Титульник\*\*

БАЗА ДАННЫХ КОМПАНИИ ПО ОПТОВОЙ ПРОДАЖЕ ТКАНЕЙ

Город 2023

СОДЕРЖАНИЕ

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3](#_Toc138103133)

[1.1 Описание предметной области 3](#_Toc138103134)

[1.2 Цель создания БД 3](#_Toc138103135)

[1.3 Возможные пользователи и сценарии взаимодействия пользователей и базы данных: 3](#_Toc138103136)

[1.4 Участвующие сущности логической модели базы данных «Компания по оптовой продаже тканей»: 4](#_Toc138103137)

[1.5 Алгоритмы обработки данных, используемые в сценариях: 6](#_Toc138103138)

[1.6 Планируемый перечень отчетных форм: 7](#_Toc138103139)

[1.7 Архитектура программного продукта может включать: 7](#_Toc138103140)

[2 ПРОЕКТ БАЗЫ ДАННЫХ 8](#_Toc138103141)

[2.1 Разработка схем 8](#_Toc138103142)

[2.2 Нормализация 9](#_Toc138103143)

[3 ЗАПОЛНЕНИЕ ДАННЫМИ 10](#_Toc138103144)

[3.1 Создание таблиц 10](#_Toc138103145)

[3.2 Добавление данных 11](#_Toc138103146)

[4 РАЗРАБОТКА 16](#_Toc138103147)

[4.1 Запросы 16](#_Toc138103148)

[4.2 Триггер 26](#_Toc138103149)

[4.3 Функции 27](#_Toc138103150)

[4.4 Представления 28](#_Toc138103151)

[4.5 Хранимые процедуры 30](#_Toc138103152)

[5 ОТЧЕТЫ 35](#_Toc138103153)

[5.1 ○ Основной отчёт Power BI – агрегированные данные со ссылками на детализацию 35](#_Toc138103154)

[5.2 ○ Детальный отчёт Power BI для drill down 35](#_Toc138103155)

[5.3 ○ Информационная панель в Excel с графиком и "срезами" 36](#_Toc138103156)

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Описание предметной области

Предметной областью, к которой относится компания по оптовой продаже тканей, является текстильная промышленность. Текстильная промышленность – одна из старейших отраслей промышленности, которая возникла еще в древности, когда человек начал производить ткани из различных материалов, таких как лен, шелк, хлопок и др. С тех пор текстильная промышленность прошла долгий путь развития и стала одной из крупнейших и наиболее важных отраслей мировой экономики.

Компания по оптовой продаже тканей занимается поставками тканей оптом на рынок. Она предлагает широкий ассортимент продукции, включая ткани различного состава и назначения, а также различные виды отделки и фактуры. Клиентами компании­­ являются различные организации и предприятия, которые занимаются производством текстильной продукции.

## Цель создания БД

Целью создания базы данных для магазина по оптовой продаже тканей является эффективное управление товарами, заказами, поставщиками и клиентами. База данных помогает автоматизировать процессы учета, заказа и отслеживания тканей, а также обеспечивает быстрый доступ к информации, позволяющий принимать взвешенные решения в области закупок и продаж.

## Возможные пользователи и сценарии взаимодействия пользователей и базы данных:

* Администраторы магазина: управляют базой данных, добавляют, изменяют и удаляют информацию о товарах, поставщиках, заказах и клиентах.
* Менеджеры по продажам: используют базу данных для обработки заказов, управления складскими запасами, составления отчетов о продажах и клиентах.
* Покупатели: могут обращаться к базе данных для получения информации о доступных товарах, ценах и сделанных заказах.

Возможные сценарии взаимодействия пользователей и базы данных:

* Администратор добавляет новые товары в базу данных, вносит изменения в информацию о поставщиках, настраивает параметры системы и обеспечивает ее безопасность.
* Менеджер по продажам создает заказы от имени клиентов, проверяет наличие товаров на складе, обновляет информацию о статусе заказов и генерирует отчеты о продажах.
* Покупатель ищет нужные ткани, просматривает их описания и цены, оформляет заказ, отслеживает статус доставки и просматривает свою историю заказов.

## Участвующие сущности логической модели базы данных «Компания по оптовой продаже тканей»:

* Пользователь: Идентификатор, email, пароль, логин.
* Корзина: Идентификатор, идентификатор пользователя, идентификатор заказа, артикул, цена, количество, суммарная стоимость.
* Заказ: Идентификатор, идентификатор пользователя, дата, цена, статус, категория.
* Продукт: Артикул, идентификатор категории, идентификатор категории, название, описание, сезон, цвет, цена, изображение.
* Цвет: Идентификатор, название.
* Категория: Идентификатор, название, описание, изображение.
* Поставщик: Идентификатор, идентификатор страны, название.
* Страна: Идентификатор, название.

Для реализации базы данных использована PostgreSQL - свободная реляционная система управления базами данных.

PostgreSQL очень быстр, надежен и легок в использовании, а также является реляционной СУБД. Реляционная база данных сохраняет данные в отдельных таблицах. Это добавляет быстродействие и гибкость. Таблицы связаны определенными отношениями, делающими возможным объединить данные из нескольких таблиц в одном запросе.

«*User*» - содержит информацию о пользователе, такую как электронная почта, пароль и логин. Описание таблицы представлено в таблице 1.

Таблица 1 – Структура таблицы «*User*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор пользователя |
| email | varchar | key | Электронная почта пользователя |
| password | varchar | - | Пароль пользователя |
| login | varchar | - | Логин пользователя |

«*Cart*» - содержит информацию о товарах в корзине клиентов, такую как цена, количество, суммарную стоимость, а также внешние ключи на таблицы пользователи, заказы и товары. Описание таблицы представлено в таблице 2.

Таблица 2 – Структура таблицы «*Cart*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор товара в корзине |
| price | numeric | - | Цена товара |
| count | int | - | Количество товара |
| summary | numeric | - | Общая стоимость |
| article | int | foreign | Уникальный идентификатор товара |
| user\_id | int | foreign | Уникальный идентификатор пользователя |
| order\_id | int | foreign | Уникальный идентификатор заказа |

«*Order*» *–* содержит информацию о заказах, такую как дата и время совершения заказа, а также внешний ключ на таблицу пользователи. Описание таблицы представлено в таблице 3.

Таблица 3 – Структура таблицы «*Order*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор заказа |
| order\_date | timestamp | - | Дата и время заказа |
| user\_id | int | foreign | Уникальный идентификатор пользователя |

«*Category*» - содержит информацию о категориях товаров такую, как название категории, описание и изображение. Описание таблицы представлено в таблице 4.

Таблица 4 – Структура таблицы «*Category*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор категории |
| category | varchar | - | Название категории |
| description | text | - | Описание категории |
| image | bytea | - | Изображение категории |

«Product» *–* содержит информацию о товарах такую, название, описание, сезон, цвет, цена и изображение, а также внешние ключи на таблицы категорий и поставщиков. Описание таблицы представлено в таблице 5.

Таблица 5 – Структура таблицы «Product»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| article | int | primary | Уникальный идентификатор товара |
| name | varchar | - | Название товара |
| description | text | - | Описание товара |
| season | varchar | - | Сезон товара |
| price | numeric | - | Цена товара |
| image | bytea | - | Изображение товара |
| color | int | foreign | Уникальный идентификатор цвета |
| category | int | foreign | Уникальный идентификатор категории |
| supplier | int | foreign | Уникальный идентификатор поставщика |

«*Supplier*» *–* содержит информацию о поставщиков, такую как название поставщика, а также внешний ключ на таблицу страны. Описание таблицы представлено в таблице 6.

Таблица 6 – Структура таблицы «*Supplier*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор поставщика |
| supplier | varchar | - | Название поставщика |
| country\_id | int | foreign | Уникальный идентификатор страны |

«*Country*» *–* содержит информацию о странах, такую как название страны. Описание таблицы представлено в таблице 7.

Таблица 7 – Структура таблицы «*Country*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор заказа |
| country | varchar | - | Название страны |

«*Color*» *–* содержит информацию о цветах, такую как название страны. Описание таблицы представлено в таблице 8.

Таблица 8 – Структура таблицы «*Color*»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Ключ | Описание |
| id | int | primary | Уникальный идентификатор заказа |
| country | varchar | - | Название страны |

## Алгоритмы обработки данных, используемые в сценариях:

* Алгоритм обработки заказов: создание новых заказов, связь с соответствующими клиентами и товарами, обновление статуса заказов.
* Алгоритм генерации отчетов: сбор и анализ данных о продажах, клиентах и товарах для создания отчетов о прибыли, статистике продаж и другой полезной информации.

## Планируемый перечень отчетных форм:

* Отчет о продажах по периоду: содержит информацию о проданных товарах, суммарной стоимости, клиентах и дате продажи.
* Отчет о клиентах: включает данные о клиентах, их контактной информации, истории заказов и предпочтениях.

## Архитектура программного продукта может включать:

* Базу данных для хранения информации о товарах, поставщиках, заказах, клиентах и других сущностях.
* Пользовательский интерфейс для администраторов, менеджеров и покупателей, обеспечивающий удобный доступ к функциональности базы данных.
* Модули обработки данных, реализующие алгоритмы управления запасами, обработки заказов и генерации отчетов.
* Систему безопасности для защиты конфиденциальности данных и предотвращения несанкционированного доступа.
* Интеграцию с другими системами, например, системой учета, электронными платежными системами и системой доставки.

# ПРОЕКТ БАЗЫ ДАННЫХ

## Разработка схем

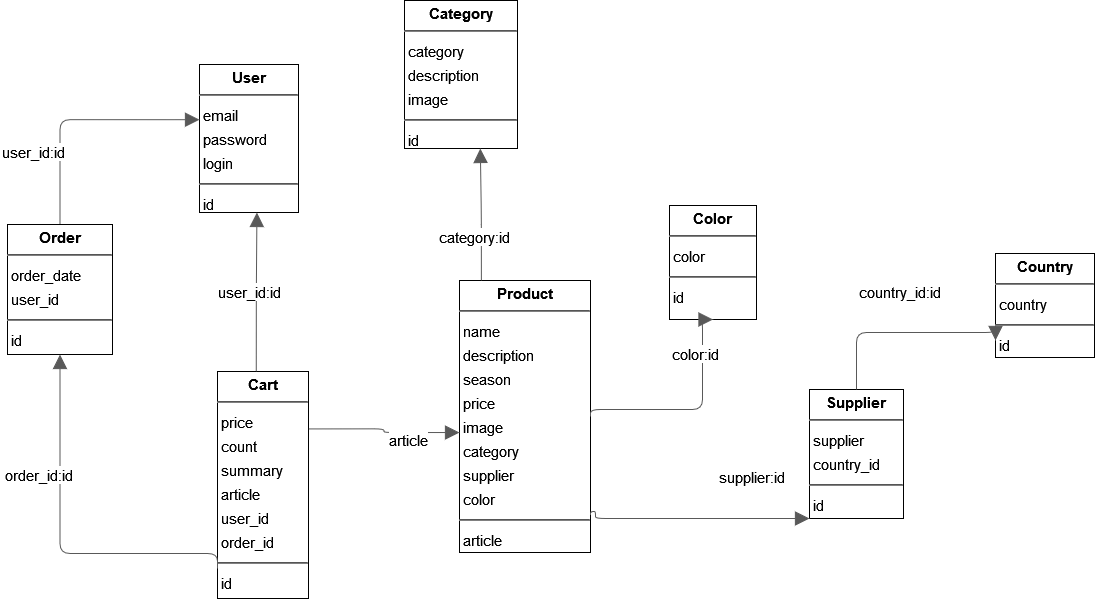


Рисунок 2.1 – Логическая схема базы данных



Рисунок 2.2 – Диаграмма «сущность-связь»

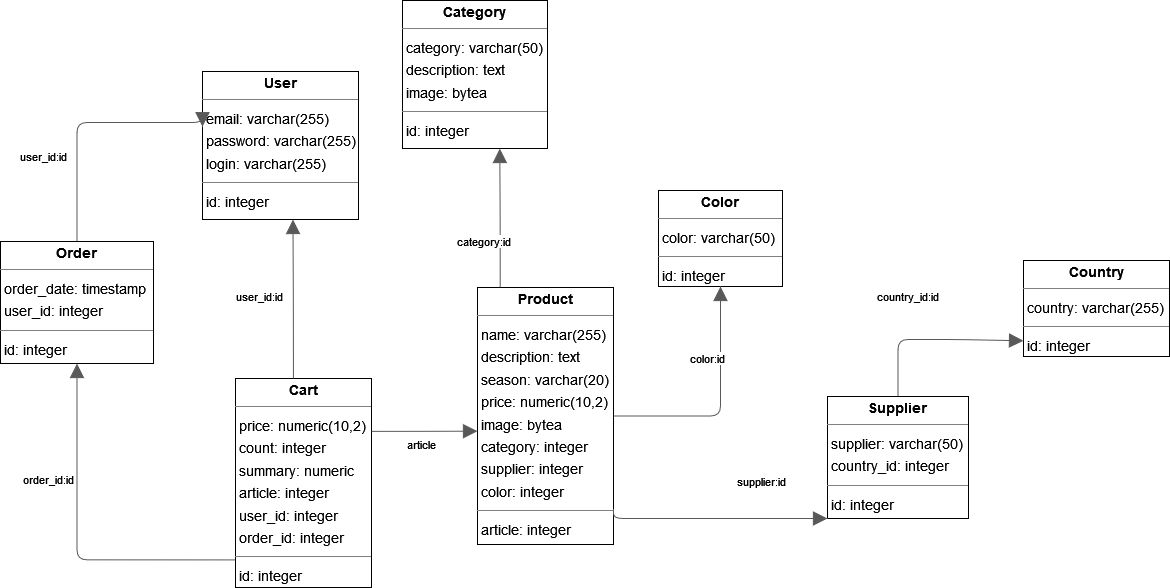


Рисунок 2.3 – Схема реляционной базы данных

## Нормализация

Разработанная нами база данных находится в третьей нормальной форме, так как:

* Она находится в первой нормальной форме, то есть атрибуты ее таблиц являются атомарными, отсутствуют повторяющиеся группы.
* Она находится во второй нормальной форме, то есть она находится в первой нормальной форме и для каждой таблицы определен первичный ключ и все атрибуты таблицы зависят от него.
* В таблицах отсутствуют транзитивные зависимости.

# ЗАПОЛНЕНИЕ ДАННЫМИ

## Создание таблиц

-- Создание таблицы "User"

CREATE TABLE "User" (

id SERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE,

password VARCHAR(255),

login VARCHAR(255)

);

-- Создание таблицы "Cart"

CREATE TABLE "Cart" (

id SERIAL PRIMARY KEY,

price NUMERIC(10, 2),

count INT,

summary NUMERIC,

article INT,

user\_id INT,

order\_id INT,

FOREIGN KEY (article) REFERENCES "Product"(article),

FOREIGN KEY (user\_id) REFERENCES "User"(id),

FOREIGN KEY (order\_id) REFERENCES "Order"(id)

);

-- Создание таблицы "Order"

CREATE TABLE "Order" (

id SERIAL PRIMARY KEY,

order\_date TIMESTAMP,

user\_id INT,

FOREIGN KEY (user\_id) REFERENCES "User"(id)

);

-- Создание таблицы "Category"

CREATE TABLE "Category" (

id SERIAL PRIMARY KEY,

category VARCHAR(50),

description TEXT,

image BYTEA

);

-- Создание таблицы "Color"

CREATE TABLE "Color" (

id SERIAL PRIMARY KEY,

color VARCHAR(50)

)

-- Создание таблицы "Product"

CREATE TABLE "Product" (

article INT PRIMARY KEY,

name VARCHAR(255),

description TEXT,

season VARCHAR(20),

price NUMERIC(10, 2),

image BYTEA,

category INT,

supplier INT,

FOREIGN KEY (color) REFERENCES "Color"(id),

FOREIGN KEY (category) REFERENCES "Category"(id),

FOREIGN KEY (supplier) REFERENCES "Supplier"(id)

);

-- Создание таблицы "Supplier"

CREATE TABLE "Supplier" (

id SERIAL PRIMARY KEY,

supplier VARCHAR(50),

country\_id INT,

FOREIGN KEY (country\_id) REFERENCES "Country"(id)

);

-- Создание таблицы "Country"

CREATE TABLE "Country" (

id SERIAL PRIMARY KEY,

country VARCHAR(20)

);

## Добавление данных

Для заполнения таблицы реалистичными данными, воспользуемся инструментами, которые помогут сгенерировать случайные значения для каждого столбца, напишем код Python. Вот пример использования библиотеки Faker для генерации таких данных в PostgreSQL и заполнения таблиц:

*from faker import Faker*

*import psycopg2*

*import random*

*fake = Faker()*

*# Подключение к базе данных PostgreSQL*

*conn = psycopg2.connect(*

*host="localhost",*

*database="fabrics",*

*user="postgres",*

*password="open38ai12"*

*)*

*# Создание курсора для выполнения SQL-запросов*

*cur = conn.cursor()*

*# Генерация и вставка 1000 значений в таблицу "User"*

*for \_ in range(1000):*

*email = fake.email()*

*password = fake.password()*

*login = fake.user\_name()*

*cur.execute("""*

*INSERT INTO "User" (email, password, login)*

*VALUES (%s, %s, %s)*

*ON CONFLICT (email) DO NOTHING*

*""", (email, password, login))*

*# Генерация и вставка 50 значений в таблицу "Order"*

*for \_ in range(50):*

*order\_date = fake.date\_time\_between(start\_date='-1y', end\_date='now')*

*user\_id = random.randint(1, 1000)*

*cur.execute("""*

*INSERT INTO "Order" (order\_date, user\_id)*

*VALUES (%s, %s)*

*""", (order\_date, user\_id))*

*# Генерация и вставка 50 значений в таблицу "Country"*

*for \_ in range(50):*

*country = fake.country()*

*cur.execute("""*

*INSERT INTO "Country" (country)*

*VALUES (%s)*

*""", (country,))*

*# Генерация и вставка 50 значений в таблицу "Supplier"*

*for \_ in range(50):*

*supplier = fake.company()*

*country\_id = random.randint(1, 50)*

*cur.execute("""*

*INSERT INTO "Supplier" (supplier, country\_id)*

*VALUES (%s, %s)*

*""", (supplier, country\_id))*

*# Генерация и вставка 50 значений в таблицу "Category"*

*for \_ in range(50):*

*category = fake.word()*

*description = fake.text()*

*image = bytes([random.randint(0, 255) for \_ in range(10)])*

*cur.execute("""*

*INSERT INTO "Category" (category, description, image)*

*VALUES (%s, %s, %s)*

*""", (category, description, image))*

*# Генерация и вставка 50 значений в таблицу "Product"*

*for article in range(1, 51):*

*name = fake.word()*

*description = fake.text()*

*season = fake.word()*

*color = fake.color\_name()*

*price = round(random.uniform(10.0, 100.0), 2)*

*image = bytes([random.randint(0, 255) for \_ in range(10)])*

*category = random.randint(1, 50)*

*supplier = random.randint(1, 50)*

*cur.execute("""*

*INSERT INTO "Product" (article, name, description, season, color, price, image, category, supplier)*

*VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)*

*""", (article, name, description, season, color, price, image, category, supplier))*

*# Генерация и вставка 50 значений в таблицу "Cart"*

*for \_ in range(50):*

*price = round(random.uniform(1.0, 100.0), 2)*

*count = random.randint(1, 10)*

*summary = price \* count*

*article = random.randint(1, 50)*

*user\_id = random.randint(1, 1000)*

*order\_id = random.randint(1, 50)*

*cur.execute("""*

*INSERT INTO "Cart" (price, count, summary, article, user\_id, order\_id)*

*VALUES (%s, %s, %s, %s, %s, %s)*

*""", (price, count, summary, article, user\_id, order\_id))*

*# Сохранение изменений в базе данных*

*conn.commit()*

*# Закрытие соединения*

*cur.close()*

*conn.close()*

Так, например, для таблицы “*User*” получили 1000 записей вида:

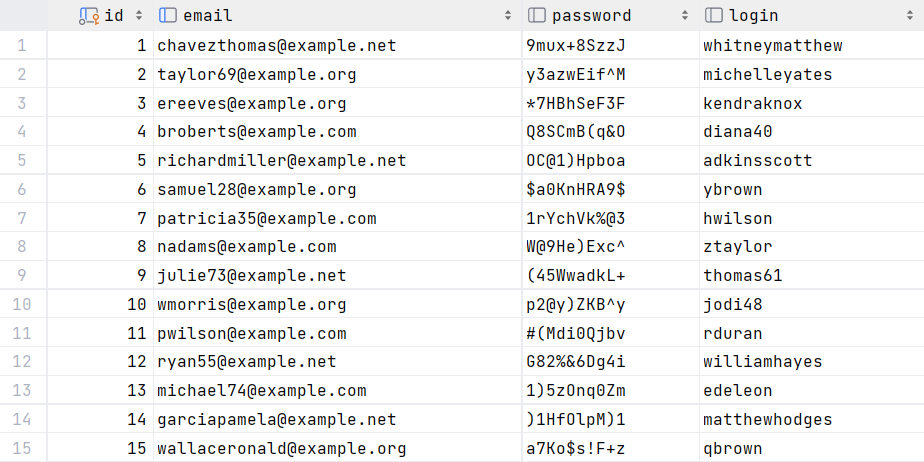


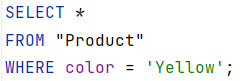
Рисунок 3.1 – Часть данных таблицы “User”

# РАЗРАБОТКА

## Запросы

**Простой запрос с условием и формулами в SELECT (2)**

1) Запрос для выборки товаров цвета " *Yellow* ":



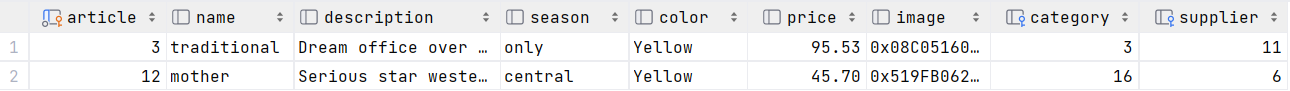
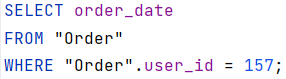


Рисунок 4.1 – Запрос №1

2) Запрос для выборки дат заказов пользователя с *id = 157*:



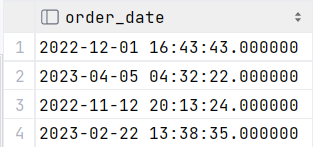
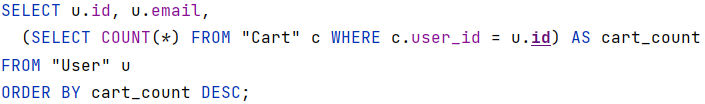


Рисунок 4.2 – Запрос №2

**Запрос с коррелированным подзапросом в SELECT (2)**

1) Запрос для выборки пользователей вместе с количеством товаров в их корзине с сортировкой по количеству товаров в корзине:



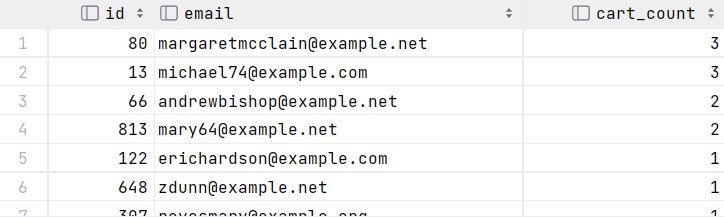
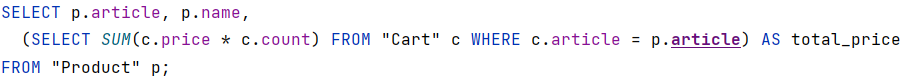


Рисунок 4.3 – Запрос №3

2) Запрос для выборки товаров вместе с суммарной стоимостью всех товаров в корзине:



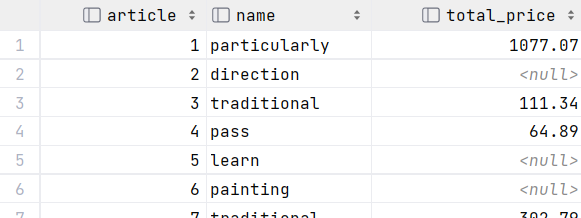


Рисунок 4.4 – Запрос №4

**Запрос с подзапросом в FROM (2)**

1) Запрос для выборки пользователей и количества товаров в их корзинах:

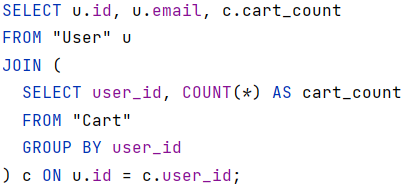
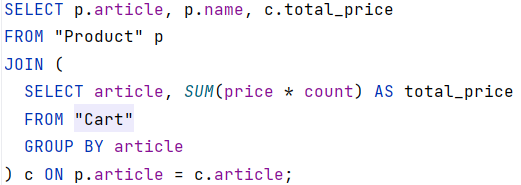




Рисунок 4.5 – Запрос №5

2) Запрос для выборки товаров вместе с суммарной стоимостью всех товаров в корзинах:



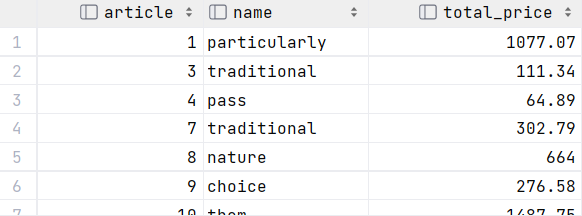
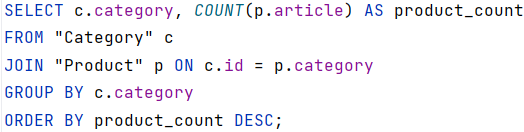


Рисунок 4.6 – Запрос №6

**Запрос с подзапросом в FROM, агрегированием, группировкой и сортировкой (2)**

1) Запрос для выборки категорий товаров вместе с количеством товаров в каждой категории, отсортированных по убыванию количества товаров:



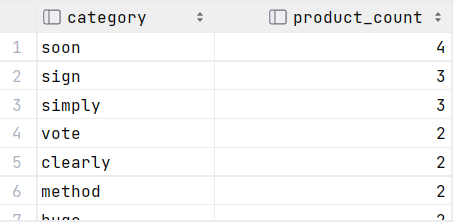


Рисунок 4.7 – Запрос №7

2) Запрос для выборки пользователей вместе с общей суммой их покупок, отсортированных по возрастанию суммы покупок:

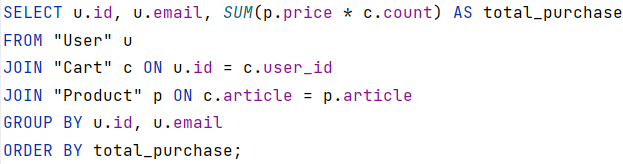
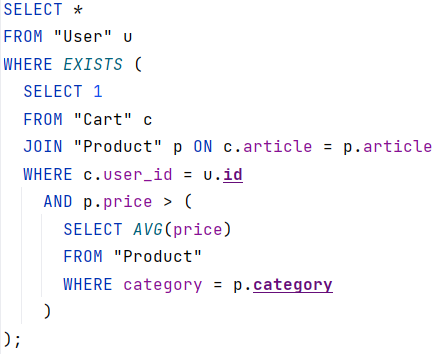




Рисунок 4.8 – Запрос №8

**Запрос с коррелированным подзапросом в WHERE (2)**

1) Запрос для выборки всех пользователей, у которых в корзине есть товары с ценой выше средней стоимости товаров в их категории:



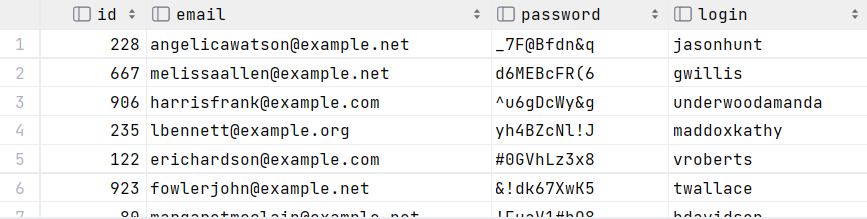


Рисунок 4.9 – Запрос №9

2) Запрос для выборки товаров, которые продаются у поставщиков из определенной страны (“*Iran*”):

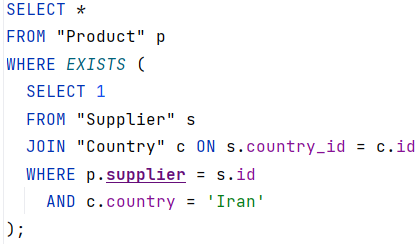
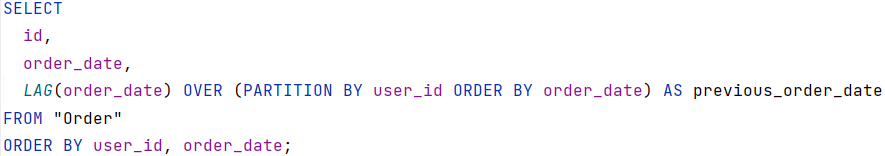




Рисунок 4.10 – Запрос №10

**Запрос, использующий оконную функцию LAG или LEAD для выполнения сравнения данных в разных периодах**

Сравнение дат заказов с предыдущими датами заказов для каждого пользователя:



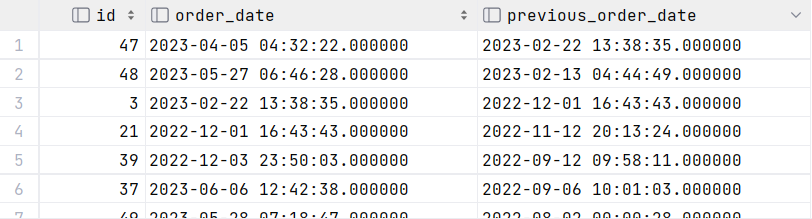
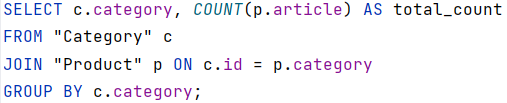


Рисунок 4.11 – Запрос №11

В этом запросе мы используем оконную функцию LAG для получения значения предыдущей даты заказа для каждого пользователя. Оконная функция LAG принимает аргументы: поле, которое хотим получить (order\_date), и параметры PARTITION BY и ORDER BY. PARTITION BY разделяет данные на группы по идентификатору пользователя (user\_id), а ORDER BY устанавливает порядок сортировки по дате заказа (order\_date).

**Запрос с агрегированием и выражением JOIN, включающим не менее 2 таблиц (3)**

1) Запрос для подсчета общего количества товаров в каждой категории:



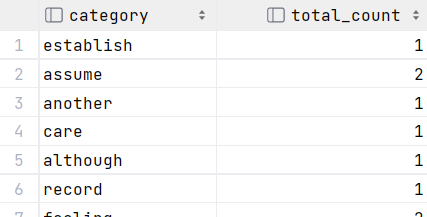
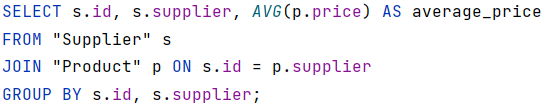


Рисунок 4.12 – Запрос №12

2) Запрос для вычисления средней стоимости товаров для каждого поставщика:



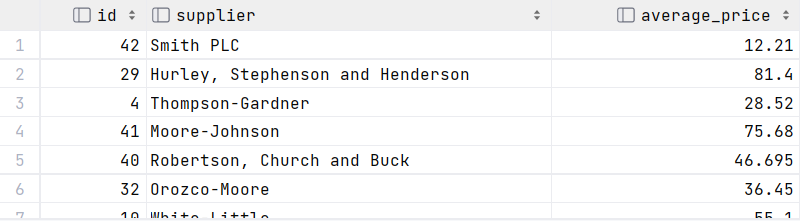
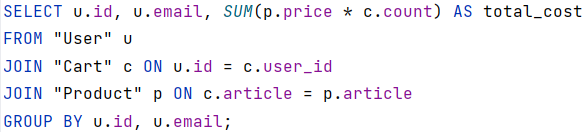


Рисунок 4.13 – Запрос №13

3) Запрос для подсчета общей стоимости всех заказов для каждого пользователя:



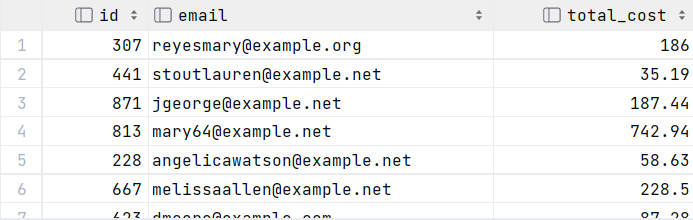
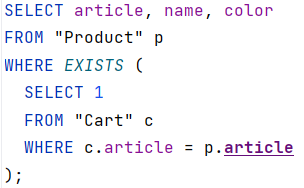


Рисунок 4.14 – Запрос №14

**Запрос с EXISTS**

Все товары, которые имеют хотя бы один заказ:



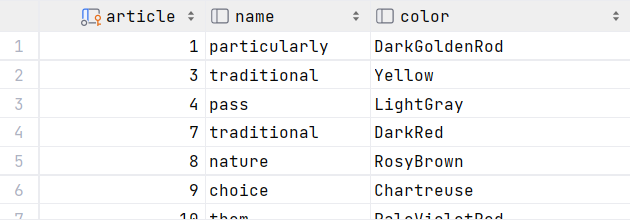
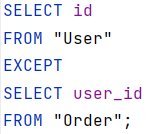


Рисунок 4.15 – Запрос №15

В этом запросе мы используем оператор EXISTS, чтобы проверить наличие хотя бы одной записи в таблице "Cart" для каждого товара в таблице "Product". Подзапрос внутри EXISTS выбирает любую запись из "Cart" с соответствующим идентификатором товара.

**Запрос, использующий манипуляции с множествами**

Cписок уникальных идентификаторов пользователей, которые имеют аккаунты и при этом не имеют заказов в таблице "Order":



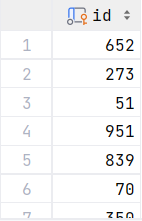
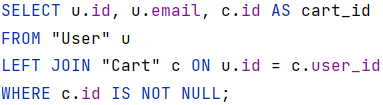


Рисунок 4.16 – Запрос №16

В этом запросе мы используем оператор EXCEPT для получения разности между множеством идентификаторов пользователей из таблицы "User" и множеством идентификаторов пользователей из таблицы "Order".

**Запрос с внешним соединением и проверкой на наличие NULL**

Список пользователей и связанных с ними корзин, но только тех, у которых есть корзины:



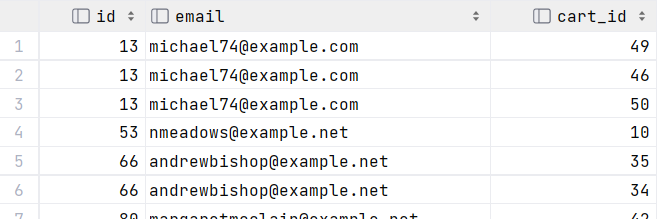


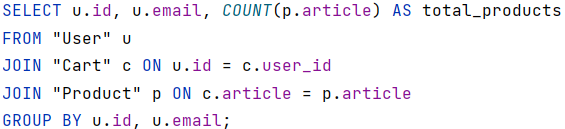
Рисунок 4.17 – Запрос №17

В этом запросе мы используем внешнее соединение LEFT JOIN для объединения таблиц "User" и "Cart" по полю "id" и "user\_id" соответственно. Это позволяет нам получить всех пользователей и связанные с ними корзины, даже если у них нет корзины.

Затем мы добавляем условие WHERE c.id IS NOT NULL для проверки наличия корзин. Это означает, что будут выбраны только те строки, где идентификатор корзины (c.id) не равен NULL.

**Запрос с агрегированием и выражением JOIN, включающим не менее 3 таблиц/выражений**

Общее количество товаров в корзинах для каждого пользователя:



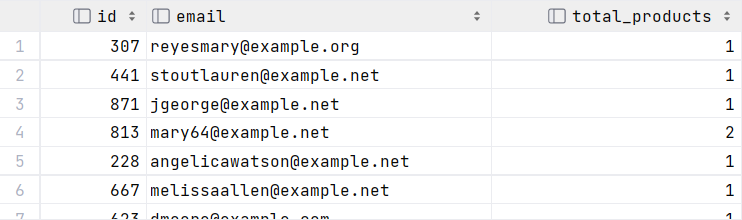


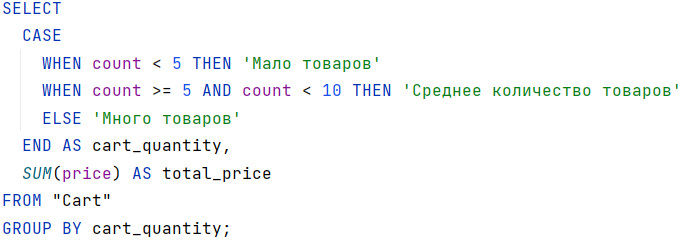
Рисунок 4.18 – Запрос №18

В этом исправленном запросе мы используем JOIN для объединения таблиц "User", "Cart" и "Product" по соответствующим связям. Затем мы используем агрегатную функцию COUNT(p.article) для подсчета количества товаров (столбец "article") в корзинах каждого пользователя.

Далее, мы используем оператор GROUP BY для группировки результатов по идентификатору пользователя (u.id) и электронной почте (u.email).

**Запрос с CASE (IIF) и агрегированием**

Сводка суммарной стоимости корзин в зависимости от их количества товаров: "Мало товаров", "Среднее количество товаров" или "Много товаров":



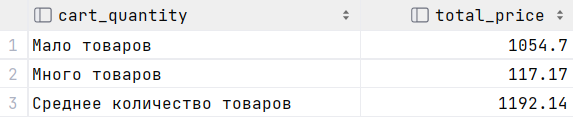


Рисунок 4.19 – Запрос №19

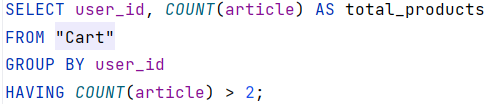
В этом запросе мы используем оператор CASE (или IIF) для проверки значения столбца "count" каждой корзины и присваивания ему соответствующего значения в новом столбце "cart\_quantity". Если количество товаров меньше 5, мы присваиваем значение 'Мало товаров', если больше или равно 5 и меньше 10, то 'Среднее количество товаров', в противном случае, 'Много товаров'.

Затем мы используем агрегатную функцию SUM(price) для подсчета общей стоимости корзин для каждой категории количества товаров.

Результатом запроса будет сводка суммарной стоимости корзин в зависимости от их количества товаров, разбитая на категории 'Мало товаров', 'Среднее количество товаров' и 'Много товаров'.

**Запрос с HAVING и агрегированием**

Список пользователей, у которых общее количество товаров в корзине превышает определенное значение (2):



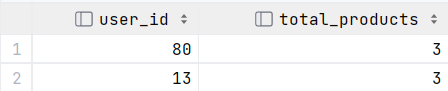
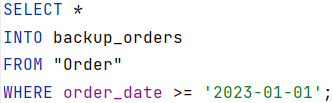


Рисунок 4.20 – Запрос №20

В этом запросе мы используем агрегатную функцию COUNT(article) для подсчета общего количества товаров в каждой корзине для каждого пользователя, группируя по полю "user\_id".

Затем мы используем оператор HAVING для фильтрации результатов. В данном примере мы выбираем только те строки, где общее количество товаров (total\_products) превышает 2.

**Запрос SELECT INTO для подготовки выгрузки**



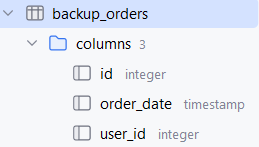


Рисунок 4.21 – Запрос №21

В этом примере мы создаем новую таблицу "backup\_orders" на основе данных из таблицы "Orders", выбирая только те строки, у которых дата заказа (order\_date) больше или равна '2023-01-01'.

## Триггер

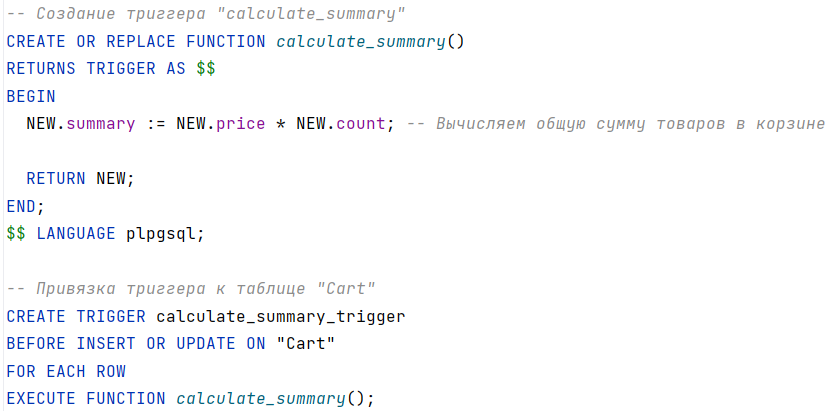


Рисунок 4.22 – Триггер

Назначение этого триггера состоит в автоматическом вычислении общей суммы товаров в корзине при каждом добавлении или обновлении записи в таблице "Cart". Триггер привязан к событиям BEFORE INSERT (перед добавлением записи) и BEFORE UPDATE (перед обновлением записи) и выполняется для каждой строки, которая подвергается изменению.

Внутри функции триггера calculate\_summary() вычисляется общая сумма товаров в корзине, умножая цену товара (price) на количество товаров (count). Результат присваивается полю summary в текущей записи (NEW). Затем функция возвращает измененную запись.

Таким образом, каждый раз при добавлении нового товара в корзину или изменении цены или количества товаров в корзине, триггер автоматически обновляет значение поля summary, обеспечивая актуальную информацию об общей сумме товаров в корзине.

## Функции

1)

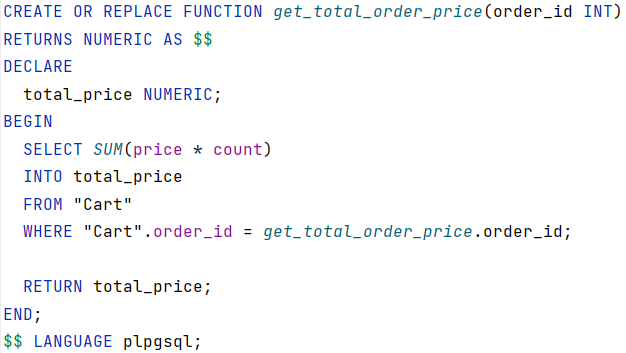


Рисунок 4.23 – Функция №1

Назначение функции: get\_total\_order\_price принимает идентификатор заказа (order\_id) в качестве параметра и возвращает общую стоимость заказа (сумму цен всех товаров в корзине, принадлежащих указанному заказу). Функция выполняет операцию суммирования (SUM) цен (price) всех товаров, умноженных на их количество (count), из таблицы "Cart". Результат сохраняется в переменной total\_price и возвращается функцией.

2)

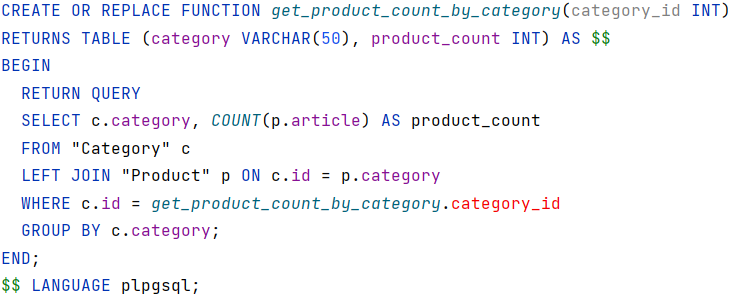


Рисунок 4.24 – Функция №2

Назначение функции: get\_product\_count\_by\_category принимает идентификатор категории (category\_id) в качестве параметра и возвращает таблицу с двумя столбцами: наименование категории (category) и количество товаров (product\_count) в данной категории. Функция выполняет операцию объединения (LEFT JOIN) таблицы "Category" и "Product" по полю id в категории и category в продукте. Затем с помощью оператора COUNT подсчитывается количество товаров (p.article) в каждой категории. Результаты группируются (GROUP BY) по наименованию категории. В итоге, функция возвращает таблицу с колонками category и product\_count.

## Представления

1)

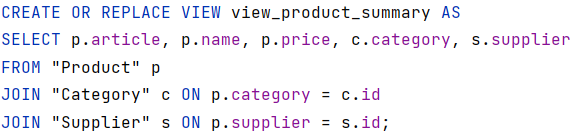


Рисунок 4.25 – Представление №1 создание



Рисунок 4.26 – Представление №1

Назначение представления: view\_product\_summary создает представление, которое содержит сводную информацию о продуктах. Представление объединяет данные из таблиц "Product", "Category" и "Supplier", используя соответствующие связи по полям category и supplier. Представление включает следующие столбцы: идентификатор продукта (article), наименование продукта (name), цена продукта (price), категория продукта (category) и поставщик продукта (supplier). Это представление может служить источником данных для отчетов о продуктах с информацией о категориях и поставщиках.

2)

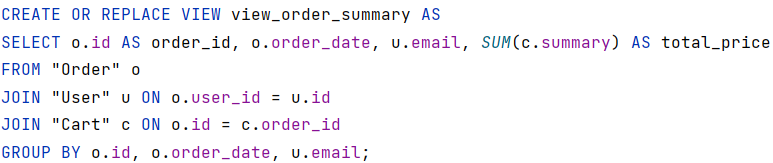


Рисунок 4.27 – Представление №2 создание

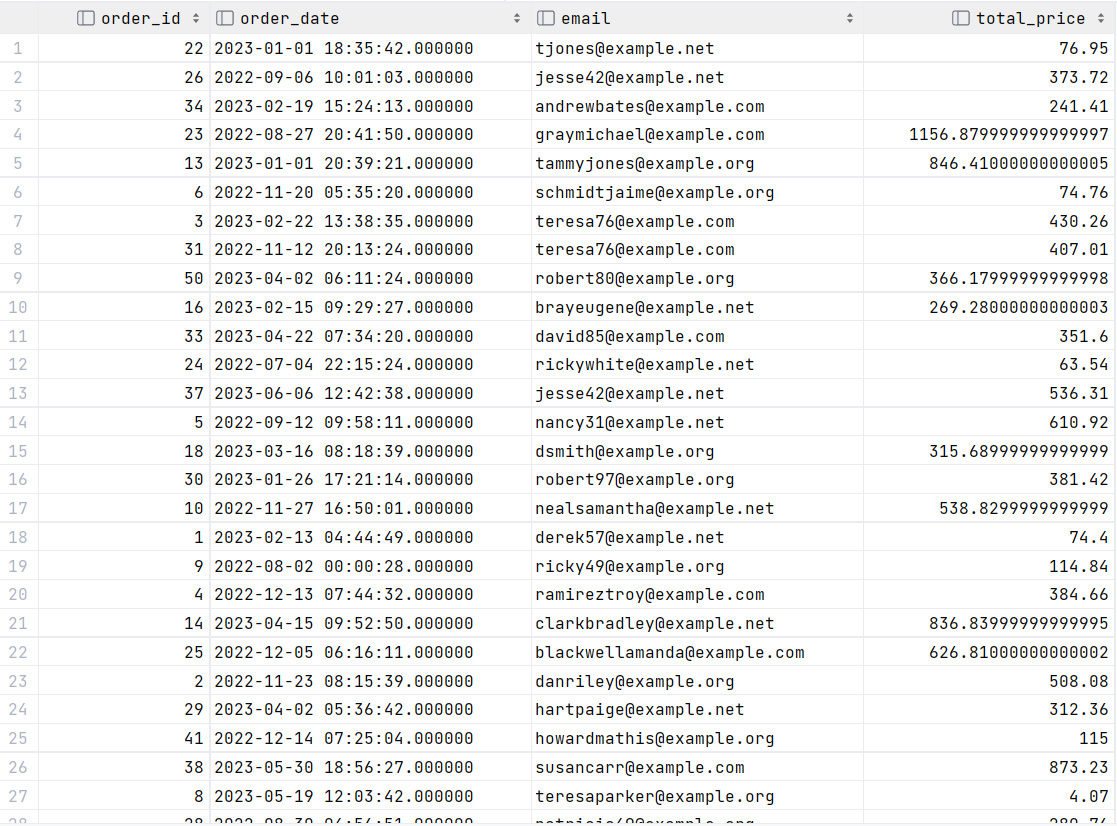


Рисунок 4.28 – Представление №2

Назначение представления: view\_order\_summary создает представление, которое содержит сводную информацию о заказах. Представление объединяет данные из таблиц "Order", "User" и "Cart", используя соответствующие связи по полям user\_id и order\_id. Представление включает следующие столбцы: идентификатор заказа (order\_id), дата заказа (order\_date), электронная почта пользователя (email) и общая стоимость заказа (total\_price). С помощью оператора SUM подсчитывается общая стоимость заказа путем суммирования полей summary из таблицы "Cart" для каждого заказа. Результаты группируются (GROUP BY) по идентификатору заказа, дате заказа и электронной почте пользователя. Это представление может использоваться для создания отчетов о сводной информации о заказах, включая общую стоимость каждого заказа и связанную информацию о пользователе.

## Хранимые процедуры

1) Процедура 1: *insert\_product*



Рисунок 4.29 – Процедура *insert\_product*

Назначение процедуры: insert\_product выполняет вставку новой записи о продукте в таблицу "Product". Процедура принимает параметры, содержащие информацию о продукте (article, name, description, season, color, price, image, category\_id, supplier\_id). Затем с помощью оператора INSERT INTO происходит вставка новой записи в таблицу "Product" со значениями, переданными через параметры. Процедура завершается командой COMMIT, подтверждающей изменения в базе данных.

2) Процедура 2: *update\_product\_price*

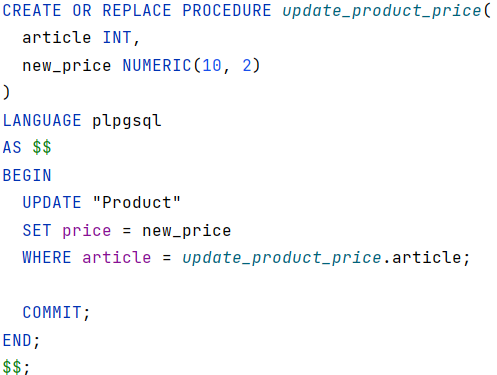


Рисунок 4.30 – Процедура *update\_product\_price*

Назначение процедуры: update\_product\_price обновляет цену продукта в таблице "Product" на основе переданного идентификатора продукта (article) и новой цены (new\_price). Процедура использует оператор UPDATE, чтобы изменить значение поля price в таблице "Product" для записи с указанным идентификатором продукта.

3) Процедура 3: *delete\_product*



Рисунок 4.31 – Процедура *delete\_product*

Назначение процедуры: delete\_product удаляет запись о продукте из таблицы "Product" на основе переданного идентификатора продукта (article). Процедура также удаляет все связанные записи в таблице "Cart", где этот продукт присутствует. Процедура использует оператор DELETE, чтобы удалить записи из таблицы "Product" и "Cart" соответственно.

4) Процедура 4: *process\_order*

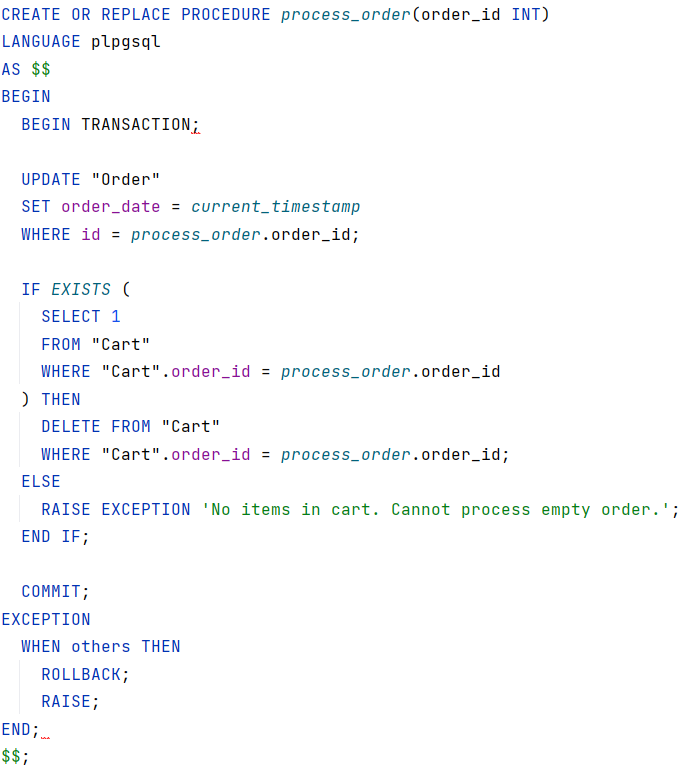


Рисунок 4.32 – Процедура *process\_order*

Назначение процедуры: process\_order обрабатывает заказ на основе переданного идентификатора заказа (order\_id). Процедура начинает транзакцию с помощью BEGIN TRANSACTION. Затем происходит обновление даты заказа (order\_date) в таблице "Order" на текущую дату и время. Если существуют записи в таблице "Cart" для указанного заказа, то они удаляются. В противном случае, генерируется исключение с сообщением о том, что невозможно обработать пустой заказ. Если происходит какая-либо ошибка во время выполнения процедуры, выполняется откат (ROLLBACK) транзакции, и исключение перехватывается для дальнейшей обработки. Если процедура завершается успешно, изменения фиксируются с помощью команды COMMIT.

5) Процедура 5: *get\_order\_total\_price*

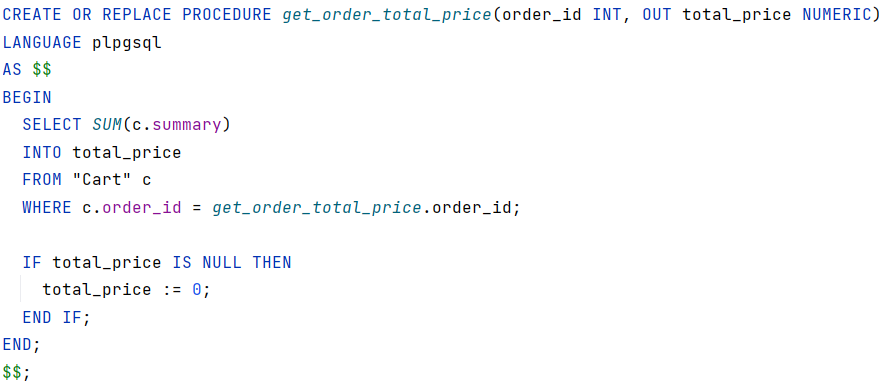


Рисунок 4.33 – Процедура *get\_order\_total\_price*

Назначение процедуры: get\_order\_total\_price вычисляет общую стоимость заказа на основе переданного идентификатора заказа (order\_id). Процедура использует оператор SELECT с функцией SUM, чтобы получить сумму значений поля summary из таблицы "Cart" для указанного заказа. Результат сохраняется в переменной total\_price. Если значение total\_price равно NULL (то есть нет записей в таблице "Cart" для данного заказа), то значение устанавливается равным 0. Конечное значение общей стоимости заказа возвращается через параметр total\_price, указанный как OUT.

# ОТЧЕТЫ

## ○ Основной отчёт Power BI – агрегированные данные со ссылками на детализацию

Отчет содержит аналитику по суммарной стоимости заказов (выручке).

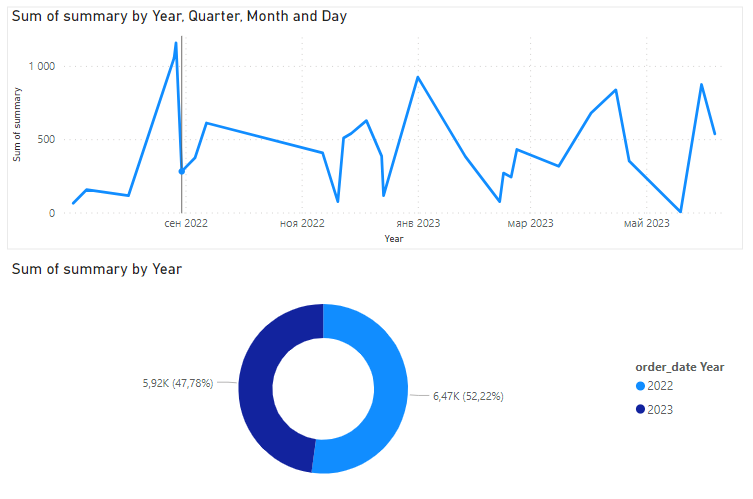


Рисунок 5.1 – Основной отчёт Power BI

## ○ Детальный отчёт Power BI для drill down

Детальный отчет по суммарной стоимости заказов позволяет увидеть аналитику по году, кварталу и месяцу.

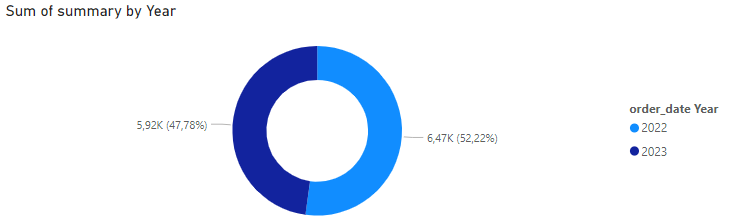


Рисунок 5.2 – Детальный отчёт Power BI №1

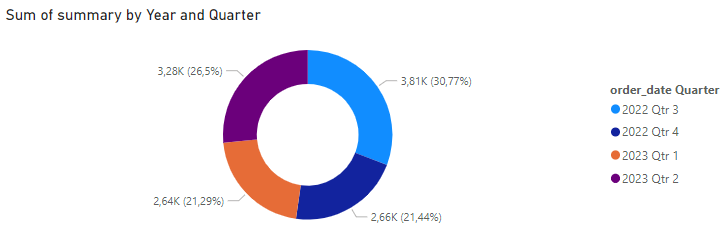


Рисунок 5.3 – Детальный отчёт Power BI №2

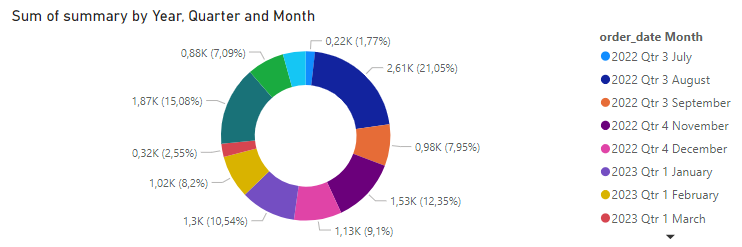


Рисунок 5.4 – Детальный отчёт Power BI №3

## ○ Информационная панель в Excel с графиком и "срезами"

На основе представления view\_order\_summary построили “график с маркерами” демонстрирующий суммарную стоимость заказов каждого из пользователей, позволяет оценить наиболее важных клиентов.

Также представлен срез по электронным почтам покупателей, для удобства в фильтрации и просмотре данных о конкретном клиенте

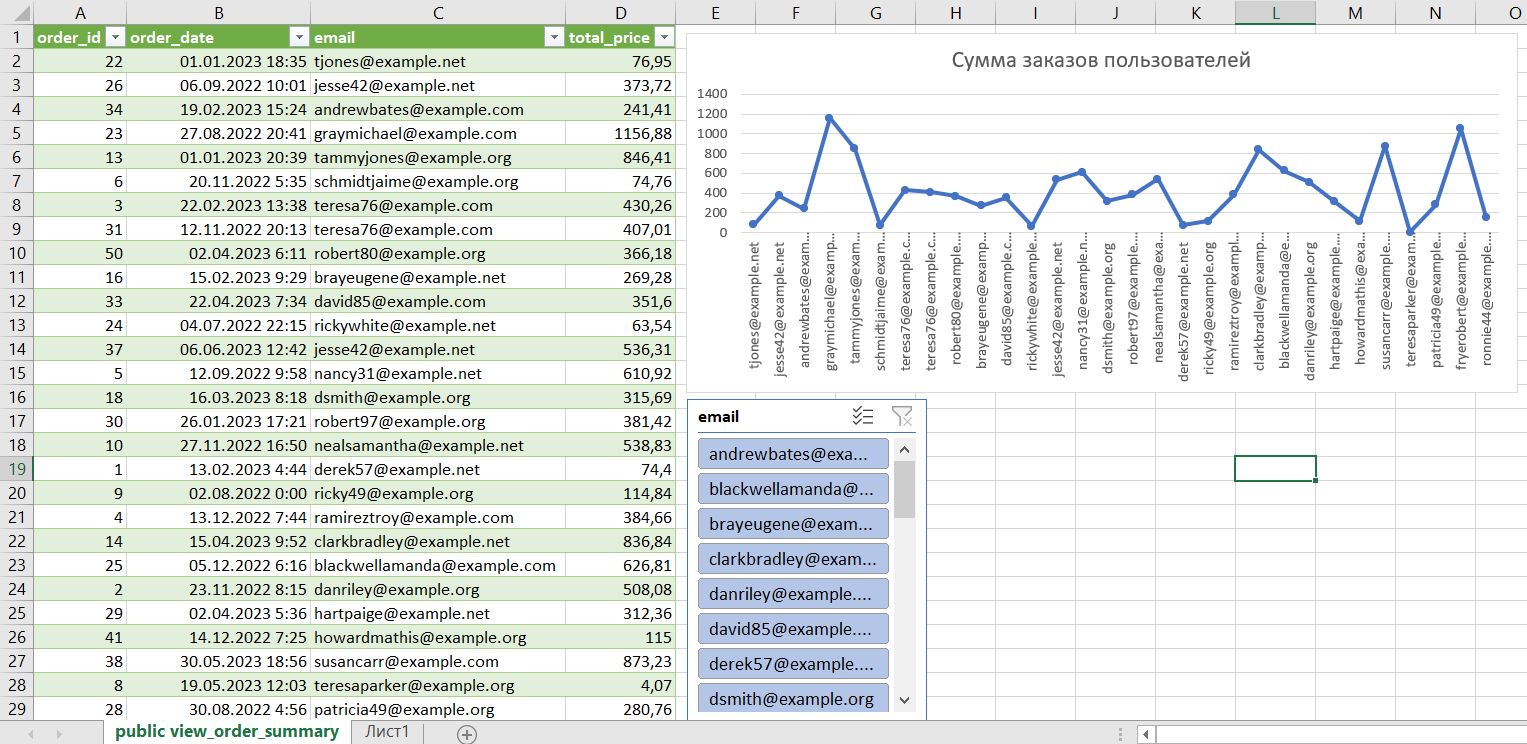


Рисунок 5.5 – Информационная панель в Excel