, load_dds] >> transfer_data >> end

```

Граф задач:



Проверяем загрузку данных:

```

C:\Users\max\AppData\Local\docker ps
CONTAINER ID   IMAGE                                COMMAND                                  NAMES      CREATED          STATUS
bd74a960e643  apache/airflow:2.8.1               /usr/bin/dumb-init -s                 32 minutes ago Up 32 minutes (healthy) 8880/tcp    diplom-airflow-scheduler-1
cd394f6e7636  apache/airflow:2.8.1               /usr/bin/dumb-init -s                 32 minutes ago Up 32 minutes (healthy) 8880/tcp    diplom-airflow-triggerer-1
ed9948f741cc  apache/airflow:2.8.1               /usr/bin/dumb-init -s                 32 minutes ago Up 32 minutes (healthy) 8880/tcp    diplom-airflow-worker-1
2f1dd8b7ca    clickhouse/clickhouse-server:23.8-alpine /entrypoint.sh                         32 minutes ago Up 32 minutes (healthy) 0.0.0.0:8080-8080/tcp, 0.0.0.0:9000-9000/tcp, 0.0.0.0:9009-9009/tcp diplom-clickhouse-1
4d8f32cc8a3a  postgres:13                        /docker-entrypoint.sh                 32 minutes ago Up 32 minutes (healthy) 0.0.0.0:5432-5432/tcp diplom-postgres-1
18852a2adedc  redis:latest                        /docker-entrypoint.sh                 32 minutes ago Up 32 minutes (healthy) 6379/tcp    diplom-redis-1

PS C:\Users\max\AppData\Local\docker exec -it 2f1dd8b7ca clickhouse-client --database=default
Clickhouse client version 23.8.16 (official build)
Connecting to database default at localhost:9000 as user default.
Connected to Clickhouse server version 23.8.16 revision 50465.

Warnings:
* Linux transparent hugepages are set to "always". Check /sys/kernel/mm/transparent_hugepage/enabled

2f1dd8b7ca : SHOW TABLES FROM default;

SHOW TABLES FROM default
+-----+
| name |
+-----+
| processed_data |
| sales_data |
| unprocessed_data |
+-----+

2f1dd8b7ca : select * from sales_data

SELECT *
FROM sales_data
Query id: 60f62995-1cd1-43ab-8824-773bf5767579

+-----+
| name |
+-----+
| processed_data |
| sales_data |
| unprocessed_data |
+-----+

2f1dd8b7ca : select * from sales_data

SELECT *
FROM sales_data
Query id: 3ba81aa-2b96-efdc-b0c8-a0b7b5d27177

+-----+
| Invoice ID | Branch | City | Customer type | Gender | Product line | Unit price | Quantity | Tax $ | Total | Date | Time | Payment | Cogs | gross_margin_pct | gross_income | Rating |
+-----+
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 765-26-6951 | A | Yangan | Normal | Male | Sports and travel | 72.61 | 6 | 21.78 | 457.44 | 2019-01-01 | 2025-03-16 10:39:00 | Credit card | 435.66 | 4.7619047 | 21.78 | 6.9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 766-04-1077 | B | Mandala | Member | Female | Food and beverages | 84.63 | 10 | 42.31 | 888.61 | 2019-01-01 | 2025-03-16 11:36:00 | Credit card | 846.3 | 4.7619047 | 42.31 | 9 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 771-77-8740 | C | Nayuyitaw | Member | Female | Sports and travel | 29.22 | 6 | 8.76 | 184.08 | 2019-01-01 | 2025-03-16 11:40:00 | Ewallet | 175.32 | 4.7619047 | 8.76 | 5 |
| 133-14-7229 | C | Nayuyitaw | Normal | Male | Health and beauty | 62.87 | 2 | 6.28 | 132.02 | 2019-01-01 | 2025-03-16 11:43:00 | Cash | 125.74 | 4.7619047 | 6.28 | 5 |
| 133-14-7229 | C | Nayuyitaw | Normal | Male | Health and beauty | 62.87 | 2 | 6.28 | 132.02 | 2019-01-01 | 2025-03-16 11:43:00 | Cash | 125.74 | 4.7619047 | 6.28 | 5 |
| 133-14-7229 | C | Nayuyitaw | Normal | Male | Health and beauty | 62.87 | 2 | 6.28 | 132.02 | 2019-01-01 | 2025-03-16 11:43:00 | Cash | 125.74 | 4.7619047 | 6.28 | 5 |
| 133-14-7229 | C | Nayuyitaw | Normal | Male | Health and beauty | 62.87 | 2 | 6.28 | 132.02 | 2019-01-01 | 2025-03-16 11:43:00 | Cash | 125.74 | 4.7619047 | 6.28 | 5 |
| 133-14-7229 | C | Nayuyitaw | Normal | Male | Health and beauty | 62.87 | 2 | 6.28 | 132.02 | 2019-01-01 | 2025-03-16 11:43:00 | Cash | 125.74 | 4.7619047 | 6.28 | 5 |
| 651-88-7328 | A | Yangan | Normal | Female | Fashion accessories | 65.74 | 9 | 29.58 | 621.24 | 2019-01-01 | 2025-03-16 13:55:00 | Cash | 591.66 | 4.7619047 | 29.58 | 7.7 |
| 651-88-7328 | A | Yangan | Normal | Female | Fashion accessories | 65.74 | 9 | 29.58 | 621.24 | 2019-01-01 | 2025-03-16 13:55:00 | Cash | 591.66 | 4.7619047 | 29.58 | 7.7 |
| 651-88-7328 | A | Yangan | Normal | Female | Fashion accessories | 65.74 | 9 | 29.58 | 621.24 | 2019-01-01 | 2025-03-16 13:55:00 | Cash | 591.66 | 4.7619047 | 29.58 | 7.7 |
| 651-88-7328 | A | Yangan | Normal | Female | Fashion accessories | 65.74 | 9 | 29.58 | 621.24 | 2019-01-01 | 2025-03-16 13:55:00 | Cash | 591.66 | 4.7619047 | 29.58 | 7.7 |
| 651-88-7328 | A | Yangan | Normal | Female | Fashion accessories | 65.74 | 9 | 29.58 | 621.24 | 2019-01-01 | 2025-03-16 13:55:00 | Cash | 59
```

Данные загружены

5. Формирование набора метрик и визуализация данных

При выборе метрик для построения дашборда основной целью было повышение эффективности мониторинга ключевых аспектов бизнеса и обеспечение более глубокого понимания его производительности. Подобранные показатели охватывают широкий спектр направлений: от финансовых результатов и эффективности продаж до взаимодействия с клиентами и качества обслуживания. Такой подход позволяет принимать более взвешенные и обоснованные решения, способствуя повышению общей эффективности компании и достижению стратегических целей.

Использование Power BI обусловлено тем что Tableau ушел из России и очень непросто подключить к нему данные.

Выбранные метрики:

1. Общая выручка компании:

- *Цель:* Оценить общую финансовую производительность компании.
- *Инсайты:* Позволяет оценить общий объем денежных средств, поступающих в компанию.

2. Налог с выручки:

- *Цель:* Обеспечить контроль за налоговой обязанностью компании и ее финансовой устойчивостью.

- *Инсайты:* Позволяет контролировать налоговую обязанность компании и оценивать долю выручки, уходящую на налоги.
3. **Общая прибыль:**
- *Цель:* Измерить общую прибыль и оценить финансовую эффективность бизнеса.
 - *Инсайты:* Позволяет оценить финансовую эффективность бизнеса и его прибыльность.
4. **Количество выполненных продаж:**
- *Цель:* Оценить активность продаж и популярность товаров или услуг.
 - *Инсайты:* Помогает оценить активность продаж.
5. **Пол покупателей:**
- *Цель:* Анализ гендерного распределения клиентов.
 - *Инсайты:* Позволяет выявить предпочтения и поведение покупателей в зависимости от пола.
6. **Способ оплаты покупки:**
- *Цель:* Изучение предпочтений клиентов по методам оплаты.
 - *Инсайты:* Помогает определить популярные методы оплаты и оптимизировать процессы оплаты
7. **Тип клиента (наличие клубной карты):**
- *Цель:* Анализ клиентской базы и их участие в программе лояльности.
 - *Инсайты:* Позволяет оценить эффективность программы лояльности и влияние на поведение клиентов.
8. **Прибыль по категориям товаров:**
- *Цель:* Оценка прибыльности различных категорий товаров.
 - *Инсайты:* Позволяет выделить наиболее прибыльные категории товаров и принимать решения по ассортименту.
9. **Выручка по филиалам:**
- *Цель:* Измерение доли выручки, генерируемой каждым филиалом.
 - *Инсайты:* Позволяет выявить ключевые источники выручки и фокусироваться на успешных филиалах.
10. **Средний рейтинг от клиентов по категориям товаров:**
- *Цель:* Оценка удовлетворенности клиентов различными категориями товаров.
 - *Инсайты:* Помогает выделить популярные категории среди клиентов и принимать решения по улучшению ассортимента.
11. **Количество продаж по филиалам:**
- *Цель:* Измерение активности клиентов в различных филиалах.
 - *Инсайты:* Помогает выявить факторы, влияющие на количество продаж в каждом филиале.
12. **Средний рейтинг филиалов:**
- *Цель:* Оценка удовлетворенности клиентов работой различных филиалов.
 - *Инсайты:* Позволяет выявить факторы, влияющие на удовлетворенность клиентов, и принимать меры по улучшению сервиса.
13. **Количество проданных товаров по категориям:**
- *Цель:* Измерение популярности конкретных категорий товаров.

- **Инсайты:** Помогает анализировать спрос на товары и оптимизировать их предложение.

Подключение источника данных в Power BI:

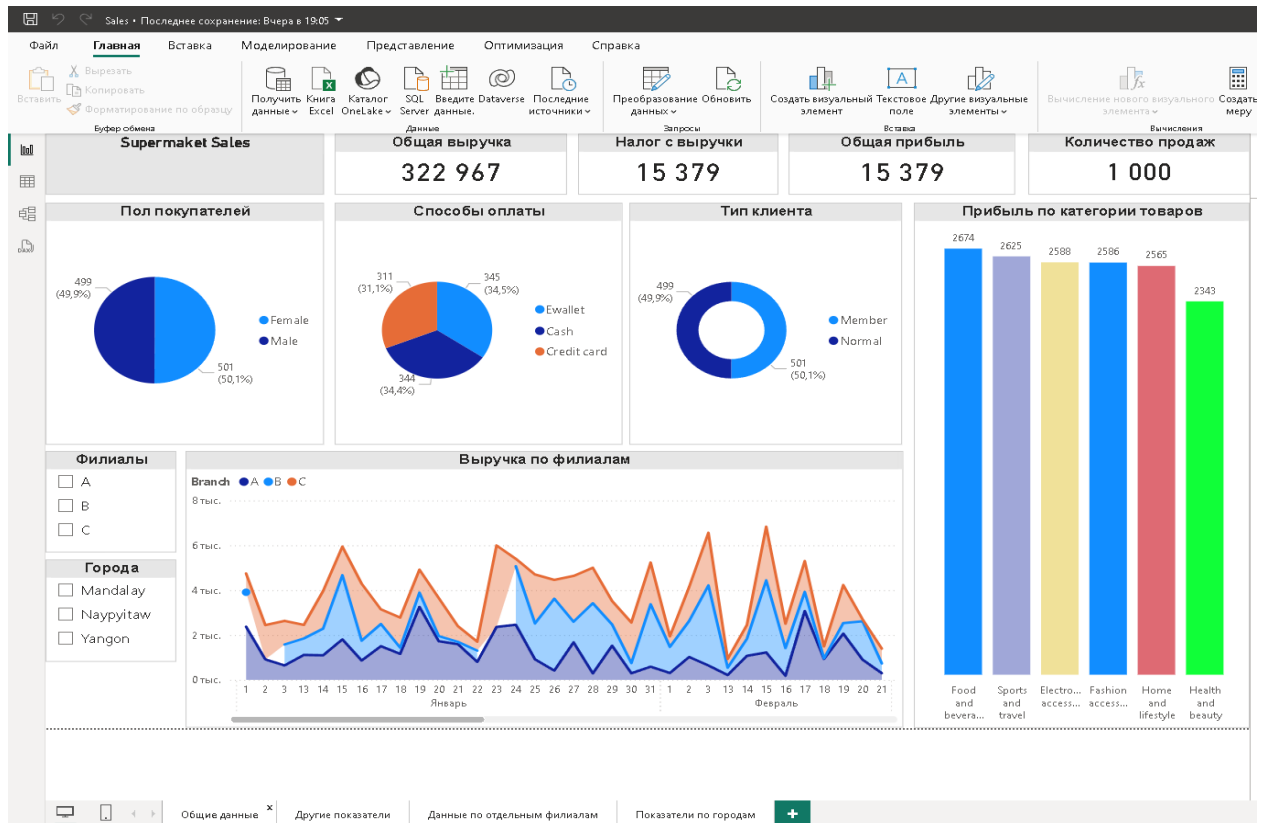
Подключение к Power BI тоже вызвало трудности в связи с тем, что мой компьютер просто не тянет Clickhouse и Tableau.

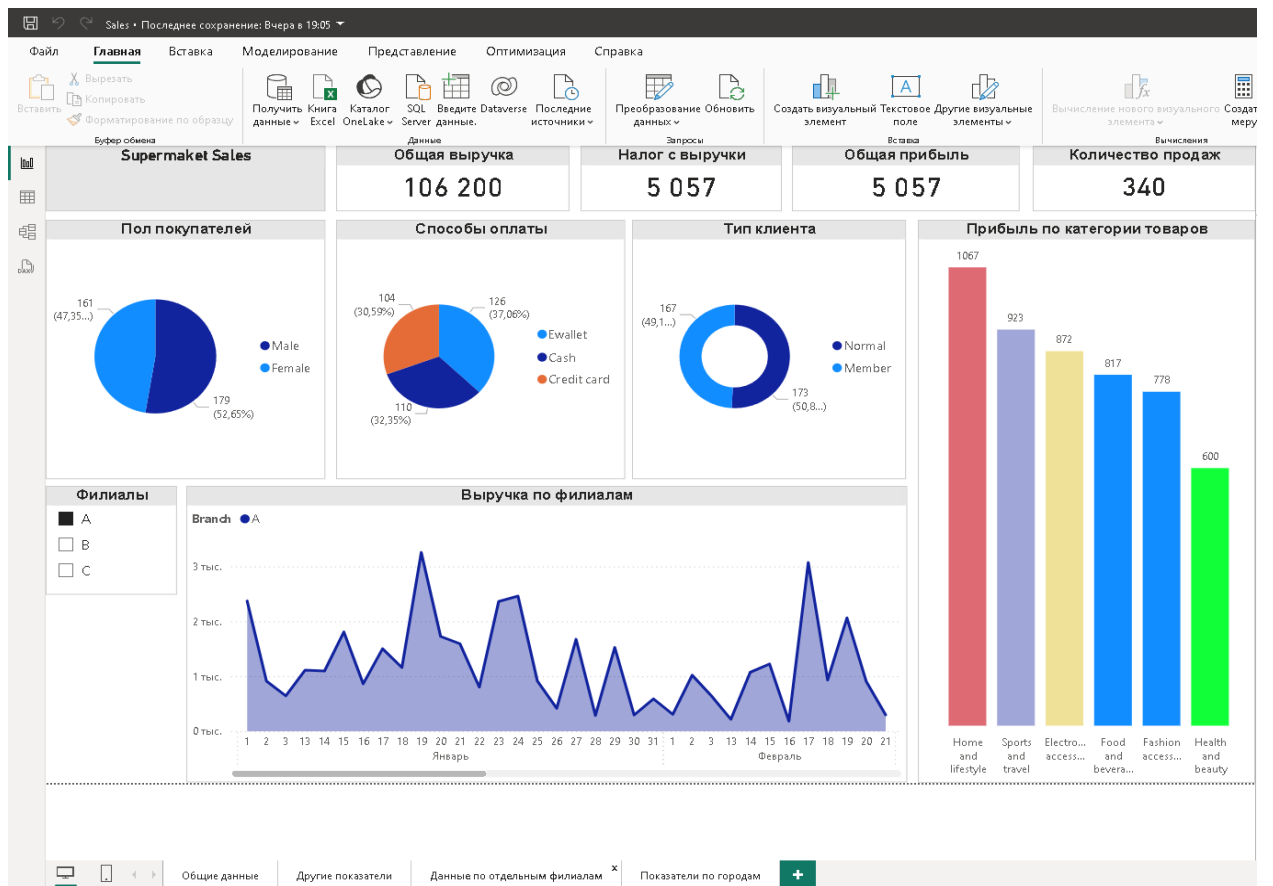
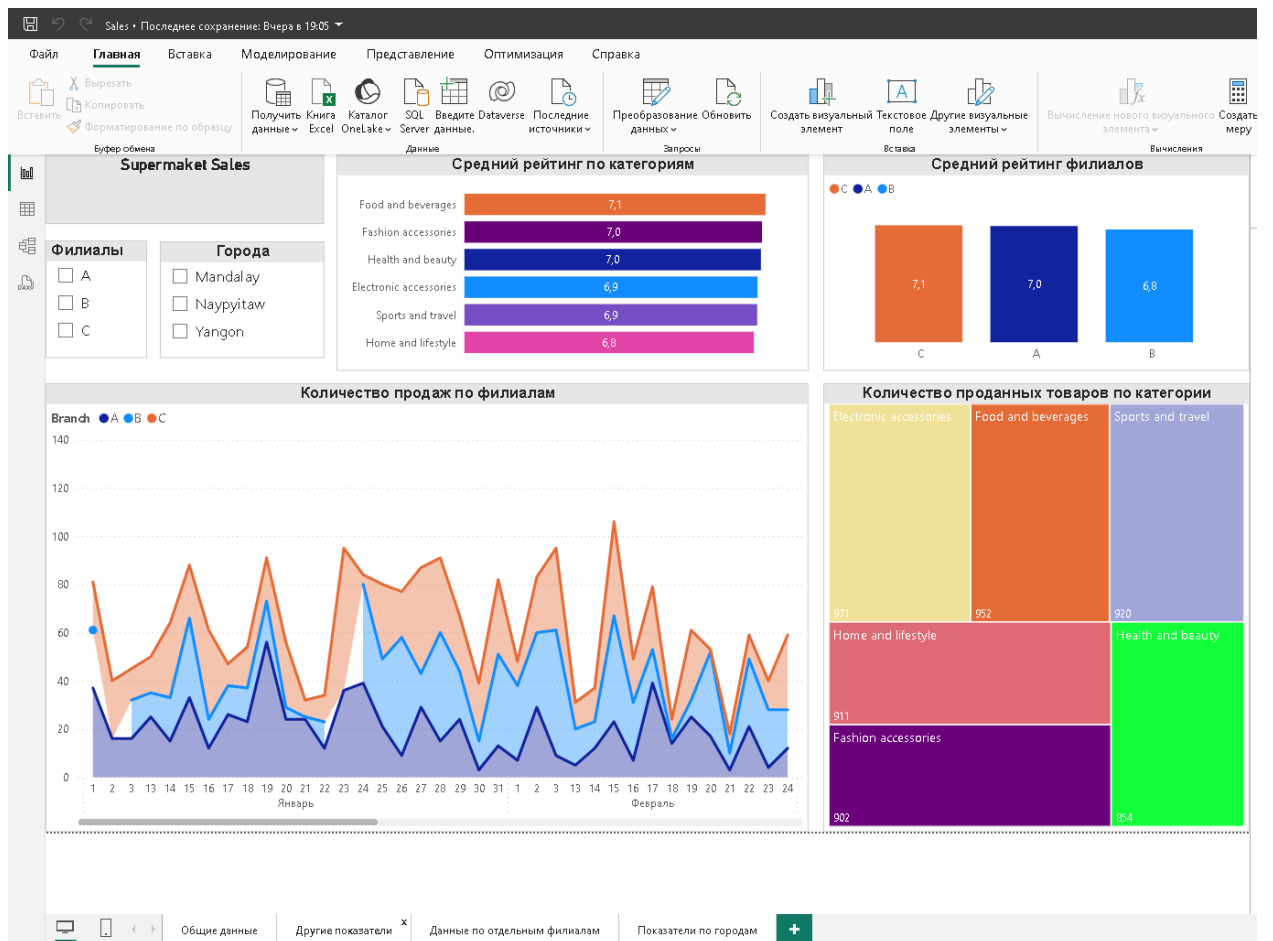
```
PS C:\Users\max_f\архив\Диплом> docker exec 2df1dd8b87ca bash -c "clickhouse-client --query 'SELECT * FROM sales_data FORMAT CSV' > /var/lib/clickhouse/sales_data.csv"
PS C:\Users\max_f\архив\Диплом> docker cp 2df1dd8b87ca:/var/lib/clickhouse/sales_data.csv ./sales_data.csv
Successfully copied 1.27MB to C:\Users\max_f\архив\Диплом\sales_data.csv
```

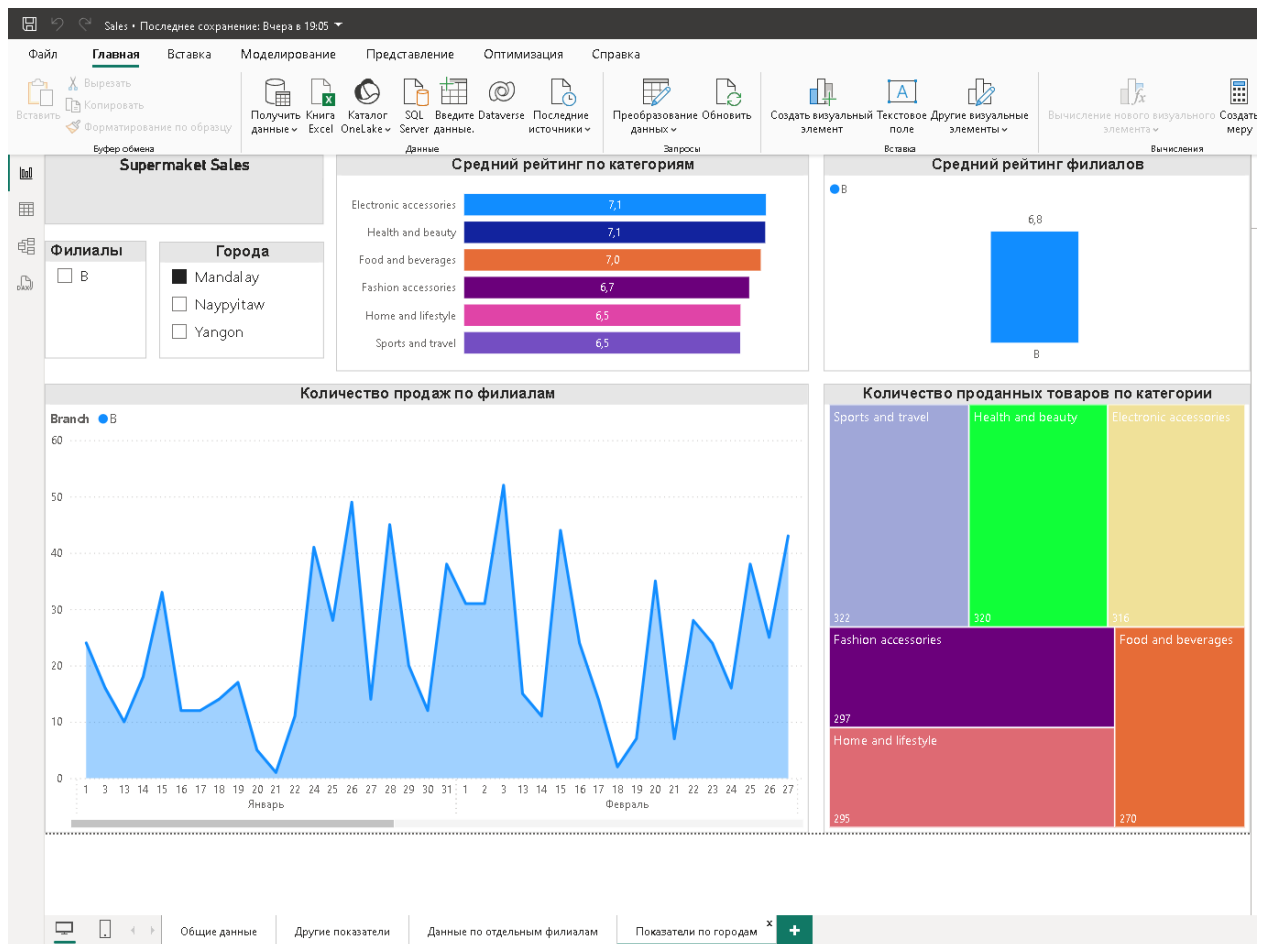
Containers / diplom-clickhouse-1			
diplom-clickhouse-1			
8123.8123 9000:9000 Show all ports (3)			
Logs Inspect Bind mounts Exec Files Stats			
Name	Note	Size	Last modified
run			9 months ago
sbin			8 months ago
srv			8 months ago
sys			2 hours ago
tmp			7 months ago
clickhouse-client-23.8.16-amd64.tgz.sha512		169 Bytes	7 months ago
clickhouse-common-static-23.8.16-amd64.tgz.sha512		176 Bytes	7 months ago
clickhouse-server-23.8.16-amd64.tgz.sha512		169 Bytes	7 months ago
usr			8 months ago
var			8 months ago
cache			8 months ago
empty			8 months ago
lib			7 months ago
clickhouse			1 minute ago
access			15 days ago
data			15 days ago
dictionaries_lib			15 days ago
flags			15 days ago
format_schemas			15 days ago
metadata			15 days ago
metadata_dropped			15 days ago
named_collections			15 days ago
preprocessed_configs			15 days ago
sales_data.csv		1.2 MB	1 minute ago
status		55 Bytes	2 hours ago
store			13 days ago
tmp			2 hours ago
user_defined			15 days ago
user_files			15 days ago
user_scripts			15 days ago
uuid		36 Bytes	15 days ago
misc			8 months ago

и

Итоговый дашборд:







6. Выводы

В ходе выполнения дипломной работы было выполнено:

1. Обработка и анализ данных:

- Проведен анализ предоставленных данных с целью выявления ошибок и пропусков.
- Выполнена очистка и подготовка данных для последующего использования.

2. Нормализованная схема данных (NDS):

- Разработана структурированная нормализованная схема данных, обеспечивающая эффективное хранение и управление информацией.

3. Таблицы фактов и измерений (DDS):

- Созданы таблицы фактов и измерений, сформировав структуру схемы «звезда», что обеспечивает удобный доступ к ключевым бизнес-показателям.

4. ETL-процессы:

- Разработаны ETL-процессы для автоматизированной загрузки данных в NDS и DDS.
- Обеспечен эффективный поток данных с минимизацией временных затрат на их обработку.

5. Дашборды в Power BI:

- Построили дашборды в Power BI, визуализируя ключевые метрики, такие как общая выручка, прибыль, количество продаж, рейтинги и другие.

Решенные бизнес-задачи:

1. Управление финансами:

- Обеспечен мониторинг общей выручки компании и её филиалов.
- Произведена оценка общей прибыли и расчёт налога с выручки.

2. Анализ продаж:

- Определена эффективность продаж в разрезе филиалов.
- Проанализирована популярность различных продуктовых линеек.

3. Понимание клиентского поведения:

- Изучены предпочтения клиентов по полу и способам оплаты.
- Выявлено влияние наличия карты клиента на поведение покупателей.

4. Оптимизация ассортимента и рейтингов:

- Выполнено ранжирование продуктовых линеек по прибыльности и объёму продаж.
- Оценен средний рейтинг товаров по категориям и филиалам.

Итог работы: Данная дипломная работа позволила закрепить полученные теоретические знания и практические навыки, приобретённые в процессе прохождения курса.

Разработанные решения обеспечивают бизнесу возможность:

- Эффективно управлять финансами.
- Анализировать ключевые показатели продаж.
- Понимать поведение клиентов и их предпочтения.
- Принимать обоснованные решения для оптимизации бизнес-процессов.
- Повышать качество стратегического планирования и развития компании.

Полученные результаты дают компании мощный инструмент для улучшения операционной эффективности, повышения удовлетворенности клиентов и достижения долгосрочных целей.