

# Отчет о финальной работе по курсу «Язык SQL»

Иван Плешков

# Содержание

<b>1</b>	<b>Описание предметной области</b>	<b>2</b>
<b>2</b>	<b>Подробные требования к базе данных</b>	<b>3</b>
1	Таблица Users . . . . .	3
2	Таблица Products . . . . .	4
3	Таблица Warehouse . . . . .	5
4	Таблица Categories . . . . .	6
5	Таблица Inventory_Log . . . . .	7
6	Таблица Reviews . . . . .	8
7	Таблица Orders . . . . .	9
8	Таблица Order_Items . . . . .	10
9	Таблица Deliveries . . . . .	11
<b>3</b>	<b>Политика удаления записей</b>	<b>12</b>
<b>4</b>	<b>Индексы</b>	<b>13</b>
<b>5</b>	<b>ER-диаграмма</b>	<b>13</b>
<b>6</b>	<b>Триггеры</b>	<b>14</b>
1	Обновление поля <code>updated_at</code> при изменении записи в <code>products</code> . . . . .	14
2	Логирование изменений остатков на складе . . . . .	15
3	Проверка остатков перед добавлением записи в <code>order_items</code> . . . . .	15
4	Автоматическое обновление общей стоимости заказа . . . . .	16
<b>7</b>	<b>Хранимые функции</b>	<b>16</b>
1	Расчет дохода за период . . . . .	16
2	Получение самых продаваемых товаров . . . . .	17
3	Поиск топ N покупателей по стоимости заказов . . . . .	17
<b>8</b>	<b>Типичные запросы к базе данных</b>	<b>17</b>
<b>9</b>	<b>Физическая схема базы данных</b>	<b>26</b>

# 1 Описание предметной области

В качестве предметной области была выбрана разработка базы данных для интернет-магазина. Основная цель — создать структуру базы данных, которая позволит эффективно хранить и обрабатывать данные о товарах, категориях, пользователях, заказах, отзывах и движении товаров.

Для хранения списка товаров, которые будут продаваться в магазине, будет использоваться таблица **products**. В ней будет храниться информация о названии товара, его текстовом описании, цене, категории, дате добавления товара и дате последнего изменения данных о товаре. В качестве ключа будет использоваться атрибут **id**. Значения для этого атрибута будут генерироваться PostgreSQL автоматически благодаря использованию типа данных **serial**. Можно было бы реализовать более продвинутый подход для генерации **id**, например, кодировать в значении категорию товара, дату добавления или другую информацию, но для упрощения задачи выбран такой подход. В других таблицах подход к атрибуту **id** будет аналогичным. Все возможные категории товаров будут храниться в таблице **categories**. Информация о наличии товара будет храниться в таблице **warehouse**, которая будет содержать **id** товара, его остаток на складе, дату следующей поставки и количество товара в следующей поставке. Информация о людях, взаимодействующих с магазином (покупателях или администраторах), будет храниться в таблице **users**. Эта таблица будет содержать уникальный **id**, уникальное имя пользователя, уникальный email-адрес, хэш пароля, номер телефона, дату создания аккаунта и флаг, указывающий, является ли пользователь администратором. Все заказы, созданные пользователями, будут храниться в таблице **orders**. В этой таблице будут храниться **id** заказа, **id** пользователя, сделавшего заказ, статус заказа, стоимость и дата создания заказа. Информация о всех товарах в одном заказе будет храниться в таблице **order\_items**, которая будет содержать **id** заказа, **id** продукта, количество продуктов и цену за штуку на момент заказа. Фиксация цены на момент заказа необходима, так как цена товара может изменяться. Будет создана таблица **reviews**, которая будет хранить отзывы клиентов о товарах. Она будет содержать информацию о пользователе, оставившем отзыв, **id** товара, на который был оставлен отзыв, оценку покупателя, текст отзыва и дату написания отзыва. Также будет таблица **inventory\_log**, в которой будет храниться информация о движении товаров: заказе, поставке товара или возврате.

Типичные запросы к базе данных могут быть следующими:

- Найти все товары, у которых средняя оценка больше 4, и вывести количество отзывов для каждого такого товара.
- Получить список всех заказов пользователя с их общей стоимостью и списком товаров в каждом заказе.
- Найти топ-5 самых продаваемых товаров.
- Вывести заказы пользователей, отсортированные по стоимости, с добавлением позиции в списке (ранга).

Типичными транзакциями для этой базы могут быть:

- **Оформление заказа:**
  - Добавить запись в таблицу **orders**.
  - Для каждого товара в заказе добавить запись в таблицу **order\_items**.
  - Уменьшить количество товара на складе.

- Добавить запись в таблицу `inventory_log`.

- **Возврат товара:**

- Добавить запись о возврате товара в таблицу `inventory_log`.
- Увеличить остаток товара на складе.
- Обновить статус заказа или удалить заказ.

## 2 Подробные требования к базе данных

### 1 Таблица Users

Мы будем хранить данные о всех пользователях (клиентов и администраторов) в таблице `users`. Эта таблица используется для аутентификации пользователей, хранения их контактной информации и управления правами доступа. Таблица находится в ЗНФ. Атрибуты:

- `id`: Уникальный идентификатор пользователя, будет первичным ключом.
- `username`: Имя пользователя (уникальное), длина ограничена до 50 символов.
- `email`: Электронная почта (уникальная), длина ограничена до 100 символов. Электронная почта должна быть валидной (например, `example@mail.com`), будем считать, что проверка выполняется на уровне приложения.
- `password_hash`: Хэш пароля, который хранится для обеспечения безопасности.
- `phone_number`: Номер телефона, длина ограничена до 20 символов. Проверка формата осуществляется на уровне приложения.
- `created_at`: Дата регистрации. Заполняется автоматически текущей датой и временем в формате `TIMESTAMP`.
- `is_admin`: Признак, является ли пользователь администратором. По умолчанию значение равно `false`, что означает, что пользователь — клиент.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
username	Имя пользователя	VARCHAR(50)	UNIQUE	NOT NULL
email	Электронная почта	VARCHAR(100)	UNIQUE	NOT NULL
password_hash	Хэш пароля	TEXT		NOT NULL
phone_number	Номер телефона	VARCHAR(20)		
created_at	Дата регистрации	TIMESTAMP		DEFAULT NOW(), NOT NULL
is_admin	Администратор (да/нет)	BOOLEAN		DEFAULT FALSE, NOT NULL

Таблица 1: Структура таблицы Users

## 2 Таблица Products

В этой таблице будет храниться информация о товарах, продаваемых в магазине. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор товара, будет первичным ключом.
- **name**: Название товара, длина ограничена до 100 символов.
- **description**: Описание товара.
- **price**: Цена товара в формате `NUMERIC(10, 2)` для хранения до двух знаков после запятой. Значение должно быть больше нуля.
- **category\_id**: Ссылка на категорию товара, внешний ключ на таблицу **Categories**. Проверяется, что категория существует в таблице **Categories**.
- **created\_at**: Дата добавления товара. Заполняется автоматически текущей датой и временем в формате `TIMESTAMP`.
- **updated\_at**: Дата последнего изменения информации о товаре. Заполняется автоматически.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
name	Название товара	VARCHAR(100)		NOT NULL
description	Описание товара	TEXT		
price	Цена товара	NUMERIC(10, 2)		NOT NULL, CHECK(price > 0)
category_id	Ссылка на категорию	INT	FOREIGN KEY	REFERENCES Categories(id), NOT NULL, проверка существования категории
created_at	Дата добавления	TIMESTAMP		DEFAULT NOW(), NOT NULL
updated_at	Дата изменения	TIMESTAMP		DEFAULT NOW(), NOT NULL

Таблица 2: Структура таблицы Products

### 3 Таблица Warehouse

Эта таблица будет содержать информацию о происходящем на складе. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор записи, будет первичным ключом.
- **product\_id**: Ссылка на товар, внешний ключ на таблицу Products.
- **stock\_quantity**: Текущее количество товара на складе. Значение должно быть больше или равно нулю.
- **next\_delivery\_date**: Дата следующей поставки.
- **next\_delivery\_quantity**: Количество товара в следующей поставке. Значение должно быть больше нуля, если дата поставки указана.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
product_id	Ссылка на товар	INT	FOREIGN KEY	REFERENCES Products(id), NOT NULL
stock_quantity	Текущее количество	INT		NOT NULL, CHECK(stock_quantity >= 0)
next_delivery_date	Дата следующей поставки	DATE		
next_delivery_quantity	Количество в поставке	INT		CHECK(next_delivery_quantity > 0)

Таблица 3: Структура таблицы Warehouse

## 4 Таблица Categories

Эта таблица будет хранить возможные категории товаров. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор категории, будет первичным ключом.
- **name**: Название категории, длина ограничена до 100 символов. Значение должно быть уникальным.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
name	Название категории	VARCHAR(100)	UNIQUE	NOT NULL

Таблица 4: Структура таблицы Categories

## 5 Таблица Inventory\_Log

Эта таблица будет содержать информацию об изменениях количества товара на складе. Таблица находится в 3НФ. Атрибуты:

- **id**: Уникальный идентификатор записи, будет первичным ключом.
- **product\_id**: Ссылка на товар, внешний ключ на таблицу **Products**.
- **change\_amount**: Изменение количества товара на складе. Может быть положительным (поставка) или отрицательным (списание).
- **change\_type**: Тип изменения, например, "поставка", "продажа", "возврат" и т.д. Длина ограничена до 50 символов.
- **created\_at**: Дата и время изменения. Заполняется автоматически текущей датой и временем в формате **TIMESTAMP**.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
product_id	Ссылка на товар	INT	FOREIGN KEY	REFERENCES Products(id), NOT NULL
change_amount	Изменение количества	INT		NOT NULL
change_type	Тип изменения	VARCHAR(50)		NOT NULL
created_at	Дата изменения	TIMESTAMP		DEFAULT NOW(), NOT NULL

Таблица 5: Структура таблицы Inventory\_Log



## 6 Таблица Reviews

Эта таблица будет хранить отзывы пользователей на товары. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор записи, будет первичным ключом.
- **user\_id**: Ссылка на пользователя, внешний ключ на таблицу **Users**.
- **product\_id**: Ссылка на товар, внешний ключ на таблицу **Products**.
- **rating**: Рейтинг товара, числовое значение, например, от 1 до 5.
- **comment**: Текстовый отзыв пользователя о товаре.
- **created\_at**: Дата и время создания отзыва. Заполняется автоматически текущей датой и временем в формате **TIMESTAMP**.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
user_id	Ссылка на пользователя	INT	FOREIGN KEY	REFERENCES Users(id), NOT NULL
product_id	Ссылка на товар	INT	FOREIGN KEY	REFERENCES Products(id), NOT NULL
rating	Рейтинг товара	SMALLINT		NOT NULL, CHECK(rating >= 1 AND rating <= 5)
comment	Текстовый отзыв	TEXT		
created_at	Дата создания	TIMESTAMP		DEFAULT NOW(), NOT NULL

Таблица 6: Структура таблицы Reviews

## 7 Таблица Orders

В этой таблице будет храниться информация о заказах. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор заказа, будет первичным ключом.
- **user\_id**: Ссылка на пользователя, который совершил заказ, внешний ключ на таблицу **Users**.
- **status**: Статус заказа, например, "Новый", "В обработке", "Отправлен", "Доставлен" и т.д. Длина ограничена до 50 символов.
- **total\_price**: Общая стоимость заказа, хранится в формате **NUMERIC(10, 2)** для двух знаков после запятой.
- **created\_at**: Дата и время создания заказа. Заполняется автоматически текущей датой и временем в формате **TIMESTAMP**.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
user_id	Ссылка на пользователя	INT	FOREIGN KEY	REFERENCES Users(id), NOT NULL
status	Статус заказа	VARCHAR(50)		NOT NULL
total_price	Общая стоимость	NUMERIC(10, 2)		NOT NULL, CHECK(total_price >= 0)
created_at	Дата создания	TIMESTAMP		DEFAULT NOW(), NOT NULL

Таблица 7: Структура таблицы Orders

## 8 Таблица Order\_Items

В этой таблице будет храниться информация какие товары есть в конкретном заказе и их количество. Таблица находится в 3НФ. Атрибуты:

- **id**: Уникальный идентификатор записи, будет первичным ключом.
- **order\_id**: Ссылка на заказ, внешний ключ на таблицу **Orders**.
- **product\_id**: Ссылка на товар, внешний ключ на таблицу **Products**.
- **quantity**: Количество единиц товара в заказе. Значение должно быть больше нуля.
- **price**: Цена за единицу товара на момент оформления заказа, хранится в формате **NUMERIC(10, 2)**.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
order_id	Ссылка на заказ	INT	FOREIGN KEY	REFERENCES Orders(id), NOT NULL
product_id	Ссылка на товар	INT	FOREIGN KEY	REFERENCES Products(id), NOT NULL
quantity	Количество товара	INT		NOT NULL, CHECK(quantity > 0)
price	Цена за единицу	NUMERIC(10, 2)		NOT NULL, CHECK(price >= 0)

Таблица 8: Структура таблицы Order\_Items

## 9 Таблица Deliveries

В этой таблице будет храниться информация о доставке для каждого заказа. Таблица находится в ЗНФ. Атрибуты:

- **id**: Уникальный идентификатор записи, будет первичным ключом.
- **order\_id**: Ссылка на заказ, внешний ключ на таблицу **Orders**.
- **delivery\_type**: Тип доставки, например, "курьер", "самовывоз", "почта". Длина ограничена до 50 символов. Допускаются только значения "самовывоз", "курьер", "почта".
- **address**: Адрес доставки, если применяется. Поле может быть пустым для самовывоза.
- **delivery\_date**: Дата предполагаемой или фактической доставки.

Столбец	Описание	Тип данных	Ключи	Дополнительная информация
id	Уникальный идентификатор	SERIAL	PRIMARY KEY	
order_id	Ссылка на заказ	INT	FOREIGN KEY	REFERENCES Orders(id), NOT NULL
delivery_type	Тип доставки	VARCHAR(50)		NOT NULL, CHECK(delivery_type IN ('самовывоз', 'курьер', 'почта'))
address	Адрес доставки	TEXT		Может быть NULL для самовывоза
delivery_date	Дата доставки	DATE		

Таблица 9: Структура таблицы Deliveries

## 3 Политика удаления записей

В этом разделе описана логика работы базы данных при удалении записей из таблиц.

### 1. Таблица `users`

- При удалении пользователя:
  - Удаляются все связанные записи в таблицах:
    - \* `orders` (заказы, оформленные пользователем).
    - \* `reviews` (отзывы, оставленные пользователем).

### 2. Таблица `categories`

- Удаление категории запрещено, если есть товары, относящиеся к этой категории. При попытке удаления такой категории операция будет заблокирована.

### 3. Таблица `products`

- При удалении товара:
  - Позиции заказов, содержащие товар (`order_items`), остаются.
  - Отзывы на товар (`reviews`) удаляются.
  - Записи движения товара на складе (`inventory_log`) остаются.
  - Запасы на складе (`warehouse`) удаляются.

### 4. Таблица `orders`

- При удалении заказа:
  - Удаляются все связанные записи в таблицах:
    - \* `order_items` (товары в составе заказа).
    - \* `deliveries` (информация о доставке заказа).

### 5. Таблица `order_items`

- Записи в этой таблице автоматически удаляются при удалении связанных записей из таблицы `orders`.
- Прямое удаление записей из таблицы `order_items` запрещено.

### 6. Таблица `deliveries`

- Записи в этой таблице автоматически удаляются при удалении связанных записей из таблицы `orders`.
- Прямое удаление записей из таблицы `deliveries` запрещено.

## 7. Таблица reviews

- При удалении записи о пользователе (**users**) или товаре (**products**):
  - Удаляются все связанные отзывы.

## 8. Таблица inventory\_log

- Записи из таблицы удаляются вручную, поскольку они не зависят от других таблиц напрямую.
- Возможность удаления записей предусмотрена для случаев корректировки данных или архивирования.

## 9. Таблица warehouse

- При удалении товара (**products**):
  - Удаляются все связанные записи в таблице **warehouse**.
- Удаление записей напрямую из таблицы **warehouse** запрещено.

# 4 Индексы

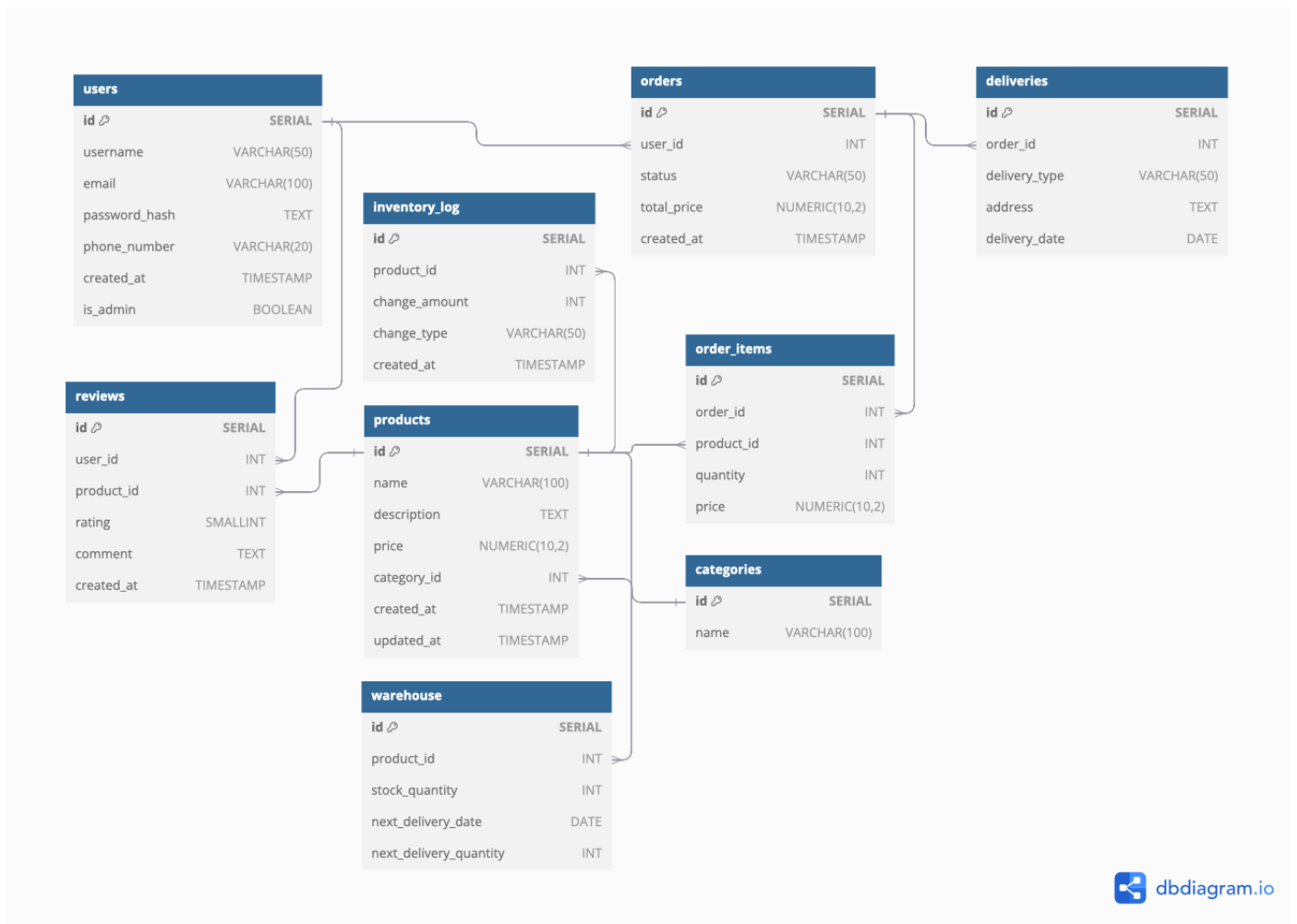
Для ускорения выполнения частых запросов к базе данных добавим несколько индексов:

- В таблицу **users** добавим индекс на **username**, чтобы ускорить поиск имени пользователя, например, при входе в личный кабинет.
- В таблицу **orders** добавим индекс на **user\_id**, чтобы быстро находить все заказы, сделанные конкретным пользователем.
- В таблицу **warehouse** добавим индекс на **product\_id**, чтобы быстро узнавать остаток товара на складе.
- В таблицу **order\_items** добавим два индекса:
  - На **order\_id**, чтобы ускорить соединение с таблицей **orders**.
  - На **product\_id**, чтобы быстро находить заказы, в которых присутствует конкретный товар.

# 5 ER-диаграмма

ER-диаграмма базы данных была построена с использованием сервиса [dbdiagram.io](https://dbdiagram.io) и языка описания схем **DBML**. Для удобства, более подробную и интерактивную версию диаграммы можно найти по следующей ссылке: <https://dbdiagram.io/d/67654c19fc29fb2b3bf81c15>.

Ниже представлен скриншот диаграммы, отображающий основные сущности и связи базы данных.



## 6 Триггеры

Для автоматизации процессов в базе данных интернет-магазина были разработаны следующие триггеры:

### 1 Обновление поля `updated_at` при изменении записи в `products`

Этот триггер автоматически обновляет значение поля `updated_at` в таблице `products` при изменении любой записи. Это позволяет отслеживать дату и время последнего обновления данных о товаре.

```

CREATE OR REPLACE FUNCTION update_product_timestamp()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
  
```

```

CREATE TRIGGER trigger_update_product_timestamp
  
```

```
BEFORE UPDATE ON products
FOR EACH ROW
EXECUTE FUNCTION update_product_timestamp();
```

## 2 Логирование изменений остатков на складе

Этот триггер добавляет запись в таблицу `inventory_log` при уменьшении остатков на складе. Тип изменения фиксируется как 'Продажа'.

```
CREATE OR REPLACE FUNCTION log_inventory_decrease()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.stock_quantity < OLD.stock_quantity THEN
        INSERT INTO inventory_log (product_id, change_amount, change_type, created_at)
        VALUES (NEW.product_id, OLD.stock_quantity - NEW.stock_quantity, 'Продажа', NOW());
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trigger_log_inventory_decrease
AFTER UPDATE OF stock_quantity ON warehouse
FOR EACH ROW
WHEN (OLD.stock_quantity > NEW.stock_quantity)
EXECUTE FUNCTION log_inventory_decrease();
```

## 3 Проверка остатков перед добавлением записи в order\_items

Этот триггер проверяет наличие достаточного количества товара на складе перед добавлением записи в таблицу `order_items`. Если остатка недостаточно, триггер генерирует исключение.

```
CREATE OR REPLACE FUNCTION check_stock_before_insert()
RETURNS TRIGGER AS $$
DECLARE
    available_stock INT;
BEGIN
    SELECT stock_quantity INTO available_stock
    FROM warehouse
    WHERE product_id = NEW.product_id;

    IF available_stock < NEW.quantity THEN
        RAISE EXCEPTION 'Недостаточно товара на складе для продукта ID %', NEW.product_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



```
CREATE TRIGGER trigger_check_stock_before_insert
BEFORE INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION check_stock_before_insert();
```

## 4 Автоматическое обновление общей стоимости заказа

Этот триггер автоматически пересчитывает общую стоимость заказа в таблице `orders` при добавлении, обновлении или удалении записи в таблице `order_items`. Это гарантирует, что поле `total_price` всегда содержит актуальную сумму.

```
CREATE OR REPLACE FUNCTION calculate_order_total()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE orders
    SET total_price = (
        SELECT SUM(quantity * price)
        FROM order_items
        WHERE order_id = NEW.order_id
    )
    WHERE id = NEW.order_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_order_total
AFTER INSERT OR UPDATE OR DELETE ON order_items
FOR EACH ROW
EXECUTE FUNCTION calculate_order_total();
```

## 7 Хранимые функции

Для выполнения аналитических задач и упрощения операций в базе данных интернет-магазина были разработаны следующие хранимые функции:

### 1 Расчет дохода за период

Эта функция позволяет рассчитать общий доход за указанный период на основании данных из таблиц `order_items` и `orders`. Она принимает две даты в качестве параметров и возвращает общий доход.

```
CREATE OR REPLACE FUNCTION calculate_revenue(start_date DATE, end_date DATE)
RETURNS NUMERIC AS $$
DECLARE
    total_revenue NUMERIC := 0;
BEGIN
```

```

SELECT SUM(oi.quantity * oi.price) INTO total_revenue
FROM order_items oi
JOIN orders o ON oi.order_id = o.id
WHERE o.created_at BETWEEN start_date AND end_date;

RETURN total_revenue;
END;
$$ LANGUAGE plpgsql;

```

## 2 Получение самых продаваемых товаров

Эта функция возвращает топ N товаров, отсортированных по количеству продаж. Она принимает количество позиций в результирующем списке в качестве параметра.

```

CREATE OR REPLACE FUNCTION get_top_selling_products(limit_count INT)
RETURNS TABLE(product_id INT, total_sold INT) AS $$
BEGIN
    RETURN QUERY
    SELECT oi.product_id, SUM(oi.quantity) AS total_sold
    FROM order_items oi
    GROUP BY oi.product_id
    ORDER BY total_sold DESC
    LIMIT limit_count;
END;
$$ LANGUAGE plpgsql;

```

## 3 Поиск топ N покупателей по стоимости заказов

Эта функция возвращает топ N покупателей, отсортированных по общей стоимости сделанных ими заказов. Она принимает количество позиций в результирующем списке в качестве параметра.

```

CREATE OR REPLACE FUNCTION get_top_customers(limit_count INT)
RETURNS TABLE(user_id INT, total_spent NUMERIC) AS $$
BEGIN
    RETURN QUERY
    SELECT o.user_id, SUM(o.total_price) AS total_spent
    FROM orders o
    GROUP BY o.user_id
    ORDER BY total_spent DESC
    LIMIT limit_count;
END;
$$ LANGUAGE plpgsql;

```

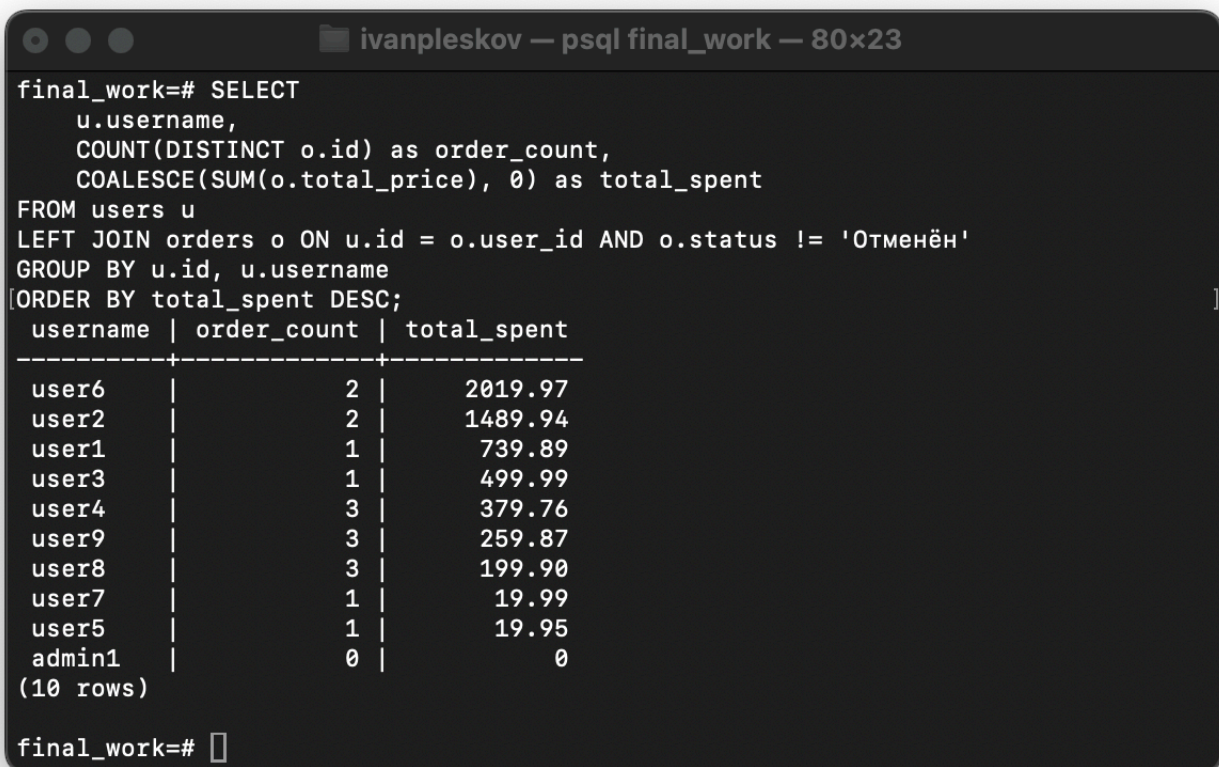
## 8 Типичные запросы к базе данных

В этом разделе приведены примеры нескольких возможных запросов к базе данных и результаты их выполнения, код запросов находится в соответствующих файлах.

## Получить список пользователей с количеством их заказов и общей суммой покупок (user\_order\_summary)

Этот запрос возвращает пользователей, количество их заказов и общую сумму их покупок. Пользователи без заказов включены в результат с нулевыми значениями.

```
SELECT
    u.username,
    COUNT(DISTINCT o.id) as order_count,
    COALESCE(SUM(o.total_price), 0) as total_spent
FROM users u
LEFT JOIN orders o ON u.id = o.user_id AND o.status != 'Отменён'
GROUP BY u.id, u.username
ORDER BY total_spent DESC;
```



The screenshot shows a terminal window titled "ivanpleskov — psql final\_work — 80x23". The query is executed, and the results are displayed in a table format. The table has three columns: username, order\_count, and total\_spent. The results are sorted by total\_spent in descending order. There are 10 rows in total, including the 'admin1' user with 0 orders and 0 total spent.

```
final_work=# SELECT
    u.username,
    COUNT(DISTINCT o.id) as order_count,
    COALESCE(SUM(o.total_price), 0) as total_spent
FROM users u
LEFT JOIN orders o ON u.id = o.user_id AND o.status != 'Отменён'
GROUP BY u.id, u.username
ORDER BY total_spent DESC;
```

username	order_count	total_spent
user6	2	2019.97
user2	2	1489.94
user1	1	739.89
user3	1	499.99
user4	3	379.76
user9	3	259.87
user8	3	199.90
user7	1	19.99
user5	1	19.95
admin1	0	0

(10 rows)

```
final_work=#
```

## Найти топ-5 самых продаваемых товаров по количеству за последний месяц (top\_5\_products\_last\_month)

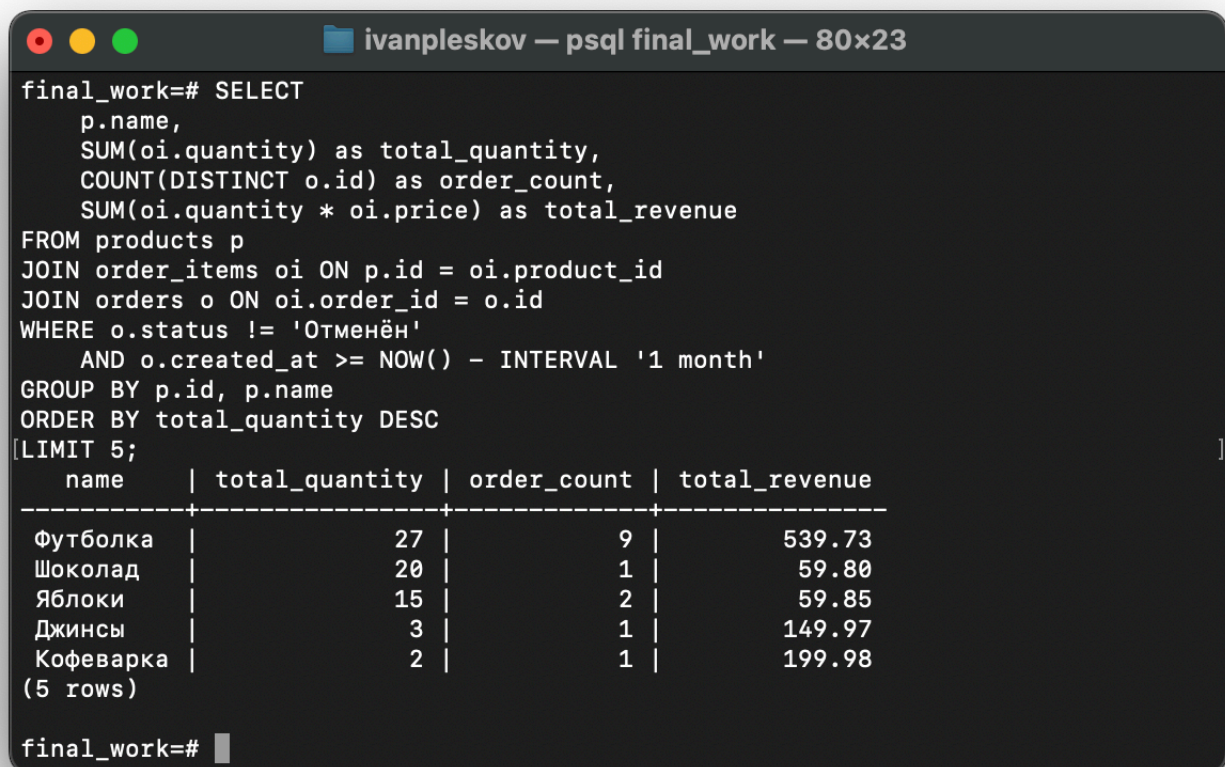
Этот запрос возвращает пять самых продаваемых товаров по количеству, с указанием количества заказов, общего дохода и количества проданных единиц за последний месяц.

```
SELECT
```

```

    p.name,
    SUM(oi.quantity) as total_quantity,
    COUNT(DISTINCT o.id) as order_count,
    SUM(oi.quantity * oi.price) as total_revenue
FROM products p
JOIN order_items oi ON p.id = oi.product_id
JOIN orders o ON oi.order_id = o.id
WHERE o.status != 'Отменён'
    AND o.created_at >= NOW() - INTERVAL '1 month'
GROUP BY p.id, p.name
ORDER BY total_quantity DESC
LIMIT 5;

```



The screenshot shows a terminal window titled "ivanpleskov — psql final\_work — 80x23". The SQL query from the previous block is entered and executed. The output shows the query text followed by a table of results with 5 rows. The table has four columns: name, total\_quantity, order\_count, and total\_revenue. The results are as follows:

name	total_quantity	order_count	total_revenue
Футболка	27	9	539.73
Шоколад	20	1	59.80
Яблоки	15	2	59.85
Джинсы	3	1	149.97
Кофеварка	2	1	199.98

The terminal also shows "(5 rows)" and the prompt "final\_work=#" with a cursor.

## Анализ динамики продаж по месяцам (monthly\_sales\_trend\_analysis)

Этот запрос возвращает динамику продаж по месяцам, включая доход за текущий месяц, доход за предыдущий и разницу в процентах.

```
WITH monthly_sales AS (
```

```

SELECT
    DATE_TRUNC('month', o.created_at) as month,
    SUM(o.total_price) as revenue
FROM orders o
WHERE o.status != 'Отменён'
GROUP BY DATE_TRUNC('month', o.created_at)
)
SELECT
    month,
    revenue,
    LAG(revenue) OVER (ORDER BY month) as prev_month_revenue,
    ROUND(
        ((revenue - LAG(revenue) OVER (ORDER BY month)) /
        LAG(revenue) OVER (ORDER BY month) * 100)::numeric,
        2
    ) as growth_percentage
FROM monthly_sales
ORDER BY month DESC;

```

```
ivanpleskov — psql final_work — 80x31
final_work=# WITH monthly_sales AS (
  SELECT
    DATE_TRUNC('month', o.created_at) as month,
    SUM(o.total_price) as revenue
  FROM orders o
  WHERE o.status != 'Отменён'
  GROUP BY DATE_TRUNC('month', o.created_at)
)
SELECT
  month,
  revenue,
  LAG(revenue) OVER (ORDER BY month) as prev_month_revenue,
  ROUND(
    ((revenue - LAG(revenue) OVER (ORDER BY month)) /
     LAG(revenue) OVER (ORDER BY month) * 100)::numeric,
    2
  ) as growth_percentage
FROM monthly_sales
[ORDER BY month DESC;
  month | revenue | prev_month_revenue | growth_percentage
  -----+-----+-----+-----
  2024-12-01 00:00:00 | 5629.26 | 159.95 | 3419.39
  2024-11-01 00:00:00 | 159.95 | 1999.98 | -92.00
  2024-10-01 00:00:00 | 1999.98 | 54.85 | 3546.27
  2024-09-01 00:00:00 | 54.85 | 179.96 | -69.52
  2024-08-01 00:00:00 | 179.96 | 699.97 | -74.29
  2024-07-01 00:00:00 | 699.97 | 2089.95 | -66.51
  2024-06-01 00:00:00 | 2089.95 |
(7 rows)

final_work=#
```

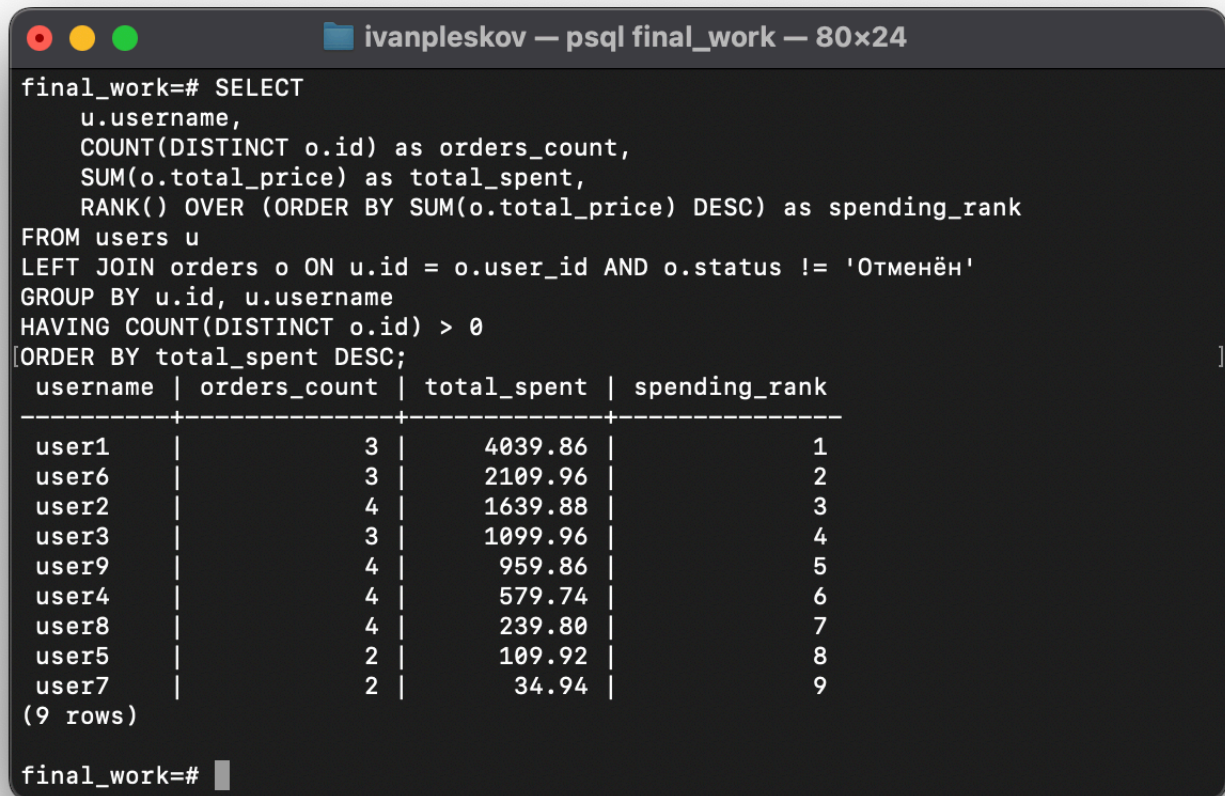
## Ранжирование пользователей по сумме покупок (user\_spending\_ranking)

Этот запрос ранжирует пользователей по общей сумме покупок.

```
SELECT
  u.username,
  COUNT(DISTINCT o.id) as orders_count,
  SUM(o.total_price) as total_spent,
  RANK() OVER (ORDER BY SUM(o.total_price) DESC) as spending_rank
FROM users u
LEFT JOIN orders o ON u.id = o.user_id AND o.status != 'Отменён'
GROUP BY u.id, u.username
```



```
HAVING COUNT(DISTINCT o.id) > 0
ORDER BY total_spent DESC;
```



The screenshot shows a terminal window titled "ivanpleskov — psql final\_work — 80x24". The prompt is "final\_work=#". The user enters a SQL query to select user information along with order statistics, ranked by total spent. The results are displayed in a table with 4 columns: username, orders\_count, total\_spent, and spending\_rank. There are 9 rows of data.

```
final_work=# SELECT
    u.username,
    COUNT(DISTINCT o.id) as orders_count,
    SUM(o.total_price) as total_spent,
    RANK() OVER (ORDER BY SUM(o.total_price) DESC) as spending_rank
FROM users u
LEFT JOIN orders o ON u.id = o.user_id AND o.status != 'Отменён'
GROUP BY u.id, u.username
HAVING COUNT(DISTINCT o.id) > 0
[ORDER BY total_spent DESC;
```

username	orders_count	total_spent	spending_rank
user1	3	4039.86	1
user6	3	2109.96	2
user2	4	1639.88	3
user3	3	1099.96	4
user9	4	959.86	5
user4	4	579.74	6
user8	4	239.80	7
user5	2	109.92	8
user7	2	34.94	9

```
(9 rows)

final_work=#
```

## Список товаров с наибольшим остатком на складе (top\_products\_by\_stock)

Этот запрос выводит информацию о 10 товарах, их остатках на складе и дате следующей поставки, отсортированных по убыванию количества.

```
SELECT
    p.name,
    w.stock_quantity,
    w.next_delivery_date
FROM warehouse w
JOIN products p ON w.product_id = p.id
ORDER BY w.stock_quantity DESC
LIMIT 10;
```

```
ivanpleskov — psql final_work — 80x24

final_work=# SELECT
    p.name,
    w.stock_quantity,
    w.next_delivery_date
FROM warehouse w
JOIN products p ON w.product_id = p.id
ORDER BY w.stock_quantity DESC
[LIMIT 10; ]

   name          | stock_quantity | next_delivery_date
-----+-----+-----
 Шоколад         |          200   | 2023-12-24
 Яблоки          |          150   | 2023-12-23
 Футболка        |          100   | 2023-12-25
 Смартфон        |           50   | 2023-12-30
 Джинсы          |           40   | 2023-12-26
 Ноутбук         |           30   | 2023-12-28
 Кофеварка       |           20   | 2023-12-29
 Мяч футбольный  |           15   | 2023-12-30
 Холодильник     |           10   | 2023-12-27
 Ракетка теннисная |           10   | 2023-12-28
(10 rows)

final_work=#
```

## Список заказов с суммой более 1000 (orders\_above\_1000)

Этот запрос находит заказы, общая сумма которых больше 1000.

```
SELECT
    o.id AS order_id,
    u.username,
    o.total_price,
    o.created_at
FROM orders o
JOIN users u ON o.user_id = u.id
WHERE o.total_price > 1000
ORDER BY o.total_price DESC;
```



```
ivanpleskov — psql final_work — 80x18
final_work=# SELECT
    o.id AS order_id,
    u.username,
    o.total_price,
    o.created_at
FROM orders o
JOIN users u ON o.user_id = u.id
WHERE o.total_price > 1000
ORDER BY o.total_price DESC;
 order_id | username | total_price |          created_at
-----+-----+-----+-----
      18 | user1    |    1999.98 | 2024-06-01 00:00:00
       9 | user6    |    1999.98 | 2024-12-23 12:10:47.213193
       2 | user2    |    1339.97 | 2024-12-23 12:10:47.213193
      27 | user1    |    1299.99 | 2024-10-20 00:00:00
(4 rows)

final_work=#
```

## Ранжирование товаров по количеству продаж (product\_sales\_ranking)

Этот запрос ранжирует товары по количеству продаж.

```
SELECT
    p.name AS product_name,
    SUM(oi.quantity) AS total_sold,
    RANK() OVER (ORDER BY SUM(oi.quantity) DESC) AS sales_rank
FROM products p
JOIN order_items oi ON p.id = oi.product_id
GROUP BY p.id, p.name
ORDER BY sales_rank;
```

```
ivanpleskov — psql final_work — 80x23
final_work=# SELECT
    p.name AS product_name,
    SUM(oi.quantity) AS total_sold,
    RANK() OVER (ORDER BY SUM(oi.quantity) DESC) AS sales_rank
FROM products p
JOIN order_items oi ON p.id = oi.product_id
GROUP BY p.id, p.name
[ORDER BY sales_rank;
 product_name | total_sold | sales_rank
-----+-----+-----
 Футболка    |         32 |         1
 Яблоки      |         25 |         2
 Шоколад     |         25 |         2
 Джинсы      |          6 |         4
 Мяч футбольный |         4 |         5
 Кофеварка   |          4 |         5
 Ноутбук     |          4 |         5
 Смартфон    |          4 |         5
 Ракетка теннисная |         2 |         9
 Холодильник  |          2 |         9
(10 rows)

final_work=#
```

## Ежемесячный доход и его доля от общего дохода

Этот запрос вычисляет ежемесячный доход и его долю от общего дохода

```
SELECT
    DATE_TRUNC('month', o.created_at) AS month,
    SUM(o.total_price) AS monthly_revenue,
    SUM(SUM(o.total_price)) OVER () AS total_revenue,
    ROUND((SUM(o.total_price) / SUM(SUM(o.total_price)) OVER ()) * 100, 2) AS revenue_share
FROM orders o
WHERE o.status != 'Отменён'
GROUP BY DATE_TRUNC('month', o.created_at)
ORDER BY month;
```

```
ivanpleskov — psql final_work — 91x21
final_work=# SELECT
    DATE_TRUNC('month', o.created_at) AS month,
    SUM(o.total_price) AS monthly_revenue,
    SUM(SUM(o.total_price)) OVER () AS total_revenue,
    ROUND((SUM(o.total_price) / SUM(SUM(o.total_price)) OVER ()) * 100, 2) AS revenue_share
FROM orders o
WHERE o.status != 'Отменён'
GROUP BY DATE_TRUNC('month', o.created_at)
ORDER BY month;

```

month	monthly_revenue	total_revenue	revenue_share
2024-06-01 00:00:00	2089.95	10813.92	19.33
2024-07-01 00:00:00	699.97	10813.92	6.47
2024-08-01 00:00:00	179.96	10813.92	1.66
2024-09-01 00:00:00	54.85	10813.92	0.51
2024-10-01 00:00:00	1999.98	10813.92	18.49
2024-11-01 00:00:00	159.95	10813.92	1.48
2024-12-01 00:00:00	5629.26	10813.92	52.06

```
(7 rows)
final_work=#
```

## 9 Физическая схема базы данных

SQL-команды для создания таблиц, триггеров, хранимых функций, а также заполнения таблиц данными находятся в файле `file_with_SQL_commands.sql`. Резервная копия базы данных находится в файле `db_dump.sql`.