СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc212842333)

[1 Описание предметной области приложения «Психологический профиль студента» 4](#_Toc212842334)

[1.1 Краткая характеристика объекта автоматизации 4](#_Toc212842335)

[1.2 Определение функционала программного приложения 5](#_Toc212842336)

[2 Разработка базы данных «Аптечная сеть» средствами PostgreSQL 6](#_Toc212842337)

[2.1 Проектирование логической структуры базы данных 6](#_Toc212842338)

[2.2 Проектирование запросов базы данных 10](#_Toc212842339)

[3 Разработка пользовательского интерфейса 35](#_Toc212842340)

[ЗАКЛЮЧЕНИЕ 48](#_Toc212842341)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 49](#_Toc212842342)

# **ВВЕДЕНИЕ**

Современные технологии активно внедряются во все сферы деятельности, включая здравоохранение и фармацевтику. Автоматизация работы аптечных сетей позволяет повысить эффективность учёта лекарственных средств, оптимизировать процессы продажи, хранения и закупок, а также снизить риск ошибок при оформлении заказов и ведении документации. Для реализации подобных решений широко применяются современные системы управления базами данных (СУБД) и веб-фреймворки.

Среди популярных СУБД выделяется PostgreSQL — мощная, надёжная и свободно распространяемая система, обеспечивающая высокую производительность и гибкость при работе с большими объёмами данных. PostgreSQL поддерживает сложные типы данных, транзакции, механизмы целостности и расширяемость, что делает её подходящей для разработки корпоративных приложений.

Для создания клиентского интерфейса используется FastAPI — современный фреймворк на языке Python, ориентированный на разработку быстрых и асинхронных веб-приложений. Он позволяет реализовать удобный и понятный интерфейс взаимодействия пользователя с базой данных, обеспечивая высокую скорость отклика и простоту интеграции с внешними сервисами.

Цель курсовой заключается в разработке программного приложения «Аптечная сеть», включающего базу данных на основе PostgreSQL и веб-интерфейс на FastAPI, предназначенного для автоматизации процессов учёта и обработки информации о лекарственных средствах, поставках и продажах.

Для достижения поставленной цели необходимо реализовать следующие задачи:

– описать предметную область и особенности функционирования аптечной сети;

– разработать и описать структуру базы данных;

– спроектировать объекты базы данных и связи между ними;

– разработать пользовательский интерфейс на основе FastAPI для взаимодействия с данными.

В результате выполнения курсовой будет создано программное решение, демонстрирующее принципы проектирования и реализации информационных систем на основе современных инструментов.

# **1 Описание предметной области приложения «Психологический профиль студента»**

## **1.1 Краткая характеристика объекта автоматизации**

Объект автоматизации — аптечная сеть, объединяющая несколько филиалов, занимающихся продажей лекарственных препаратов и сопутствующих товаров. Работа сети строится вокруг постоянного движения информации: поступление новых партий лекарств, распределение продукции по филиалам, оформление продаж и контроль остатков. Каждая аптека хранит данные о товарах, их форме выпуска, цене, количестве, сроках годности, а также фиксирует операции реализации и возвращения товара.

При традиционном ведении учёта данные часто разрознены: часть информации хранится в бумажных журналах, часть — в независимых таблицах. Отсутствие единого централизованного доступа усложняет поиск нужных сведений и увеличивает риск ошибок. Например, несвоевременное обновление информации о количестве товара может привести к ситуации, когда препарат значится в наличии, хотя фактически он уже был реализован.

В аптечной сети важное место занимает работа с партиями лекарств. Каждая партия имеет срок годности, цену, поставщика и уникальный номер, что позволяет отслеживать происхождение продукции и исключать продажу просроченных препаратов. Дополнительно фиксируется необходимость рецепта для отдельных товаров, что влияет на порядок реализации и взаимодействие с покупателем.

Автоматизация позволяет централизовать управление процессами и отказаться от ручного контроля большинства операций. Данные хранятся в структурированном виде, доступ к ним осуществляется через единую базу, что упрощает анализ продаж и планирование закупок. Это способствует снижению операционных затрат, улучшению качества обслуживания клиентов и формированию актуальной отчётности по каждому филиалу.

## **1.2 Определение функционала программного приложения**

Программное приложение «Аптечная сеть» предназначено для автоматизации ключевых процессов, связанных с учётом лекарственных средств, работой с партиями товаров и оформлением продаж. Основная задача системы — обеспечить удобное взаимодействие сотрудников с данными и сократить количество ручных операций, которые ранее выполнялись в отдельных таблицах или на бумаге.

Функционал приложения строится вокруг нескольких основных сущностей: филиалы аптечной сети, товары, партии, пациенты и продажи. Для каждой группы данных предусмотрены операции добавления, редактирования, просмотра и удаления записей. Благодаря этому достигается полный цикл управления товаром — от поступления новой партии до момента реализации конкретному покупателю.

Важным функциональным элементом является учёт партий лекарственных средств. Для каждой партии сохраняется срок годности, уникальный номер и количество. Это обеспечивает контроль движения товара внутри сети и позволяет отслеживать актуальные остатки по каждому филиалу. При продаже препарата фиксируются данные о партии, из которой был реализован товар, что делает возможным анализ продаж и контроль сроков годности.

Приложение также предоставляет возможность регистрации покупателей и формирования чеков. В момент продажи система рассчитывает итоговую сумму по чеку, учитывая цену товаров и их количество. Такая логика позволяет использовать приложение как инструмент для оперативного обслуживания клиентов и формирования финансовой отчётности.

Для доступа к данным и выполнению операций используется клиентский интерфейс, созданный на FastAPI. С его помощью формируются запросы к базе данных PostgreSQL, обеспечивающие быстрое получение и обновление информации. Централизованное хранение данных делает процессы прозрачными, упорядоченными и удобными для анализа.

# **2 Разработка базы данных «Аптечная сеть» средствами PostgreSQL**

## **2.1 Проектирование логической структуры базы данных**

Для программного приложения «Аптечная сеть» была создана база данных в среде PostgreSQL. Сначала выбирается имя базы данных и параметры кодировки, что гарантирует корректное хранение русскоязычных данных. Создание базы выполняется следующим запросом:

CREATE DATABASE pharmacy\_network WITH OWNER = postgres ENCODING = 'UTF8' LC\_COLLATE = 'ru\_RU.UTF-8' LC\_CTYPE = 'ru\_RU.UTF-8' TEMPLATE = template0;

После этого выполняется подключение к базе командой:

\c pharmacy\_network

и начинается формирование таблиц.

Первая создаваемая таблица отвечает за хранение данных о филиалах аптечной сети. Она содержит идентификатор, наименование и адрес филиала. Идентификатор реализуется через тип SERIAL — это удобный способ автоматически генерировать уникальные значения. Поле name помечено как обязательное (NOT NULL), что исключает появление безымянных филиалов. Код для создания таблицы выглядит так:

CREATE TABLE branches (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

address TEXT

);

Следующая таблица предназначена для хранения сведений о товарах — лекарственных препаратах. Для каждого товара сохраняется наименование, форма выпуска, цена и признак рецептурного отпуска. Цена записывается в формате NUMERIC(10,2), который подходит для финансовых значений, а обязательность поля гарантирует, что товар не будет добавлен без стоимости. Код создания таблицы:

CREATE TABLE products (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

form TEXT,

price NUMERIC(10,2) NOT NULL,

is\_prescription BOOLEAN DEFAULT FALSE

);

Третья таблица — batches — хранит информацию о партиях медикаментов. Здесь указывается, какой товар относится к какой партии, в каком филиале партия находится, срок годности и количество. Поля product\_id и branch\_id являются внешними ключами и обеспечивают ссылочную целостность. Ограничение CHECK (quantity >= 0) предотвращает появление отрицательных остатков товара. Код для создания таблицы:

CREATE TABLE batches (

id SERIAL PRIMARY KEY,

product\_id INT NOT NULL REFERENCES products(id) ON DELETE CASCADE,

branch\_id INT NOT NULL REFERENCES branches(id) ON DELETE CASCADE,

lot\_number TEXT,

expiry\_date DATE,

quantity INT NOT NULL CHECK (quantity >= 0)

);

Далее создаётся таблица для хранения данных о пациентах. Здесь фиксируются ФИО и контактные данные покупателя. Идентификатор создаётся автоматически, а поле full\_name помечено как обязательное. Код создания таблицы:

CREATE TABLE patients (

id SERIAL PRIMARY KEY,

full\_name TEXT NOT NULL,

phone TEXT

);

Для фиксации факта покупки создаётся таблица чеков. В ней сохраняется дата продажи, итоговая сумма и ссылки на филиал и пациента. Дата по умолчанию устанавливается через now(), что упрощает оформление операции. Код таблицы:

CREATE TABLE sales\_invoices (

id SERIAL PRIMARY KEY,

branch\_id INT REFERENCES branches(id),

patient\_id INT REFERENCES patients(id),

sale\_date TIMESTAMP DEFAULT now(),

total\_amount NUMERIC(10,2)

);

Завершающая таблица — позиции продажи. Она фиксирует, какие товары и из каких партий были проданы, а также цену за единицу. Наличие внешних ключей обеспечивает корректную связь с чеком, товаром и партией, а ограничение CHECK (quantity > 0) предотвращает невозможные операции. Код таблицы:

CREATE TABLE sales\_items (

id SERIAL PRIMARY KEY,

sales\_invoice\_id INT REFERENCES sales\_invoices(id) ON DELETE CASCADE,

product\_id INT REFERENCES products(id),

batch\_id INT REFERENCES batches(id),

quantity INT NOT NULL CHECK (quantity > 0),

unit\_price NUMERIC(10,2) NOT NULL

);

Так сформирована логическая структура базы данных. Она отражает реальные процессы работы аптечной сети, связывает данные между собой и предотвращает появление некорректных записей. Благодаря использованию первичных и внешних ключей достигается целостность информации, а база данных остаётся удобной для масштабирования и дальнейшего развития.

Диаграмма базы данных «Аптечная сеть» представлена на рисунке 1.

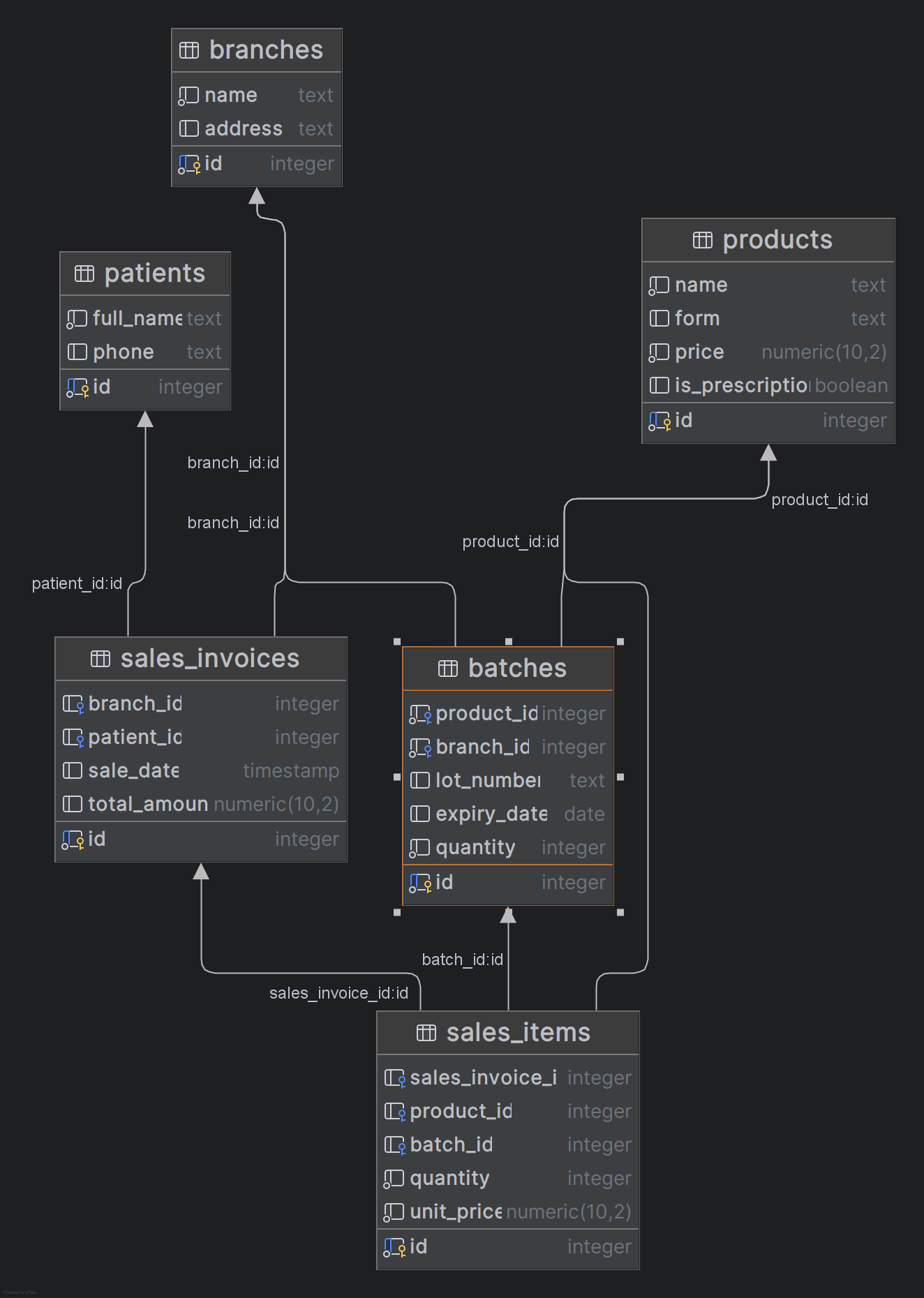


Рисунок 1 — Диаграмма базы данных

Таблица филиалов (branches) связана с таблицей партий товаров (batches) по принципу «один филиал — много партий». Каждая партия относится только к одному филиалу, но в одном филиале может быть несколько партий разных товаров. Такая структура отражает то, что каждое помещение аптеки хранит свои остатки.

Таблица товаров (products) также связана с таблицей партий (batches) по тому же принципу «один-ко-многим». Один товар может поступать несколькими партиями с разным сроком годности и разными ценами от разных поставок. Это делает учёт более точным и позволяет управлять остатками не только по товару, но и по каждой его партии.

Таблица пациентов (patients) связана с таблицей продаж (sales\_invoices), где фиксируются чеки. Один пациент может совершить множество покупок, значит связь также «один-ко-многим». При этом допускается, что чек может быть оформлен без указания пациента, если продажа производится без регистрации покупателя.

Таблица продаж (sales\_invoices) связана с таблицей позиций продажи (sales\_items) по принципу «один чек — много товаров». В одном чеке может быть один или несколько проданных препаратов, каждый из которых хранится как отдельная позиция. В таблице sales\_items каждая позиция указывает, какой именно товар был продан и из какой партии он выбыл.

Таблица sales\_items одновременно связана с products и batches. Это обеспечивает уникальный механизм контроля остатков: товар списывается не просто абстрактно, а с конкретной партии. Благодаря этому сохраняется возможность отслеживания срока годности и происхождения каждого проданного товара.

Наличие внешних ключей гарантирует соблюдение ссылочной целостности: невозможно создать позицию продажи, если нет чека, товара или партии. При удалении записи каскадные действия предотвращают появление «осиротевших» данных.

## **2.2 Проектирование запросов базы данных**

Запрос 1. Нормализация наименований форм выпуска и подсчёт частоты встречаемости

Данный запрос выполняет обработку данных о форме выпуска лекарственных препаратов. В таблице products форма выпуска хранится в виде текстового значения, и разные записи могут содержать одно и то же значение, но с различиями в написании — например, с большой буквы или с дополнительными словами. Для получения более точной статистики используется функция split\_part, которая выделяет только первое слово из значения поля form. Это позволяет нормализовать данные и рассматривать формы выпуска как единые категории: «таблетки», «капсулы», «сироп» и т. д.

Дополнительно применяется функция lower, приводящая текст к нижнему регистру, что устраняет различия, связанные с регистром букв. Далее происходит группировка всех товаров по нормализованной форме выпуска, и для каждой категории подсчитывается количество позиций. Итог сортируется по убыванию, чтобы на первых строках отображались наиболее распространённые формы препаратов.

Запрос:

SELECT lower(split\_part(form, ' ', 1)) AS form\_norm,

COUNT(\*) AS products\_count

FROM products

GROUP BY form\_norm

ORDER BY products\_count DESC NULLS LAST;

В итоге формируется статистика по частоте встречаемости различных форм выпуска препаратов. Полученный результат представлен на рисунке 2.

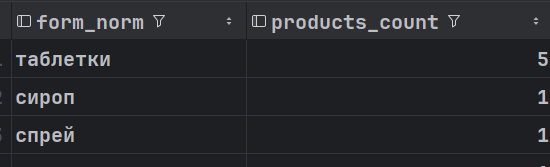


Рисунок 2 Результат

Запрос 2. Поиск пациентов с корректным телефонным номером

Данный запрос используется для выборки пациентов, у которых телефонный номер содержит не менее десяти цифр. В таблице patients телефон может быть записан в разных форматах: с пробелами, скобками, дефисами или другими символами. Чтобы определить реальное количество цифр, применяется функция regexp\_replace. Она удаляет из значения поля phone все символы, которые не являются цифрами, и оставляет только числовую последовательность.

В запросе выполняется проверка длины очищенного номера: если после удаления нецифровых символов остаётся десять или более цифр, такая запись считается валидной, и пациент включается в результат. Дополнительно выводятся идентификатор, ФИО, исходное значение телефона и очищенный номер, что облегчает последующую проверку данных. Сортировка выполняется по имени пациента, чтобы результат был удобным для визуального просмотра.

Используемый запрос:  
SELECT id,  
full\_name,  
phone,  
regexp\_replace(phone, '\D', '', 'g') AS digits\_only  
FROM patients  
WHERE length(regexp\_replace(phone, '\D', '', 'g')) >= 10  
ORDER BY full\_name;

В результате формируется выборка пациентов, чьи телефонные номера содержат достаточное количество цифр и могут считаться корректными для связи. Полученный результат представлен на рисунке 3.

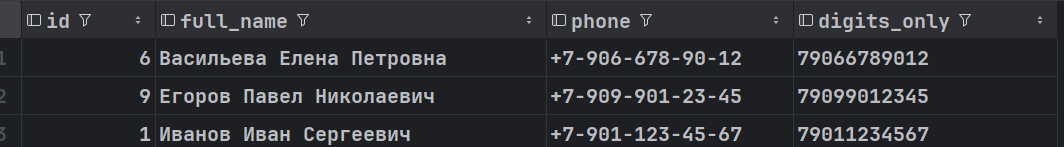


Рисунок 3Результат

Запрос 3. Определение товаров с высоким объёмом продаж

Запрос выполняет анализ продаж лекарственных препаратов на основании данных из таблиц sales\_items и products. Для каждого товара подсчитывается общее количество единиц, проданных через систему. Связь между таблицами осуществляется через поле product\_id, поэтому данные из таблицы позиций продаж объединяются с информацией о наименовании товара. После суммирования количества реализованного товара выбираются только те препараты, по которым объём продаж превышает значение 2. Такой подход позволяет выявить товары, пользующиеся спросом, и оценить динамику продаж для последующего анализа ассортимента.

Запрос:  
SELECT p.name AS product,  
SUM(si.quantity) AS total\_sold  
FROM sales\_items si  
JOIN products p ON p.id = si.product\_id  
GROUP BY p.name  
HAVING SUM(si.quantity) > 2;

В результате формируется список товаров, чья суммарная продажа превысила заданный порог. Полученный результат представлен на рисунке 4.

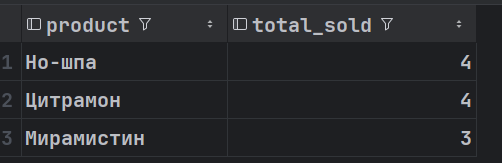


Рисунок 4 Результат

Запрос 4. Расчёт среднего остаточного срока годности по товарам

Запрос выполняет анализ партий лекарственных препаратов, хранящихся в аптечной сети. Цель запроса — определить среднее количество дней до окончания срока годности для каждого товара. Для этого используется связь между таблицей products и batches, где batches содержит информацию о партиях, включая дату истечения срока годности.

Условие b.expiry\_date >= CURRENT\_DATE исключает из расчётов просроченные партии, оставляя только актуальные остатки. Разница между датой окончания срока годности и текущей датой вычисляется через выражение (b.expiry\_date - CURRENT\_DATE). Полученный интервал приводится к целому типу (::int), чтобы отобразить число дней. Далее высчитывается среднее значение по всем партиям конкретного продукта, что позволяет оценить, насколько долго товар может находиться в продаже.

Запрос сгруппирован по идентификатору и названию товара. Сортировка проводится по возрастанию средних значений, благодаря чему в результате первыми отображаются товары с минимальным количеством дней до истечения срока годности. Это делает результат полезным для контроля ассортимента — можно заранее увидеть риск скорого списания непроданных товаров и принять меры для перераспределения партий между филиалами.

Запрос:  
SELECT p.id AS product\_id,  
p.name AS product\_name,  
AVG((b.expiry\_date - CURRENT\_DATE))::int AS avg\_days\_to\_expiry  
FROM products p  
JOIN batches b ON b.product\_id = p.id  
WHERE b.expiry\_date >= CURRENT\_DATE  
GROUP BY p.id, p.name  
ORDER BY avg\_days\_to\_expiry;

В итоге формируется список товаров с указанием среднего количества дней до окончания срока годности по актуальным партиям. Полученный результат представлен на рисунке 5.

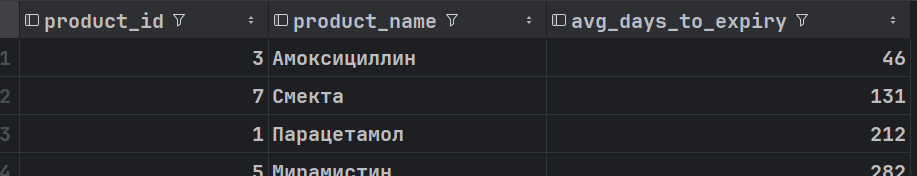


Рисунок 5 Результат

Запрос 5. Анализ активности продаж по часам суток

Запрос предназначен для оценки распределения продаж в течение дня. В таблице sales\_invoices хранится информация о чеках, включая дату и время оформления. Из значения даты извлекается час с помощью функции date\_part. Это позволяет определить, в какой промежуток дня оформляется больше всего продаж.

Далее выполняется объединение с таблицей sales\_items, содержащей позиции продаж. Для каждой позиции доступно количество проданного товара и его цена. Перемножение этого количества на цену даёт сумму по позиции. Затем запрос группирует данные по часам суток. В выборке подсчитывается количество чеков, оформленных в каждый час, и суммарная стоимость проданных позиций. Использование COUNT(DISTINCT si.id) позволяет учитывать именно количество чеков, а не количество позиций в них, что даёт корректное представление о количестве операций продажи.

Запрос:  
SELECT date\_part('hour', si.sale\_date) AS hour\_of\_day,  
COUNT(DISTINCT si.id) AS invoices\_count,  
SUM(it.quantity \* it.unit\_price) AS line\_amount  
FROM sales\_invoices si  
LEFT JOIN sales\_items it ON it.sales\_invoice\_id = si.id  
GROUP BY hour\_of\_day  
ORDER BY hour\_of\_day;

В результате формируется распределение продаж по часам суток с количеством чеков и суммарной выручкой. Полученный результат представлен на рисунке 6.

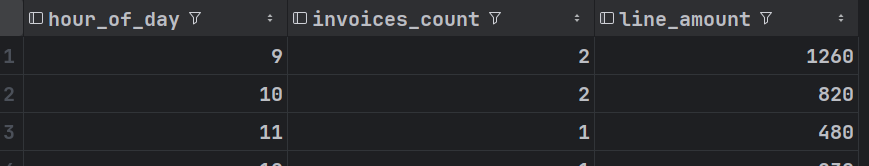


Рисунок 6 Результат

Запрос 6. Определение просроченных остатков по филиалам и товарам

Запрос выполняет анализ партий товаров, срок годности которых уже истёк. В таблице batches хранятся сведения о партиях, включая дату окончания срока годности и количество остатка по каждой из них. Условие b.expiry\_date < CURRENT\_DATE позволяет выбрать только просроченные партии, то есть те, которые больше не могут быть реализованы.

Далее данные связываются с таблицами branches и products, что даёт возможность сопоставить каждую просроченную партию с её филиалом и соответствующим товаром. Суммирование количества просроченных единиц товара для каждой группы позволяет определить общий объём нереализованного остатка по каждому продукту в каждом филиале.

Группировка по названию филиала и товара формирует удобную агрегированную таблицу, в которой отображается итоговое количество просроченного товара. Наличие условия HAVING SUM(b.quantity) > 0 исключает строки, где вычисленное количество равно нулю. Сортировка по названию филиала и товара делает результат структурированным и удобным для анализа.

Запрос:  
SELECT br.name AS branch,  
p.name AS product,  
SUM(b.quantity) AS total\_qty\_expired  
FROM batches b  
JOIN branches br ON br.id = b.branch\_id  
JOIN products p ON p.id = b.product\_id  
WHERE b.expiry\_date < CURRENT\_DATE  
GROUP BY br.name, p.name  
HAVING SUM(b.quantity) > 0  
ORDER BY branch, product;

В результате формируется отчёт о просроченных товарах по каждой аптеке и каждому препарату, что позволяет оперативно выявлять потери и планировать дальнейшие действия. Полученный результат представлен на рисунке 7.

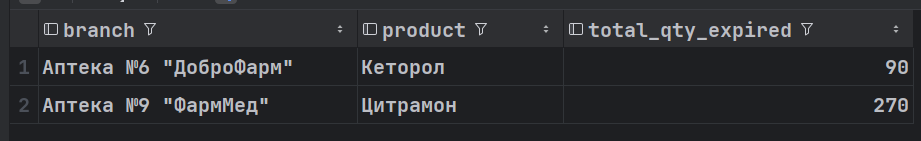


Рисунок 7Результат

Запрос 7. Определение трёх самых продаваемых товаров в каждом филиале

Запрос позволяет определить, какие товары пользуются наибольшим спросом в каждом филиале аптечной сети. Для выполнения задачи используется оконная функция DENSE\_RANK, которая помогает распределить товары по уровням популярности внутри каждого филиала.

На первом этапе, во временной выборке sold, осуществляется объединение таблицы sales\_items с таблицей sales\_invoices. Используется связь через идентификатор продажи, что позволяет определить, какой товар и в каком количестве был продан. Затем данные группируются по филиалу и товару, а сумма количества продаж отображает общий объём реализации каждого препарата в конкретной аптеке.

На втором этапе выборка ranked применяет оконную функцию DENSE\_RANK. Она присваивает товарам ранги внутри каждого филиала в зависимости от объёма продаж. В отличие от обычного ранжирования, DENSE\_RANK не пропускает значения при равенстве результатов: если два товара проданы в одинаковом количестве, они получают одинаковый ранг, а следующий после них ранг остаётся последовательным.

Далее происходит выборка только тех записей, у которых ранг меньше либо равен трём. Это означает, что в итоговый отчёт попадут только три товара с наибольшим количеством продаж в каждом филиале. После ранжирования данные связываются с таблицами branches и products, чтобы вывести названия филиалов и товаров.

Запрос:  
WITH sold AS (SELECT si.branch\_id,  
it.product\_id,  
SUM(it.quantity) AS qty  
FROM sales\_items it  
JOIN sales\_invoices si ON si.id = it.sales\_invoice\_id  
GROUP BY si.branch\_id, it.product\_id),  
ranked AS (SELECT s.\*,  
DENSE\_RANK() OVER (PARTITION BY s.branch\_id ORDER BY s.qty DESC) AS rnk  
FROM sold s)  
SELECT br.name AS branch,  
p.name AS product,  
qty,  
rnk  
FROM ranked r  
JOIN branches br ON br.id = r.branch\_id  
JOIN products p ON p.id = r.product\_id  
WHERE r.rnk <= 3  
ORDER BY branch, rnk, product;

В результате формируется рейтинг самых популярных товаров по каждому филиалу, что позволяет определить лидеров продаж и анализировать спрос. Полученный результат представлен на рисунке 8.

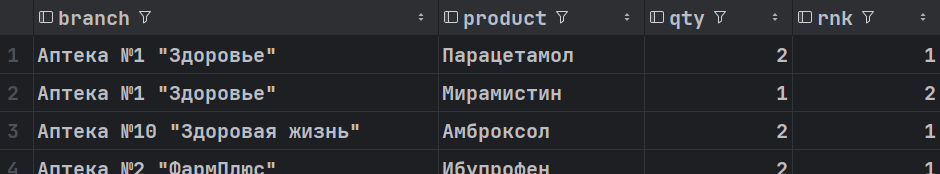


Рисунок 8 Результат

Запрос 8. Скользящая 7-дневная выручка по датам

Запрос формирует динамику выручки с агрегированием по календарным датам и рассчитывает скользящую сумму за последние семь дней. На первом шаге в общей таблице выражений daily собираются дневные итоги: из таблицы продаж sales\_invoices извлекается календарная дата, а через соединение с позициями продаж sales\_items суммируется произведение количества на цену, что даёт дневную выручку. Благодаря группировке по DATE(si.sale\_date) получается по одной строке на каждый день, в который были продажи.

На втором шаге применяется оконная функция SUM по упорядоченным датам. Оконная рамка ROWS BETWEEN 6 PRECEDING AND CURRENT ROW задаёт горизонт из семи последовательных строк, включающих текущий день и шесть предыдущих. Это обеспечивает корректный расчёт «скользящего окна» даже при неравномерных объёмах продаж: каждое значение rolling\_7d\_amount отражает сумму выручки за последние семь календарных дней, присутствующих в выборке. Итоговая выдача содержит дату, дневную выручку и рассчитанную семидневную сумму, отсортированные по возрастанию дат, что удобно для визуализации тренда.

Запрос:  
WITH daily AS (SELECT DATE(si.sale\_date) AS d,  
SUM(it.quantity \* it.unit\_price) AS amount  
FROM sales\_invoices si  
JOIN sales\_items it ON it.sales\_invoice\_id = si.id  
GROUP BY DATE(si.sale\_date))  
SELECT d,  
amount,  
SUM(amount) OVER (  
ORDER BY d  
ROWS BETWEEN 6 PRECEDING AND CURRENT ROW  
) AS rolling\_7d\_amount  
FROM daily  
ORDER BY d;

В результате получается последовательность дат с дневной выручкой и её скользящим 7-дневным суммированием, удобная для оценки динамики и выявления локальных спадов или пиков. Полученный результат представлен на рисунке 9.

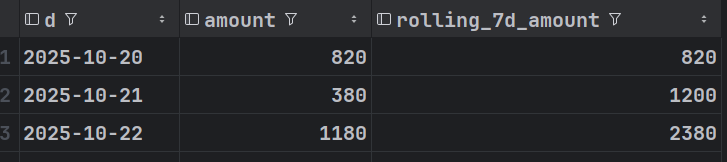


Рисунок 9 Результат

Запрос 9. Доля выручки каждого товара внутри филиала

Запрос формирует структуру выручки по товарам в разрезе каждого филиала. На первом шаге общий табличный выражение branch\_product агрегирует выручку по паре «филиал–товар», суммируя произведение количества на цену из таблицы позиций продаж. Такая агрегация позволяет получить итоговую выручку по каждому продукту в конкретной аптеке без повторов и дублирующих строк. На втором шаге результирующий набор связывается с таблицами branches и products, чтобы вывести человекочитаемые названия филиалов и товаров. Для оценки вклада каждого товара внутри филиала применяется оконная функция: сумма выручки по всем товарам данного филиала вычисляется через SUM(revenue) OVER (PARTITION BY branch\_id), после чего считается процентная доля товара — 100.0 \* revenue / этой суммы. Итоговая выборка сортируется по названию филиала и убыванию выручки товара, что делает структуру наглядной и удобной для анализа.

Запрос:  
WITH branch\_product AS (SELECT si.branch\_id,  
it.product\_id,  
SUM(it.quantity \* it.unit\_price) AS revenue  
FROM sales\_invoices si  
JOIN sales\_items it ON it.sales\_invoice\_id = si.id  
GROUP BY si.branch\_id, it.product\_id)  
SELECT br.name AS branch,  
p.name AS product,  
revenue,  
100.0 \* revenue / SUM(revenue) OVER (PARTITION BY branch\_id) AS revenue\_share\_pct  
FROM branch\_product bp  
JOIN branches br ON br.id = bp.branch\_id  
JOIN products p ON p.id = bp.product\_id  
ORDER BY branch, revenue DESC;

В результате получается распределение выручки по товарам внутри каждого филиала вместе с процентной долей, что позволяет увидеть вклад каждого продукта в общий оборот аптеки. Полученный результат представлен на рисунке 10.

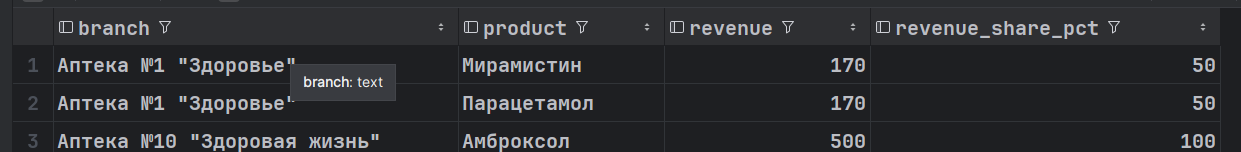


Рисунок 10 Результат

Запрос 10. Построение непрерывного календаря дат с подстановкой нулевой выручки

Запрос формирует сплошной временной ряд по дням от первой до последней даты продаж и подставляет нули там, где операций не было. Такая задача возникает при визуализации динамики: если в какие-то дни продаж нет, график не должен «рваться» — требуется явное значение 0. На первом шаге общее табличное выражение bounds вычисляет минимальную и максимальную даты из sales\_invoices. Затем рекурсивное выражение calendar генерирует последовательность дат от dmin до dmax с шагом один день. Параллельно во временной выборке daily считается дневная выручка как сумма произведений количества на цену по объединению sales\_invoices и sales\_items, сгруппированная по календарной дате. Итоговая выборка строится левым соединением calendar с daily; функция COALESCE заменяет NULL на 0 для дней без продаж, благодаря чему получается непрерывный временной ряд, удобный для построения графиков и расчёта метрик скользящих окон.

Запрос:  
WITH RECURSIVE  
bounds AS (SELECT MIN(DATE(sale\_date)) AS dmin,  
MAX(DATE(sale\_date)) AS dmax  
FROM sales\_invoices),  
calendar(d) AS (SELECT dmin  
FROM bounds  
UNION ALL  
SELECT c.d + 1  
FROM calendar c,  
bounds  
WHERE c.d < bounds.dmax),  
daily AS (SELECT DATE(si.sale\_date) AS d,  
SUM(it.quantity \* it.unit\_price) AS amount  
FROM sales\_invoices si  
JOIN sales\_items it ON it.sales\_invoice\_id = si.id  
GROUP BY DATE(si.sale\_date))  
SELECT c.d, COALESCE(d.amount, 0) AS amount  
FROM calendar c  
LEFT JOIN daily d ON d.d = c.d  
ORDER BY c.d;

В результате формируется непрерывная последовательность дат с дневной выручкой, где отсутствие продаж отражено нулевыми значениями. Полученный результат представлен на рисунке 11.

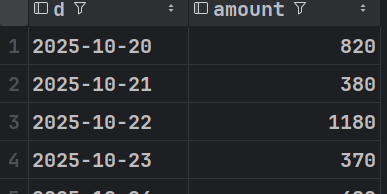


Рисунок 11 Результат

Запрос 11. Прогноз остатка дней до истечения срока годности по партиям на ближайшую неделю

Запрос строит по каждой партии «сетку» дат на ближайшие семь дней и для каждой точки рассчитывает, сколько суток остаётся до даты истечения срока годности. Такой разрез позволяет увидеть траекторию приближения партии к критической отметке и заранее планировать приоритет продаж или перераспределение остатков между филиалами. В общей структуре задействованы два CTE. В horizon с помощью рекурсивного выражения создаётся горизонт сдвигов по дням от 0 до 7 для каждой партии. В grid на основании этого горизонта рассчитываются конкретные календарные точки (day\_point) от текущей даты и вычисляется разница до expiry\_date в сутках. Далее сетка соединяется с batches и products, чтобы вывести идентификатор партии, наименование товара, номер партии и прогнозные значения по дням. Фильтр по g.day\_point <= b.expiry\_date исключает даты после наступления срока годности; дополнительное выражение CASE вместе с GREATEST обнуляет отрицательные значения и оставляет только осмысленные остатки. Итоговая выдача упорядочивается по партии и дате, что удобно для построения календарной диаграммы.

Запрос:  
WITH RECURSIVE  
horizon AS (SELECT b.id AS batch\_id, b.expiry\_date, 0 AS offset\_days  
FROM batches b  
UNION ALL  
SELECT batch\_id, expiry\_date, offset\_days + 1  
FROM horizon  
WHERE offset\_days < 7),  
grid AS (SELECT h.batch\_id,  
(CURRENT\_DATE + (h.offset\_days || ' days')::interval)::date AS day\_point,  
(h.expiry\_date - (CURRENT\_DATE + (h.offset\_days || ' days')::interval)::date)::int AS days\_to\_expiry  
FROM horizon h)  
SELECT b.id AS batch\_id,  
p.name AS product,  
b.lot\_number,  
g.day\_point,  
CASE WHEN g.day\_point <= b.expiry\_date THEN GREATEST(g.days\_to\_expiry, 0) END AS days\_to\_expiry  
FROM grid g  
JOIN batches b ON b.id = g.batch\_id  
JOIN products p ON p.id = b.product\_id  
WHERE g.day\_point <= b.expiry\_date  
ORDER BY batch\_id, day\_point;

В результате получается помесячная сетка по дням на предстоящую неделю для каждой партии с числом оставшихся суток до истечения срока годности. Полученный результат представлен на рисунке 12.

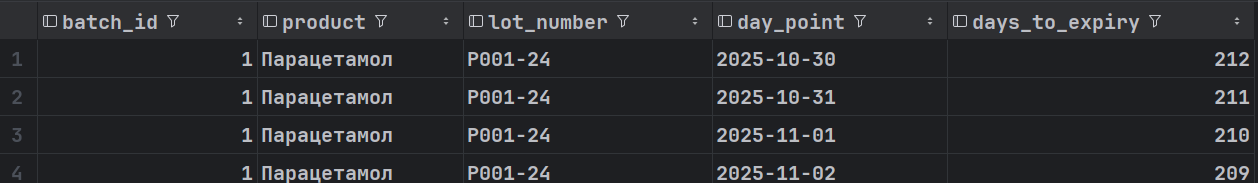


Рисунок 12 Результат

Запрос 12. Подсчёт количества дней доступности товара до истечения срока годности

Запрос рассчитывает, сколько календарных дней товар остаётся «в наличии» с точки зрения срока годности. Логика основана на генерации непрерывного ряда дат для каждой партии от текущей даты до даты истечения её срока годности включительно. В общем табличном выражении dates используется рекурсивная часть, которая последовательно прибавляет по одному дню, пока значение не превысит дату expiry\_date. Такой подход создаёт календарную сетку «партия \* день», где каждая строка означает, что партия формально годна к продаже в конкретный день. Далее данные связываются с таблицей products, после чего агрегируются по наименованию товара: считается количество строк (то есть сумма «годных» дней по всем партиям товара), а также фиксируются минимальная и максимальная даты в получившемся промежутке. В результате получается обобщённая метрика «days\_available», отражающая суммарную доступность товара во времени без учёта фактических остатков; при необходимости контроль количества может быть добавлен отдельным условием. Итог сортируется по убыванию количества дней, что позволяет увидеть товары с максимальным горизонтом годности.

Запрос:  
WITH RECURSIVE dates AS (  
SELECT b.id AS batch\_id,  
b.product\_id,  
b.expiry\_date,  
CURRENT\_DATE::date AS d  
FROM batches b  
UNION ALL  
SELECT dates.batch\_id,  
dates.product\_id,  
dates.expiry\_date,  
(dates.d + INTERVAL '1 day')::date  
FROM dates  
WHERE (dates.d + INTERVAL '1 day')::date <= dates.expiry\_date)  
SELECT p.name AS product,  
COUNT(\*) AS days\_available,  
MIN(d) AS start\_date,  
MAX(d) AS end\_date  
FROM dates  
JOIN products p ON p.id = dates.product\_id  
GROUP BY p.name  
ORDER BY days\_available DESC;

В результате формируется агрегированная таблица с суммарным числом «годных» дней для каждого товара и границами периода. Полученный результат представлен на рисунке 13.

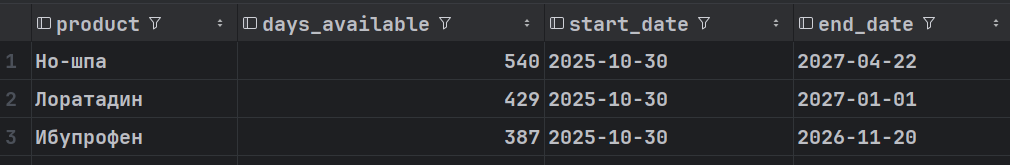


Рисунок 13 Результат

Запрос 13. Определение 90-го перцентиля и медианной цены по формам выпуска

Запрос выполняет анализ ценовой категории товаров, сгруппированных по форме выпуска. В таблице sales\_items хранится цена товара в момент продажи, а в таблице products — информация о форме выпуска. Формирование группировки начинается с обработки поля form: сначала выполняется trim, убирающий лишние пробелы, затем NULLIF превращает пустое значение строки в NULL, и COALESCE заменяет NULL на фразу «— неизвестно —». Это позволяет корректно сгруппировать данные, даже если форма выпуска не была указана.

Для анализа цен используются упорядоченные агрегатные функции. PERCENTILE\_CONT(0.90) WITHIN GROUP (ORDER BY unit\_price) вычисляет 90-й перцентиль, то есть цену, выше которой располагаются только 10% значений. Данный показатель отражает верхние границы цен внутри своей группы и помогает определить наиболее дорогие товары среди конкретной формы выпуска. PERCENTILE\_CONT(0.50) WITHIN GROUP (ORDER BY unit\_price) вычисляет медиану, то есть значение, разделяющее множество цен пополам. В отличие от среднего, медиана не искажается резко выбивающимися значениями и даёт более объективное понимание ценового уровня.

Запрос:  
SELECT COALESCE(NULLIF(trim(form), ''), '— неизвестно —') AS form\_group,  
PERCENTILE\_CONT(0.90) WITHIN GROUP (ORDER BY unit\_price) AS p90\_unit\_price,  
PERCENTILE\_CONT(0.50) WITHIN GROUP (ORDER BY unit\_price) AS median\_unit\_price  
FROM sales\_items it  
LEFT JOIN products p ON p.id = it.product\_id  
GROUP BY form\_group  
ORDER BY form\_group;

В результате формируется таблица с формами выпуска и ключевыми статистическими показателями стоимости товаров: медианой и 90-м перцентилем. Полученный результат представлен на рисунке 14.

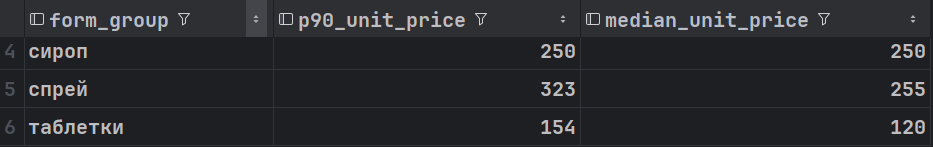


Рисунок 14 Результат

Запрос 14. Когортный анализ пациентов: размер когорты, активность, выручка и удержание по месяцам

Запрос предназначен для построения когортной аналитики по пациентам: для каждого месяца первой покупки определяется размер когорты, затем по последовательным месяцам рассчитывается число активных клиентов, суммарная выручка и процент удержания. На первом этапе формируется надёжная выручка по чеку без опоры на поле total\_amount, путём суммирования количества и цены позиций. Далее отбираются все покупки с известным пациентом, вычисляются дата и месяц продажи. Для каждого пациента определяется «нулевой» месяц — месяц первой покупки, который служит когортной меткой. Затем для всех последующих покупок высчитывается смещение в месяцах относительно когорты (months\_since\_first), что позволяет построить временную траекторию активности. Параллельно считается размер когорты как число уникальных пациентов в нулевом месяце. Итоговый запрос агрегирует данные по паре «месяц когорты — смещение», возвращая активных клиентов, выручку, исходный размер когорты и долю удержания как отношение активных к размеру когорты, округлённую до десятых. Такой результат удобен для тепловых карт и оценки динамики LTV/ретеншна.

Запрос:  
WITH  
-- выручка по чеку (если invoices.total\_amount ненадёжно)  
invoice\_revenue AS (  
SELECT  
si.id AS sales\_invoice\_id,  
SUM(it.quantity \* it.unit\_price) AS amount  
FROM sales\_items it  
JOIN sales\_invoices si ON si.id = it.sales\_invoice\_id  
GROUP BY si.id  
),  
-- все покупки пациентов  
purchases AS (  
SELECT  
inv.patient\_id,  
DATE(inv.sale\_date) AS d,  
date\_trunc('month', inv.sale\_date)::date AS d\_month,  
ir.amount  
FROM sales\_invoices inv  
JOIN invoice\_revenue ir ON ir.sales\_invoice\_id = inv.id  
WHERE inv.patient\_id IS NOT NULL  
),  
-- первый месяц покупки (когорта)  
first\_purchase AS (  
SELECT  
patient\_id,  
MIN(d\_month) AS cohort\_month  
FROM purchases  
GROUP BY patient\_id  
),  
-- факты с отступом по месяцам от первой покупки  
facts AS (  
SELECT  
p.patient\_id,  
fp.cohort\_month,  
p.d\_month,  
p.amount,  
(EXTRACT(YEAR FROM p.d\_month) - EXTRACT(YEAR FROM fp.cohort\_month)) \* 12

(EXTRACT(MONTH FROM p.d\_month) - EXTRACT(MONTH FROM fp.cohort\_month)) AS months\_since\_first  
FROM purchases p  
JOIN first\_purchase fp ON fp.patient\_id = p.patient\_id  
),  
-- размер когорты (число уникальных пациентов в нулевом месяце)  
cohort\_sizes AS (  
SELECT  
cohort\_month,  
COUNT(DISTINCT patient\_id) AS cohort\_size  
FROM facts  
WHERE months\_since\_first = 0  
GROUP BY cohort\_month  
)  
SELECT  
TO\_CHAR(f.cohort\_month, 'YYYY-MM') AS cohort\_month,  
f.months\_since\_first::int AS month\_offset,  
COUNT(DISTINCT f.patient\_id) AS active\_customers,  
SUM(f.amount) AS revenue,  
cs.cohort\_size,  
ROUND(100.0 \* COUNT(DISTINCT f.patient\_id) / NULLIF(cs.cohort\_size, 0), 1) AS retention\_pct  
FROM facts f  
JOIN cohort\_sizes cs USING (cohort\_month)  
GROUP BY f.cohort\_month, f.months\_since\_first, cs.cohort\_size  
ORDER BY f.cohort\_month, f.months\_since\_first;

В результате получается когортная таблица, показывающая размер когорты, число активных пациентов, их выручку и процент удержания по месяцам относительно первой покупки. Полученный результат представлен на рисунке 15. 

Рисунок 15 Результат

Запрос 15. Помесячный дашборд по филиалам за последние 3 месяца: выручка, средний чек, уникальные пациенты, доля Rx и топ-товар

Запрос формирует управленческий срез по каждому филиалу за текущий и два предыдущих календарных месяца. В параметрическом блоке вычисляются границы окна и генерируется список месяцев через generate\_series. Далее собираются строки чеков с позициями в указанном диапазоне: фиксируется филиал, месяц продажи, товар, суммарная сумма строки, количество и признак рецептурности. Из этих строк строится сумма по чеку, чтобы корректно посчитать средний чек как среднее по суммам чеков, а не по позициям. На уровне агрегирования «филиал × месяц» вычисляются ключевые метрики: выручка, число чеков, средний чек, количество уникальных пациентов и доля рецептурной выручки (отношение суммы по Rx-позициям к общей сумме). Параллельно через оконную функцию DENSE\_RANK определяется топ-1 товар по выручке в каждом филиале и месяце; при равенстве выручек применяется детерминирующее упорядочивание по product\_id. Для полноты отчёта создаётся сетка «все филиалы × все месяцы», к которой левыми соединениями подмешиваются рассчитанные показатели и топ-товар; отсутствующие данные заполняются нулями с помощью COALESCE. Итоговый набор отсортирован по названию филиала и возрастанию месяца, что удобно для табличной подачи и визуализации.

Запрос:  
WITH  
-- 1) Параметры и список месяцев (последние 3 календарных месяца, включая текущий)  
params AS (  
SELECT  
date\_trunc('month', CURRENT\_DATE)::date AS this\_month,  
(date\_trunc('month', CURRENT\_DATE) - INTERVAL '2 months')::date AS start\_month  
),  
months AS (  
SELECT g::date AS month\_start  
FROM params p  
CROSS JOIN generate\_series(p.start\_month, p.this\_month, INTERVAL '1 month') AS g  
),  
-- 2) Линии чеков в нужном диапазоне месяцев  
invoice\_lines AS (  
SELECT  
si.id AS sales\_invoice\_id,  
si.branch\_id,  
si.patient\_id,  
date\_trunc('month', si.sale\_date)::date AS month\_start,  
it.product\_id,  
(it.quantity \* it.unit\_price) AS line\_amount,  
it.quantity,  
p.is\_prescription  
FROM sales\_items it  
JOIN sales\_invoices si ON si.id = it.sales\_invoice\_id  
JOIN products p ON p.id = it.product\_id  
WHERE date\_trunc('month', si.sale\_date)::date BETWEEN  
(SELECT start\_month FROM params) AND (SELECT this\_month FROM params)  
),  
-- 3) Сумма по чеку (для среднего чека)  
check\_amounts AS (  
SELECT  
il.sales\_invoice\_id,  
il.branch\_id,  
il.month\_start,  
SUM(il.line\_amount) AS invoice\_amount  
FROM invoice\_lines il  
GROUP BY il.sales\_invoice\_id, il.branch\_id, il.month\_start  
),  
-- 4) Агрегации по филиалу × месяцу  
branch\_month AS (  
SELECT  
il.branch\_id,  
il.month\_start,  
SUM(il.line\_amount) AS revenue,  
COUNT(DISTINCT il.sales\_invoice\_id) AS invoices\_count,  
AVG(ca.invoice\_amount) AS avg\_ticket,  
COUNT(DISTINCT il.patient\_id) AS unique\_patients,  
100.0 \* SUM(CASE WHEN il.is\_prescription THEN il.line\_amount ELSE 0 END)  
/ NULLIF(SUM(il.line\_amount), 0) AS rx\_share\_pct  
FROM invoice\_lines il  
JOIN check\_amounts ca  
ON ca.sales\_invoice\_id = il.sales\_invoice\_id  
AND ca.branch\_id = il.branch\_id  
AND ca.month\_start = il.month\_start  
GROUP BY il.branch\_id, il.month\_start  
),  
-- 5) Топ-1 продукт по выручке в филиале в месяц (через окно)  
branch\_month\_top1 AS (  
SELECT branch\_id, month\_start, product\_id, product\_revenue  
FROM (  
SELECT  
il.branch\_id,  
il.month\_start,  
il.product\_id,  
SUM(il.line\_amount) AS product\_revenue,  
DENSE\_RANK() OVER (  
PARTITION BY il.branch\_id, il.month\_start  
ORDER BY SUM(il.line\_amount) DESC, il.product\_id  
) AS rnk  
FROM invoice\_lines il  
GROUP BY il.branch\_id, il.month\_start, il.product\_id  
) r  
WHERE r.rnk = 1  
),  
-- 6) Сетка "все филиалы × все месяцы", чтобы показать и нули  
grid AS (  
SELECT b.id AS branch\_id, b.name AS branch\_name, m.month\_start  
FROM branches b  
CROSS JOIN months m  
)  
-- 7) Финал  
SELECT  
g.branch\_name AS branch,  
TO\_CHAR(g.month\_start, 'YYYY-MM') AS month,  
COALESCE(bm.revenue, 0)::numeric(12,2) AS revenue,  
COALESCE(bm.invoices\_count, 0) AS invoices\_count,  
COALESCE(bm.avg\_ticket, 0)::numeric(10,2) AS avg\_ticket,  
COALESCE(bm.unique\_patients, 0) AS unique\_patients,  
COALESCE(bm.rx\_share\_pct, 0)::numeric(5,2) AS rx\_share\_pct,  
p.name AS top\_product,  
COALESCE(t1.product\_revenue, 0)::numeric(12,2) AS top\_product\_revenue  
FROM grid g  
LEFT JOIN branch\_month bm  
ON bm.branch\_id = g.branch\_id  
AND bm.month\_start = g.month\_start  
LEFT JOIN branch\_month\_top1 t1  
ON t1.branch\_id = g.branch\_id  
AND t1.month\_start = g.month\_start  
LEFT JOIN products p  
ON p.id = t1.product\_id  
ORDER BY g.branch\_name, g.month\_start;

В результате получается сводная таблица по каждому филиалу и месяцу с ключевыми показателями и лидером по выручке; нулевые периоды также отражены. Полученный результат представлен на рисунке 16.

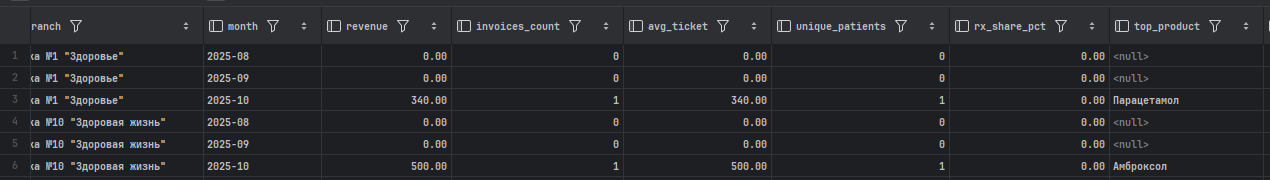


Рисунок 16 Результат

# **3 Разработка пользовательского интерфейса**

Веб-слой реализован на FastAPI и работает поверх асинхронного SQLAlchemy с драйвером asyncpg. Приложение запускается с заголовком Pharmacy Analytics API версии 1.0.0. Подключение к БД задаётся через переменную окружения DATABASE\_URL; при её отсутствии используется строка по умолчанию: postgresql+asyncpg://postgres:postgres@localhost:5432/pharmacy. Движок создаётся вызовом create\_async\_engine(DATABASE\_URL, future=True, echo=False). Для унифицированного доступа к данным предусмотрена вспомогательная корутина run\_query(sql, params=None), которая открывает соединение, выполняет текстовый SQL (text(sql)) и возвращает список словарей; при ошибке выбрасывается HTTPException(500, detail=…).

Запуск сервиса осуществляется командой uvicorn: uvicorn.run("main:app", host="0.0.0.0", port=8000, reload=True). Для проверки доступности добавлен технический маршрут /health, возвращающий {"status": "ok"}.

Ниже приведены ключевые аналитические маршруты. Для каждого указано назначение и ожидаемый результат. Кусочки кода даны в том виде, как они присутствуют в модуле, чтобы их можно было быстро сопоставить с реализацией. Для визуальных материалов зарезервированы ссылки на рисунки.

Маршрут /analytics/forms-breakdown вычисляет частоты форм выпуска по первому слову и в нижнем регистре. Используется SQL с lower(split\_part(form,' ',1)) и группировкой; ответы возвращаются в виде массива объектов {form\_norm, products\_count}. Результат представлен на рисунке 17.

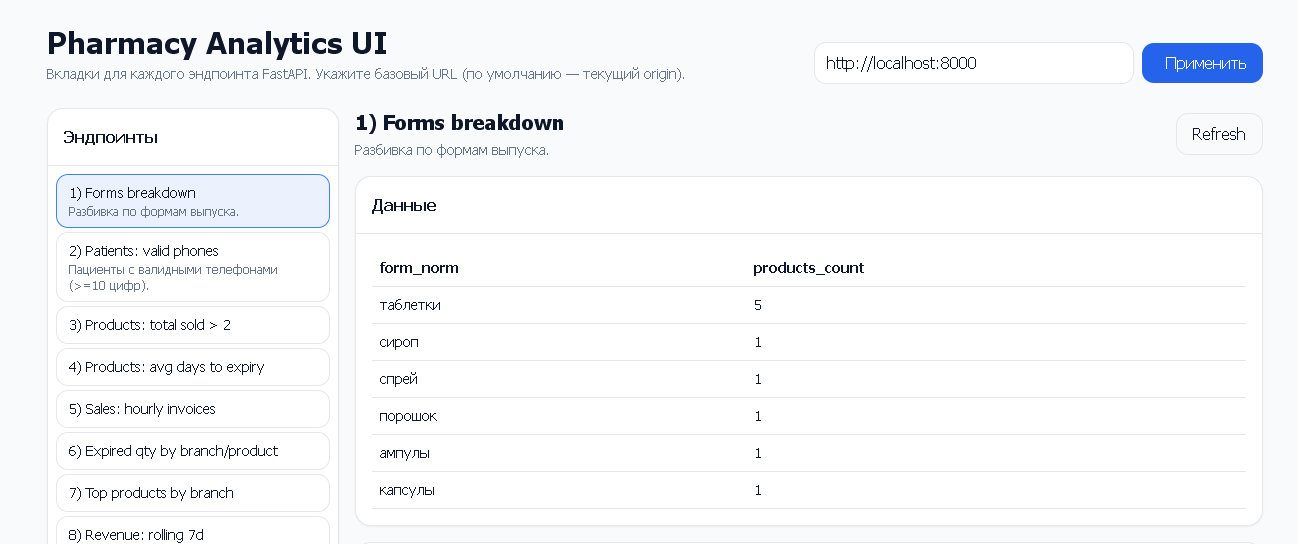


Рисунок 17 BreakDown

Маршрут /analytics/patients/valid-phones выделяет покупателей с «длинными» телефонами. Внутри запроса применяется regexp\_replace(phone, '\D','', 'g') и фильтр длины ≥ 10; дополнительно возвращается очищенная версия digits\_only. Результат представлен на рисунке 18.

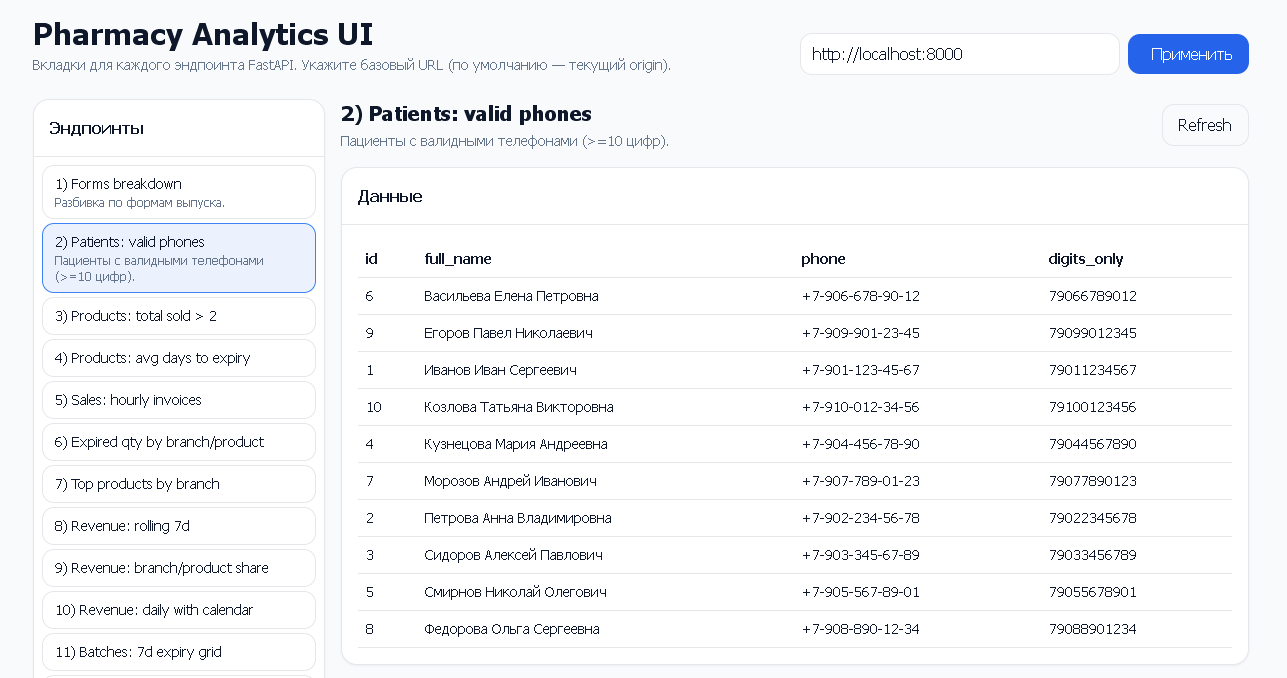


Рисунок 18 Валидные номера

Маршрут /analytics/products/total-sold-gt2 агрегирует количество продаж по товарам и отбирает позиции, где SUM(quantity) > 2. Применяется объединение sales\_items ↔ products и HAVING; отдаётся список лидеров по спросу. Результат представлен на рисунке 19.

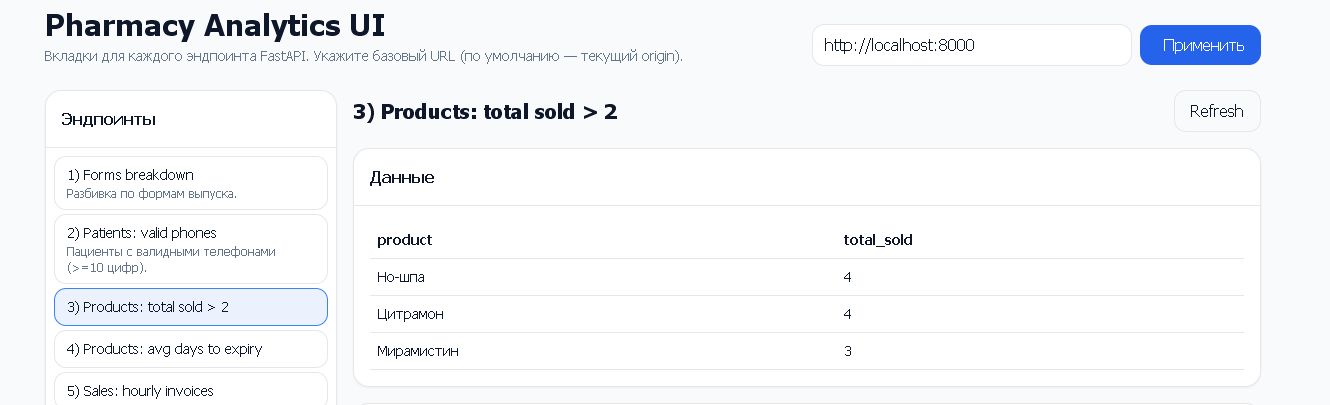


Рисунок 19Общие продажи

Маршрут /analytics/products/avg-days-to-expiry оценивает средний остаточный срок годности по товарам только для непросроченных партий. Вычисление строится на AVG((b.expiry\_date - CURRENT\_DATE))::int с фильтром expiry\_date >= CURRENT\_DATE. Результат представлен на рисунке 20.

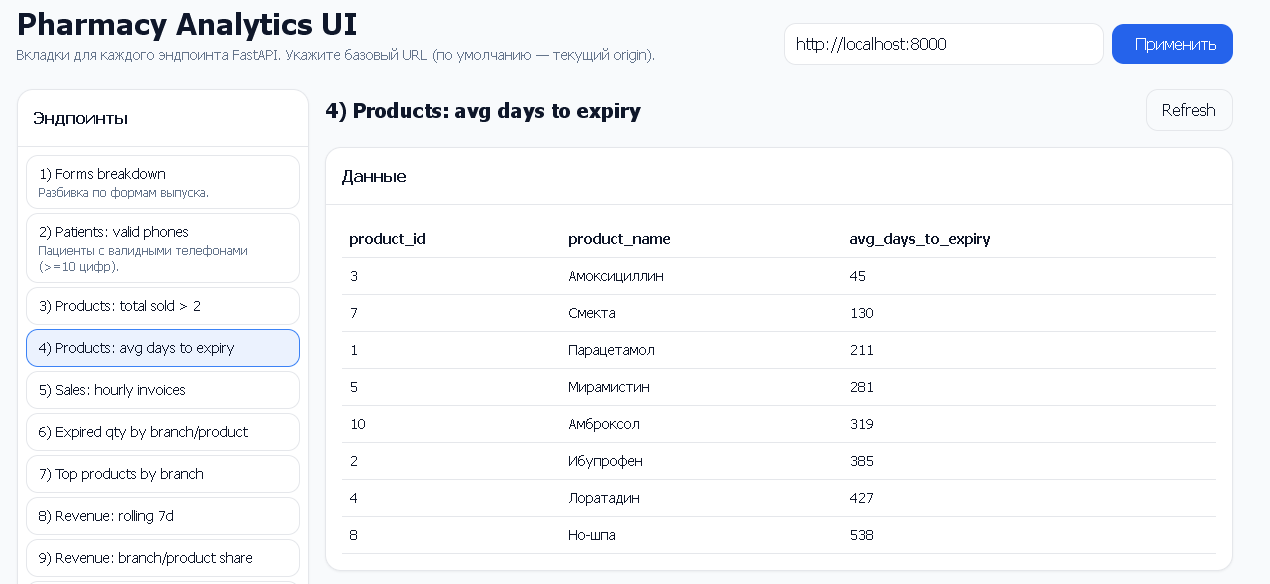


Рисунок 20 Дней до истечения срока

Маршрут /analytics/sales/hourly-invoices показывает активность по часам суток: COUNT(DISTINCT чеков) и сумму позиций per hour. Часы извлекаются через date\_part('hour', sale\_date), соединение с sales\_items — левое для учёта чеков без позиций. Результат представлен на рисунке 21.

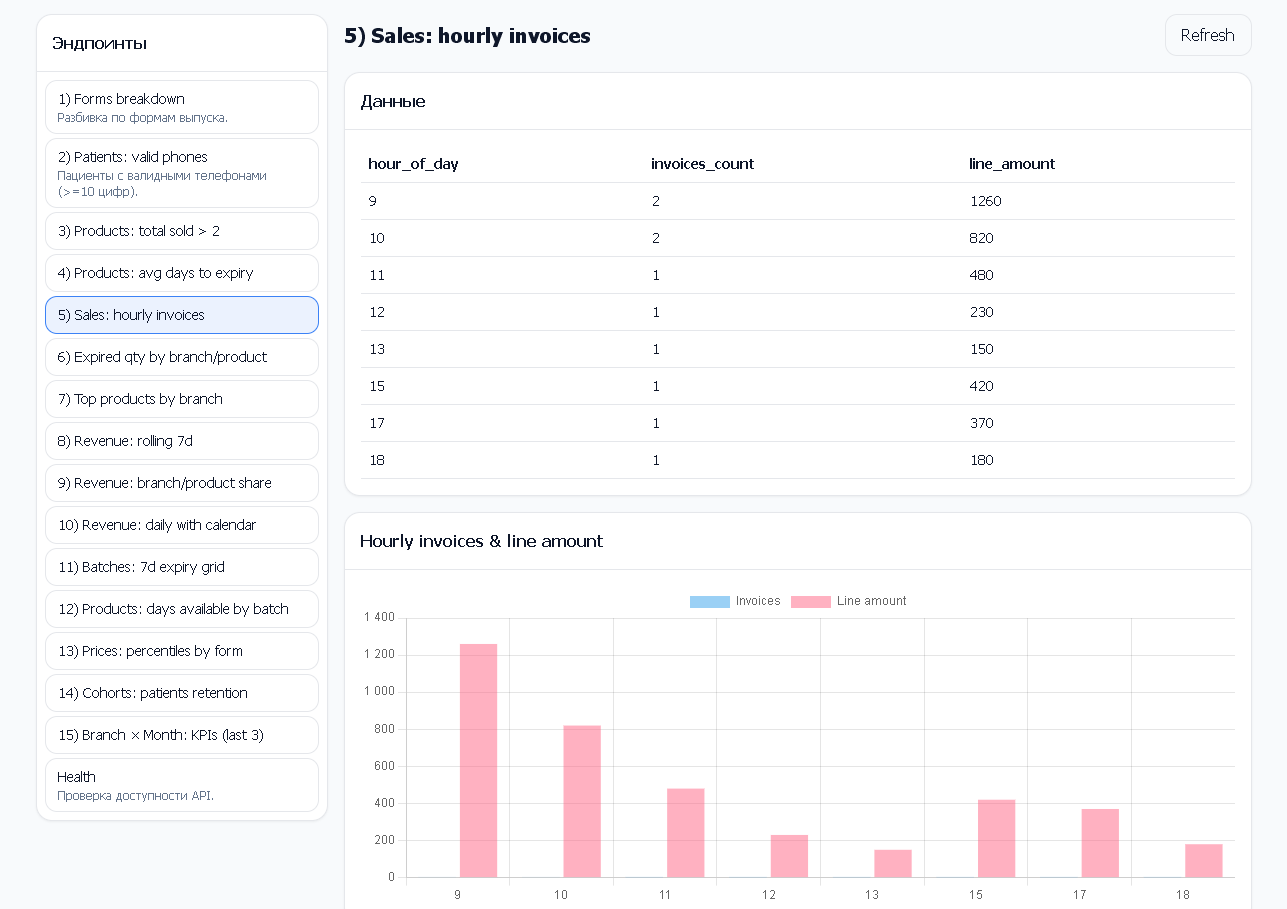


Рисунок 21 Часовые продажи

Маршрут /analytics/expired/qty-by-branch-product формирует реестр просроченных остатков по филиалам и товарам. Отбор по expiry\_date < CURRENT\_DATE, группировка по (branch, product), условие HAVING SUM(quantity) > 0. Результат представлен на рисунке 22.

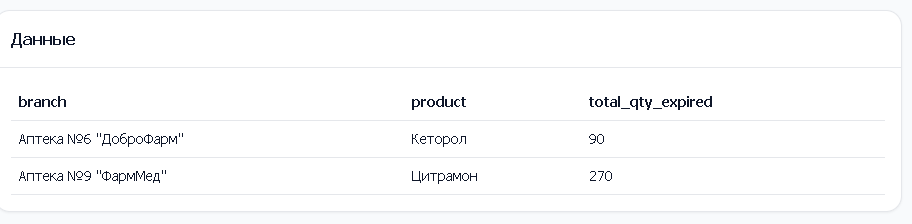


Рисунок 22 Реестр остатков

Маршрут /analytics/top-products-by-branch определяет топ-3 товара по количеству в каждом филиале. В CTE sold суммируется qty, во втором CTE ranked применяется DENSE\_RANK() OVER (PARTITION BY branch ORDER BY qty DESC); фильтр rnk <= 3 возвращает рейтинг. Результат представлен на рисунке 23.

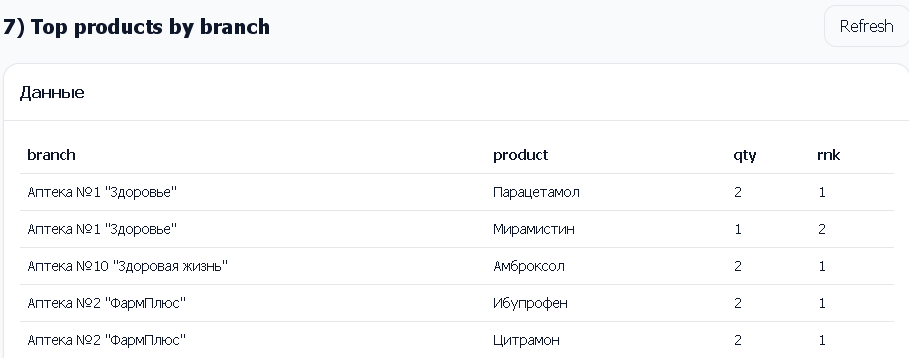


Рисунок 23 Топ продуктов

Маршрут /analytics/revenue/daily-rolling-7d собирает дневную выручку и считает скользящую сумму за 7 дней. В CTE daily агрегируется amount по дате, затем SUM(amount) OVER (ORDER BY d ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) формирует rolling\_7d\_amount. Результат представлен на рисунке 24.

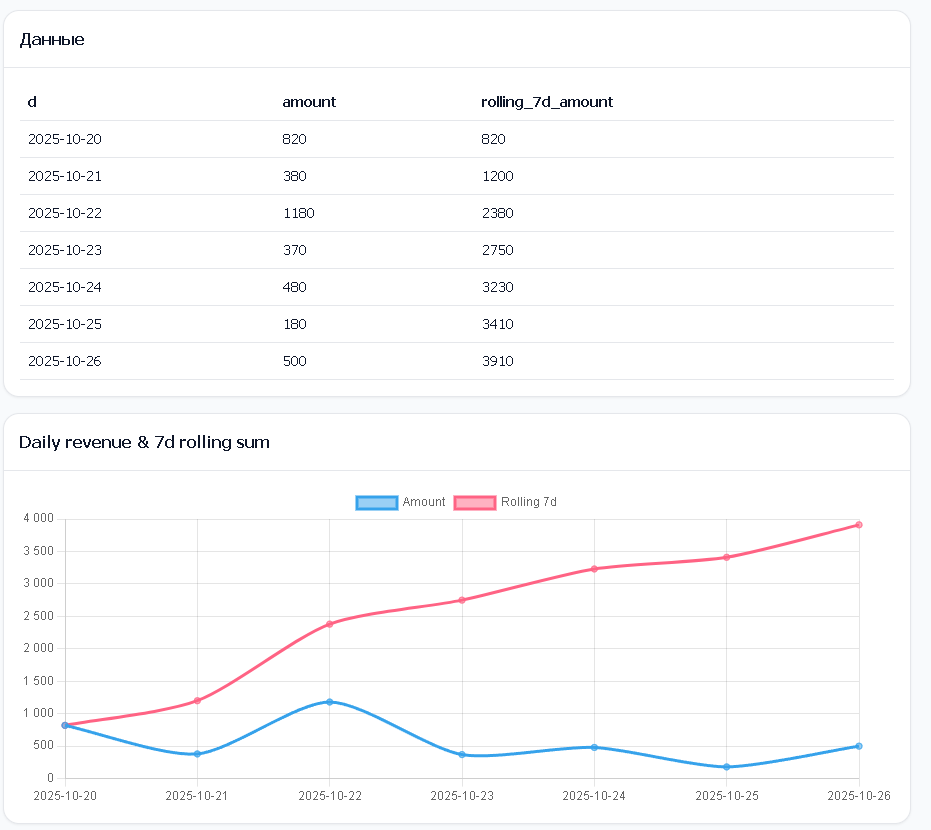


Рисунок 24 Результат

Маршрут /analytics/revenue/branch-product-share рассчитывает вклад каждого товара в выручку филиала. В CTE branch\_product получается revenue на пару (branch\_id, product\_id), далее 100.0 \* revenue / SUM(revenue) OVER (PARTITION BY branch\_id) выдаёт процентную долю. Результат представлен на рисунке 25.

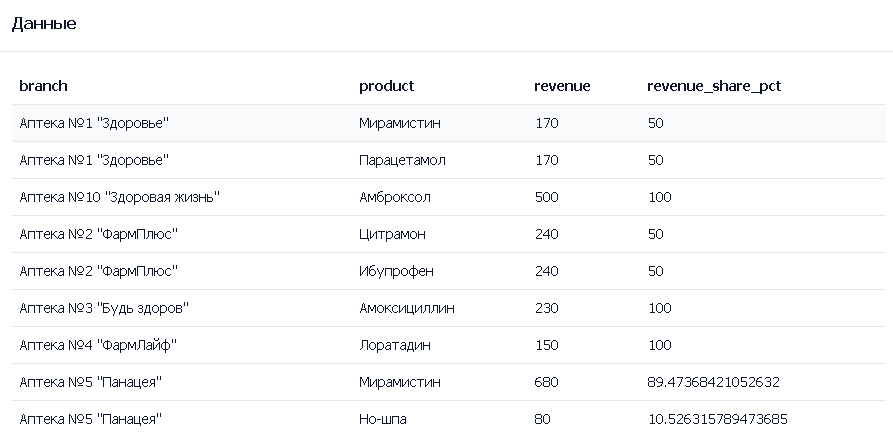


Рисунок 25 Вклад товара

Маршрут /analytics/revenue/daily-with-calendar строит непрерывный календарь между минимальной и максимальной датами продаж. Рекурсивный CTE calendar генерирует диапазон дат, левое соединение с дневной выручкой подставляет нули через COALESCE. Результат представлен на рисунке 26.

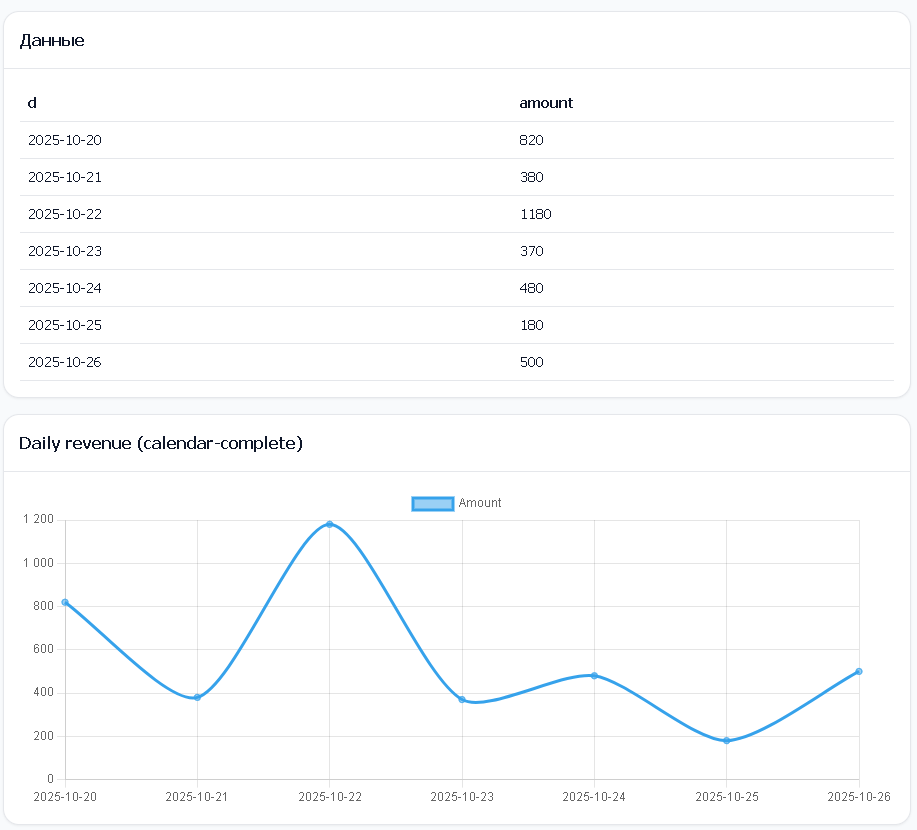


Рисунок 26 Дневная выручка

Маршрут /analytics/batches/7d-expiry-grid прогнозирует остаток дней до истечения срока по партиям на ближайшую неделю. Рекурсивный horizon создаёт смещения 0…7, затем рассчитывается day\_point и days\_to\_expiry; отрицательные значения обнуляются через GREATEST. Результат представлен на рисунке 27.

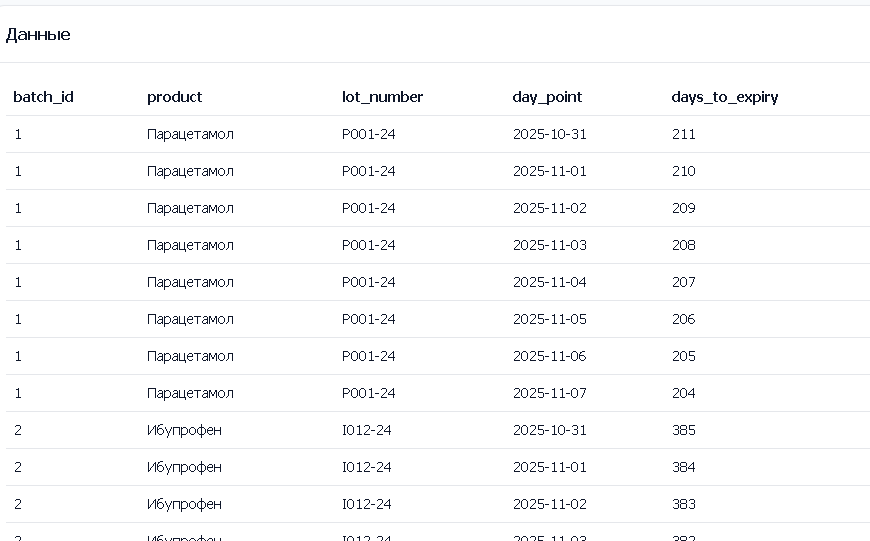


Рисунок 27 Результат

Маршрут /analytics/products/days-available-by-batch подсчитывает количество календарных дней «доступности» товара до expiry по всем партиям. Рекурсивная генерация дат ведётся от CURRENT\_DATE до expiry\_date включительно, далее агрегирование по имени продукта с MIN/MAX периода. Результат представлен на рисунке 28.

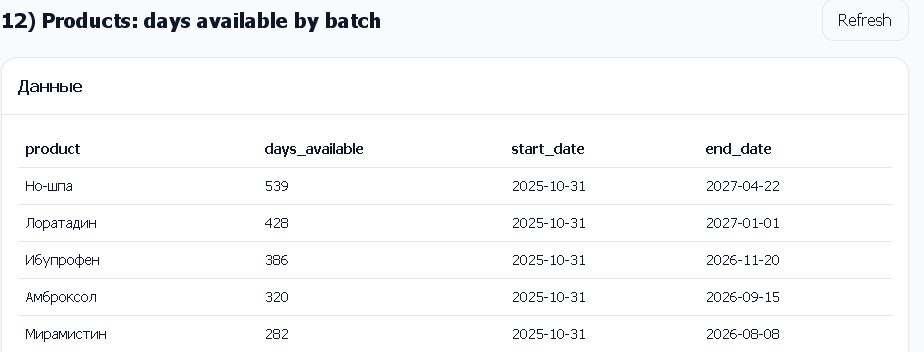


Рисунок 28Дней доступности

Маршрут /analytics/prices/percentiles-by-form вычисляет ценовые ориентиры по формам выпуска: медиану и 90-й перцентиль. Поле form предварительно нормализуется выражением COALESCE(NULLIF(trim(form), ''), '— неизвестно —'); перцентили считаются через PERCENTILE\_CONT WITHIN GROUP (ORDER BY unit\_price). Результат представлен на рисунке 29.

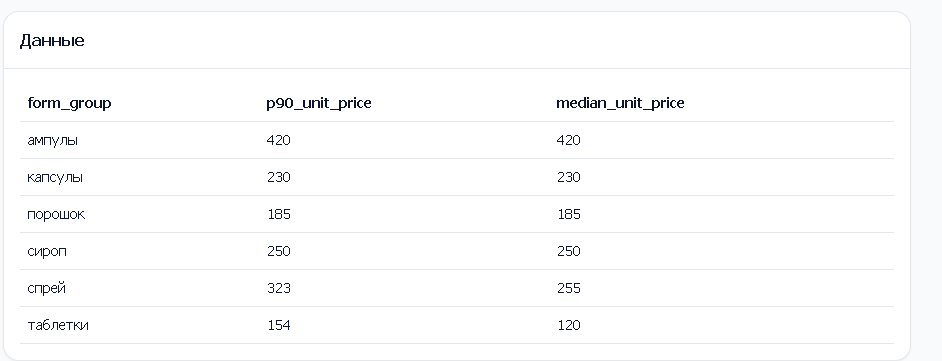


Рисунок 29 Медианы и 90-й перцентиль

Маршрут /analytics/cohorts/patients-retention реализует когортную таблицу удержания пациентов по месяцам. Выручка по чеку считаётся надёжно в invoice\_revenue, затем определяется cohort\_month, вычисляется смещение months\_since\_first, формируется размер когорты и метрики active\_customers, revenue, retention\_pct. Результат представлен на рисунке 30.

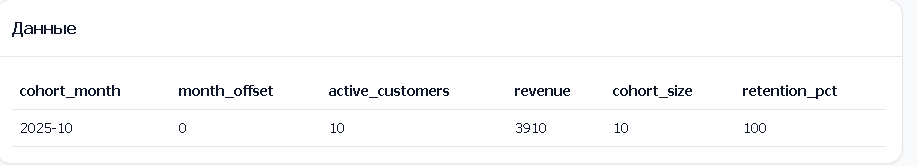


Рисунок 30 Результат

Маршрут /analytics/branch-month/kpis-last-3-months создаёт дашборд «филиал × месяц» за текущий и два предыдущих месяца: выручка, количество чеков, средний чек, уникальные пациенты, доля Rx, топ-товар и его выручка. Сетка всех сочетаний строится в grid, отсутствующие данные заполняются нулями через COALESCE. Результат представлен на рисунке 31.

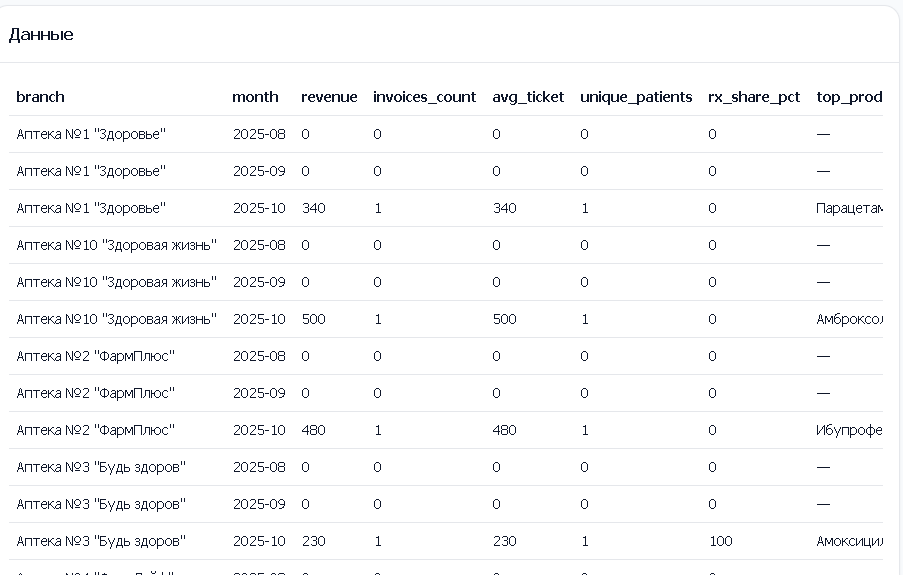


Рисунок 31Дешбоард

Каждый маршрут возвращает JSON-массив словарей, удобный для прямой отрисовки в веб-клиенте или для построения диаграмм в внешнем инструменте. Для простого ручного вызова подойдёт HTTP-клиент или curl; пример обращения: GET /analytics/forms-breakdown. Подключение к реальной БД задаётся через переменную окружения DATABASE\_URL; при развёртывании на сервере параметр указывается в системной конфигурации, что освобождает код от «жёстких» секретов. Обработка ошибок централизована в run\_query: любые исключения из драйвера БД транслируются в статус 500 с описанием причины, вследствие этого упрощается диагностика и журналирование.

Итогом раздела является готовый набор REST-эндпоинтов для аналитики аптечной сети, покрывающий нормализацию справочников, продажи и запасы, временные ряды, когорты и KPI-дашборды.

Для визуализации результатов разработан лёгкий фронтенд на чистом HTML + Tailwind CSS + Chart.js. Интерфейс запускается как статическая страница и взаимодействует с FastAPI по HTTP, что упрощает развёртывание: API может работать на любом хосте/порте, а UI — открываться локально в браузере. Вверху страницы размещено поле для базового URL API (input c id="baseUrl") и кнопка применения настроек (id="applyBtn"); при изменении адреса запросы отправляются на новый origin без перезагрузки страницы. В левой колонке выводится список вкладок с эндпоинтами; конфигурация задаётся в массиве ENDPOINTS, где для каждого маршрута определены ключ, заголовок, путь и краткое описание.

Центральная область содержит три блока: «Данные», «Графики» и «Raw JSON». После выбора вкладки фронтенд делает fetch по адресу baseUrl + path, обрабатывает ответ и одновременно строит таблицу и, при необходимости, диаграмму. Универсальный рендеринг таблицы реализован по принципу авто-схемы: сначала собирается объединённый набор ключей всех строк, затем формируется thead и tbody, причём каждое значение приводится к строке функцией formatCell, а потенциально опасные символы экранируются через escapeHtml — этим предотвращаются XSS при показе произвольного текста из БД. Блок ошибок (id="errorBox") включается, если ответ неуспешен; в «Данные» в таком случае выводится «Нет данных», а «Raw JSON» показывает дефис.

Для визуализации задействован Chart.js. Реализованы целевые сценарии: для ключа hourly-invoices строится столбчатая диаграмма по часам с двумя рядами — количество чеков и сумма позиций; для rolling-7d — линейный график дневной выручки и её 7-дневной скользящей суммы; для daily-with-calendar — линейный график непрерывной дневной выручки с подставленными нулями. Генерация графиков инкапсулирована в renderCharts, при переключении вкладок предыдущие экземпляры диаграмм корректно уничтожаются (destroy), что предотвращает утечки и повторные рендеры поверх старых canvas.

Навигация по вкладкам работает без роутинга: список кнопок создаётся динамически, активная вкладка подсвечивается классом tab-active. Таб контента пересобирается функцией renderPanel: в шапке показываются заголовок и краткая подпись эндпоинта, рядом — кнопка обновления (id="refreshBtn"), которая инициирует повторный fetch. Для адаптивной вёрстки применены утилиты Tailwind (grid, rounded-2xl, shadow-sm и пр.), за счёт чего интерфейс корректно ужимается на малых экранах и раскладывается в две колонки на широких.

Особенности интеграции учтены на уровне CORS: если API расположен на другом домене/порту, фронтенд корректно работает при включённом CORSMiddleware на стороне FastAPI; подсказка добавлена в футере. Набор колонок в таблице не «зашит» под конкретные схемы — вывод происходит по фактическим ключам из ответа, поэтому любой новый отчёт, добавленный в FastAPI, автоматически отображается без доработки фронтенда. В результате формируется единая панель для всех аналитических маршрутов: вкладки соответствуют эндпоинтам /analytics/…, данные показываются в таблице и в «Raw JSON», а для временных рядов и агрегатов — строятся наглядные графики. Такая связка даёт законченную пользовательскую часть: от параметризации адреса API до интерактивного просмотра и первичного анализа показателей прямо в браузере.

# **ЗАКЛЮЧЕНИЕ**

В ходе курсового проекта было создано программное приложение «Аптечная сеть», включающее базу данных на PostgreSQL, аналитический backend на FastAPI и веб-интерфейс для визуализации данных. База данных спроектирована с учётом логических связей предметной области: филиалы, товары, партии, пациенты, продажи и позиции продаж. Такая структура позволила обеспечить целостность данных и корректное выполнение аналитических запросов. Для каждой таблицы заданы первичные и внешние ключи, ограничения и проверки, что устранило дублирование информации и исключило невозможные значения, например продажи несуществующей партии.

Создан набор SQL-запросов, позволяющий получать полезные аналитические результаты: динамику продаж по времени, показатели выручки, частоту форм выпуска, просроченные партии, прогноз истечения срока годности, распределение выручки по товарам и филиалам, ценовые перцентили и когортный анализ пациентов. Backend реализует доступ к этим запросам через REST-эндпоинты, возвращая данные в формате JSON. Это позволило отделить слой данных от пользовательской части и обеспечить гибкость в использовании.

На основе FastAPI-эндпоинтов создан веб-интерфейс. Через него данные отображаются в таблицах и диаграммах, что делает процесс анализа более наглядным. Графики строятся с помощью Chart.js, а элементы интерфейса оформлены средствами Tailwind CSS.

В итоге сформировано законченное решение, позволяющее работать с данными аптечной сети не только на уровне хранения, но и в плане аналитики и визуализации.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. FastAPI: официальная документация [Электронный ресурс] – Режим доступа: https://fastapi.tiangolo.com/docs/
2. PostgreSQL: официальная документация СУБД PostgreSQL [Электронный ресурс] – Режим доступа: https://www.postgresql.org/docs/
3. Chart.js: документация по визуализации данных [Электронный ресурс] – Режим доступа: https://www.chartjs.org/docs/
4. Tailwind CSS: документация по использованию CSS-фреймворка [Электронный ресурс] – Режим доступа: https://tailwindcss.com/docs
5. PEP 8 – Style Guide for Python Code // Python.org [Электронный ресурс] – Режим доступа: https://peps.python.org/pep-0008/
6. SQLAlchemy: документация по работе с базами данных в Python [Электронный ресурс] – Режим доступа: https://docs.sqlalchemy.org/