МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 6-05-0612-01 Программная инженерия (профилизация Программное обеспечение информационных технологий)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Реализация базы данных планировщика питания с использованием технологии Oracle Mobile Server»

Выполнил студент Филипюк Илья Андреевич
(Ф.И.О.)

Руководитель работы преп. Нистюк О.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Зав. кафедрой ст. преп. Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2025

Содержание

Введение

На сегодняшний день здоровый образ жизни и контроль питания становятся все более популярными среди пользователей разных возрастов. Люди все активнее следят за своим рационом, количеством потребляемых калорий и динамикой веса, используя мобильные приложения и онлайн-сервисы для планирования питания. В условиях обработки персональных данных пользователей, таких как вес, возраст, пол, особенности рациона и цели по здоровью, критически важно обеспечить безопасное хранение информации и разграничение прав доступа к данным.

Цель данной работы заключается в проектировании и разработке реляционной базы данных для планировщика питания, предназначенной для хранения, обработки и анализа данных о пользователях, их рационе, дневниках питания, калориях и физических показателях. Особое внимание уделено реализации механизма **Row Level Security**, который позволяет ограничивать доступ пользователей к своим личным данным, а также применению **Security Definer** для безопасного выполнения хранимых процедур с повышенными привилегиями. База данных служит фундаментом для приложения, обеспечивая целостность, согласованность и безопасность всех операций с информацией о пользователях и их рационе.

В качестве системы управления базами данных выбрана PostgreSQL, так как она предоставляет широкие возможности для реализации сложной бизнес-логики на уровне БД, поддерживает хранимые процедуры, функции, триггеры и транзакции, а также обеспечивает высокий уровень безопасности и гибкость управления доступом. Применение технологий **Row Level Security** и **Security Definer** позволяет эффективно разграничивать права пользователей, предотвращать несанкционированный доступ к персональной информации и минимизировать вероятность ошибок при работе с данными.

Реализация базы данных с использованием указанных технологий обеспечит надежное управление персональными данными, ускорит выполнение операций и создаст безопасную основу для дальнейшего развития приложения, включая возможность интеграции с внешними сервисами для анализа питания и контроля здоровья.

1 Постановка задачи

1.1 Обзор аналогичных решений

Одним из первых и достаточно важных этапов для создания качественного программного продукта является анализ уже существующих на рынке аналогичных решений. Данный этап помогает выделить ключевые характеристики, принципы работы и функционал подобных приложений, а также выявить их преимущества и недостатки, оценить подходы к ведению дневников питания, расчету калорий, мониторингу веса и методы защиты персональных данных пользователей. Полученные данные позволяют учесть лучшие практики и ошибки конкурентов при разработке собственного проекта, чтобы создать конкурентоспособное и безопасное приложение для контроля питания.

1.1.1 Аналог MyFitnessPal

Первым рассматриваемым решением будет сервис «MyFitnessPal». На рисунке 1.1 представлена его главная страница.

Рисунок 1.1 – Приложение MyFitnessPal

На главном экране приложения MyFitnessPal пользователь получает централизованный обзор своего дневного прогресса: сводку по потребленным и оставшимся калориям, баланс макронутриентов (белки, жиры, углеводы), а также список всех съеденных продуктов и выполненных упражнений за день. Здесь же

представлены быстрые ссылки на добавление еды, воды, тренировок и просмотр полного дневника.

Представлен следующий ключевой функционал:
–возможность авторизации через существующий аккаунт Google, Facebook или Apple, а также через электронную почту;
–создание и ведение пищевого дневника с возможностью добавления продуктов и блюд по приемам пищи (завтрак, обед, ужин, перекусы);
–обширная база данных продуктов питания с возможностью сканирования штрих-кодов для быстрого добавления;
–ведение дневника тренировок с привязкой к расходу калорий;
– отслеживание потребления воды и других личных метрик (вес, замеры тела, шаги);
– постановка индивидуальных целей по калориям и макронутриентам, весу и физической активности.
– поиск по базе данных продуктов, блюд, ресторанов и готовых блюд из супермаркетов.

Проведя анализ внешнего вида приложения, мы можем вывести следующие предположения о сущностях базы данных:

–таблица с пользователями содержит информацию с учётными данными, целями и персонализированными настройками;

–таблица с записями содержит данные за конкретную дату, отображающие действия пользователя за день;

–таблица с продуктами содержит данные о блюдах с информацией о содержащихся в них калориях, белках, жирах, углеводах, размере порций и отдельно – сохранённые пользователем блюда и рецепты;

–таблица с упражнениями содержит записи о ежедневной активности пользователя, конкретном её виде, продолжительности и иных показателях;

–таблица с целями содержит установленные пользователем для себя цели по калориям и весу.

В итоге можно сказать, что ключевой особенностью данного приложения является легковесная и понятная структура, позволяющая выделить его как оптимальный и лёгкий в освоении инструмент для пользователей, не имеющих особого опыта, но желающий быстро и просто начать следить за своим питанием, физической активностью и показателями здоровья.

1.1.2 Аналог YAZIO

Следующим аналогом для обзора будет приложение YAZIO, которое, в сравнении с прошлым аналогичным решением предлагает расширенный функционал отслеживания питания и гидратации, а также гораздо больше интерактивного взаимодействия с пользователем. Главные окна данного приложения представлены на рисунке 1.2

Рисунок 1.2 – Приложение YAZIO

Функционал обоих решений схож, но в YAZIO степень интерактивности и графической репрезентации данных гораздо выше, чем в предыдущем аналоге. Так на главном экране приложения YAZIO пользователь видит интуитивную сводку дня: круговую диаграмму, наглядно показывающую прогресс по калориям, а также кольца выполнения целей по белкам, жирам и углеводам. Ниже расположен лента-дневник с записями о еде, воде и упражнениях, отсортированная по приемам пищи. Интерфейс сфокусирован на визуальной простоте и мотивации.

В приложении представлен следующий ключевой функционал:

–процесс калибровки и адаптивные цели. При запуске приложение проводит пользователя через опрос (цели, вес, рост, активность, пищевые предпочтения) и на основе этого автоматически рассчитывает персональный план по калориям и макронутриентам;

–расширенный функционал для быстрого добавления продуктов: не только сканер штрих-кода, но и возможность сфотографировать продукт для его распознавания (премиум-функция);

–возможность планировать и добавлять еду не только на текущий, но и на будущие дни, что помогает сформировать пищевые привычки;

–в приложении представлена обширная библиотека рецептов (включая раздел "Бюджетные") с пошаговыми инструкциями и возможностью автоматического добавления ингредиентов в дневник.

Структура данных данного аналога является более продуманной, что является несомненным достоинством данного решения.

Подводя итоги по данному аналогу, можно сказать, что это решение является более подробным, с достаточно хорошо продуманной структурой и наполнением базы данных. Стоит учесть это в собственном решении.

1.2 Разработка функциональных требований

Функциональные требования базы данных определяют, каким образом система должна обеспечивать хранение, обработку и доступ к данным, связанным с отслеживанием потреблённых калорий и расчётом питания. Основная цель разрабатываемой базы данных – предоставление надежной структуры для хранения информации о пользователях, блюдах, меню, потреблённых калориях, весе и прогрессе пользователя.

На основе этого можем выделить следующие требования:

–хранение информации о пользователях (идентификаторы, имена, пароли, лимиты калорий);

–хранение информации о блюдах (идентификаторы, названия, калорийность, размер порции, содержание БЖУ);

–хранение информации о прогрессе пользователя (потреблённые блюда, количество потреблённых калорий, изменения в массе тела);

–взаимодействие с внешним приложением, предоставляющим удобный интерфейс для работы с вышеописанными данными.

1.3 Выводы по разделу

Были рассмотрены и проанализированы аналогичные решения в единой области изучения на основе которых были сформулированы основные функциональные требования к будущей базе данных.

Результатом работы в данном разделе стали четко сформулированные требования к будущей базе данных.

2 Проектирование базы данных

2.1 Описание моделей взаимодействия сущностей базы данных

Перед тем, как приступать к разработке базы данных нужно продумать ее структуру, определить, какие таблицы будут входить в базу данных, состав каждой таблицы, а также какие типы данных и ограничения целостности будет целесообразно использовать для каждой из них.

Для наглядности данная структура может быть представлена в виде диаграммы базы данных, с помощь которой можно увидеть какие таблицы будут присутствовать в базе данных и по каким полям они будут связаны между собой.

Диаграмма базы данных представлена на рисунке 2.1.

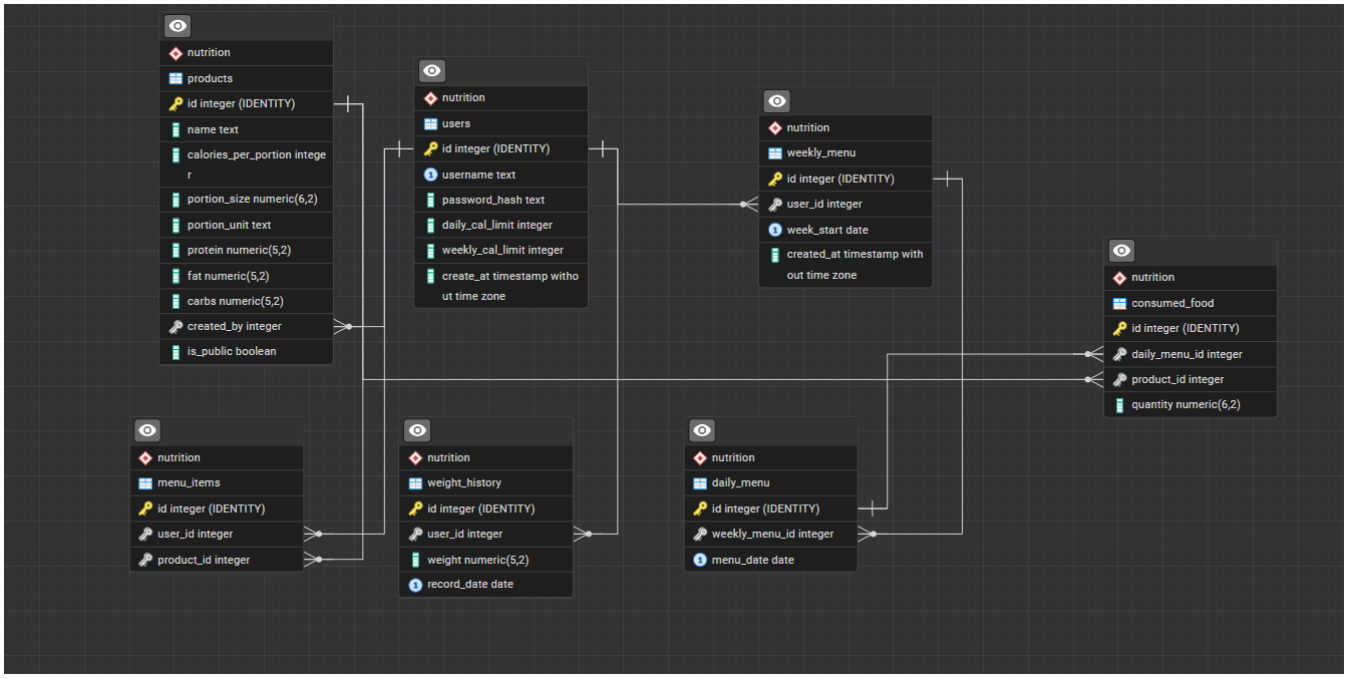

Рисунок 2.1 – Диаграмма базы данных

Одна из основных связей, которая была использована, это связь «один ко многим», которая позволяет отобразить отношение одной записи в первой таблице ко многим записям во второй таблице. Данная связь является достаточно распространенным явлением, так как она позволяет отобразить большинство реальных отношений, встречающихся на практике.

Также была использована и связь «один к одному», которая позволяет отобразить отношение сущностей, при котором одной записи первой таблицы соответствует только одна запись из второй таблицы. Данный вид связи не является достаточно распространенным, однако достаточно хорошо описывает, например, связь личной информации о пользователях и их учетных.

2.2 Варианты использования

Определение вариантов использования будущей базы данных позволяет четко определить требуемый для разработки функционал. Использование ролей позволяет лучше представить какие возможности будут у пользователя в зависимости от его привилегий. Для лучшего восприятия будем использовать UML диаграммы вариантов использования. На рисунке 2.2 представлена такая диаграмма, описывающая проектируемую базу данных.

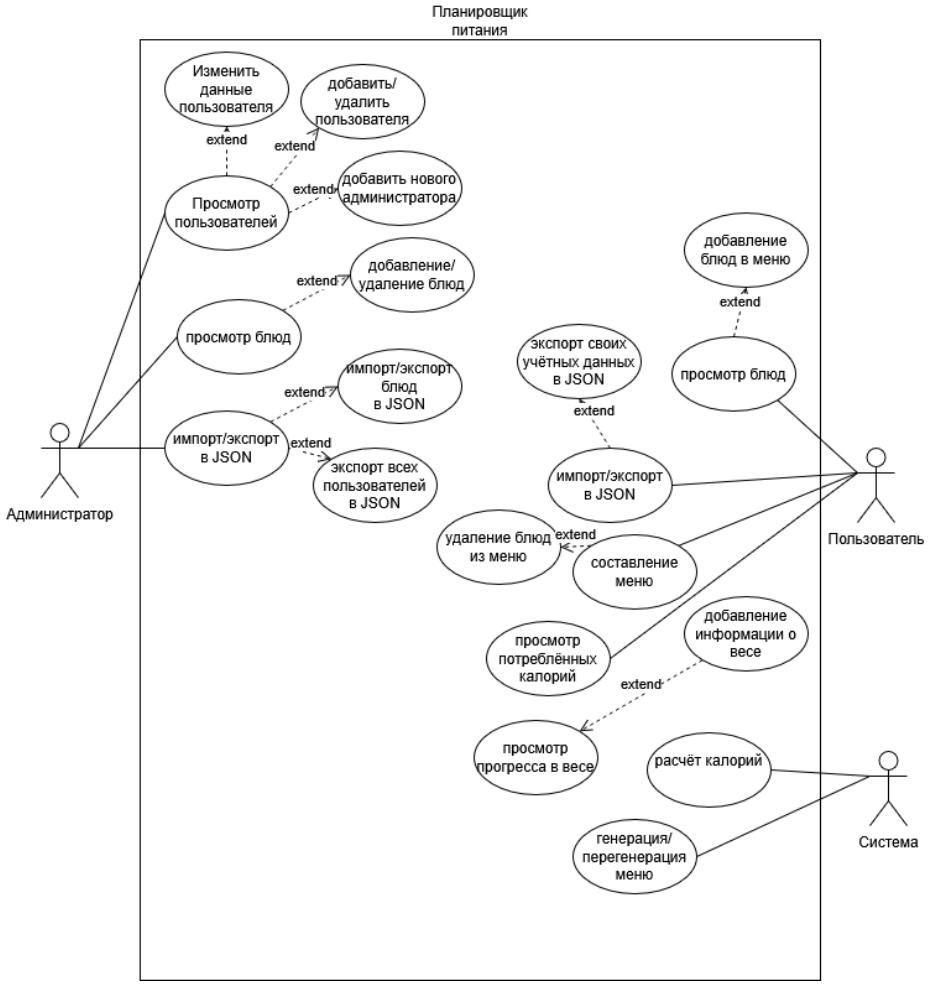

Рисунок 2.2 – Диаграмма вариантов использования

В проектируемой системе выделены 3 актёра: администратор, пользователь и система. В начале взаимодействия с системой, после прохождения процедуры регистрации и авторизации пользователь приобретает соответствующую роль и получает соответствующие права и возможности для работы с системой.

Роль администратора имеет права на управление и взаимодействие основными сущностями базы данных: блюдами и пользователями, а также экспорт и импорт в JSON всех блюд, доступных пользователям и данных о всех пользователях системы.

Пользователь, в свою очередь, наделён базовыми правами и может просматривать блюда, добавлять их себе в меню, рассчитывать для себя меню на день и на неделю, а также следить за калориями и изменениями в массе тела. Также

он может экспортировать данные своей учётной записи и изменений в массе тела в JSON.

Также можно отдельно выделить роль системы, которая проводит действия по подсчёту агрегируемых значений нужных для результата, который запросил пользователь с помощью процедур, не связанных с этими значениями напрямую.

2.3 Описание таблиц базы данных

На основе анализа аналогов и собранных данных, было разработано 7 таблиц: USERS, PRODUCTS, DAILY_MENU, MENU_ITEMS, CONSUMED_FOOD, WEEKLY_MENU, WEIGHT_HISTORY.

Проанализируем каждую из них.

Таблица USERS хранит информацию о пользователях системы. Структура данной таблицы представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы USERS

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор пользователя
username	TEXT	UNIQUE NOT NULL	Имя пользователя в системе
password_hash	TEXT	NOT NULL	Хэшированное значение пароля пользователя
daily_cal_limit	INT	NOT NULL CHECL(daily_cal_limit>0)	Лимит калорий в день для конкретного пользователя
weekly_cal_limit	INT	NOT NULL CHECK(weekly_cal_limit>0)	Лимит калорий на неделю для конкретного пользователя
created_at	TIMESTAMP	DEFAULT now()	Время создания учётной записи пользователя

Таблица PRODUCTS хранит информацию о блюдах, которые пользователь может добавить в своё меню. Структура данной таблицы представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы PRODUCTS

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор блюда

Окончание таблицы 2.2

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
name	TEXT	NOT NULL	Название блюда
calories_per_portion	INT	NOT NULL CHECK(calories_per_portion>0)	Количество калорий в одной порции блюда
portion_size	NUMERIC(6, 2)	NOT NULL CHECK CHECK(portion_size>0)	Размер одной порции блюда
portion_unit	TEXT	NOT NULL CHECK(portion_unit in(‘g’,’kg’,’ml’,’l’,’pcs’,’г’,’кг’,’мл’,’л’,’шт’)	Единица измерения одной порции конкретного блюда
protein	NUMERIC(5, 2)	NOT NULL CHECK(protein>=0)	Количество граммов протеина в одной порции
fat	NUMERIC(5, 2)	NOT NULL CHECK(protein>=0)	Количество граммов жиров в одной порции
carbs	NUMERIC(5, 2)	NOT NULL CHECK(carbs>=0)	Количество граммов углеводов в одной порции
created_at	TIMESTAMP	NOT NULL FOREIGN KEY	Время создания блюда
is_public	BOOLEAN	DEFAULT FALSE	Идентификатор доступности блюда для всех пользователей

Таблица DAILY_MENU содержит информацию о меню на день для конкретного пользователя. Структура данной таблицы представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы DAILY_MENU

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор меню
weekly_menu_id	INT	FOREIGN KEY	Идентификатор соответствующего меню на неделю
menu_date	DATE	NOT NULL UNIQUE	Дата для которой создаётся меню

Таблица MENU_ITEMS хранит информацию о конкретных блюдах для конкретного меню. Структура данной таблицы представлена в таблицу 2.4.

Таблица 2.4 – Структура таблицы MENU_ITEMS

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор блюда в меню
user_id	INT	FOREIGN_KEY	Идентификатор соответствующего пользователя
product_id	INT	FOREIGN_KEY	Идентификатор соответствующего продукта

Таблица CONSUMED_FOOD содержит информацию о блюдах, потреблённых пользователем. Структура данной таблицы представлена в таблицу 2.5

Таблица 2.5 – Структура таблицы CONSUMED_FOOD

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор потреблённого блюда
daily_menu_id	INT	NOT NULL FOREIGN KEY	Идентификатор соответствующего меню на день
product_id	INT	NOT NULL FOREIGN KEY	Идентификатор потреблённого продукта

Окончание таблицы 2.5

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
quantity	INT	NOT NULL DEFAULT 1 CHECK (quantity>0)	Количество потреблённых порций соответствующего блюда

Таблица WEEKLY_MENU содержит информацию о меню для пользователя на неделю. Структура данной таблицы представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы WEEKLY_MENU

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор меню на неделю
user_id	INT	NOT NULL FOREIGN KEY	Идентификатор пользователя, для которого создано меню
week_start	DATE	NOT NULL	Дата начала недели, для которой составляется меню
created_at	TIMESTAMP	DEFAULT now()	Дата создания меню

Таблица WEIGHT_HISTORY содержит информацию об изменении массы тела пользователя. Структура данной таблицы представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы WEIGHT_HISTORY

Имя столбца	Тип данных	Ограничение целостности	Краткое пояснение
id	INT	PRIMARY KEY	Идентификатор записи
user_id	INT	NOT NULL FOREIGN KEY	Идентификатор пользователя, для которого сделана запись
weight	NUMERIC(5,2)	NOT NULL CHECK (weight>0)	Значение веса
record_date	DATE	NOT NULL	Дата, когда сделана запись

Такая структура базы данных обеспечивает надежное хранение и эффективное управление данными, что является важным для успешной работы системы. Использование ограничений целостности и внешних ключей между таблицами обеспечивает высокий уровень безопасности и предотвращает несогласованность информации. Скрипт создания таблиц представлен в приложении А.

2.4 Описание других объектов базы данных

Для достижения базой данных целей, помимо хранения данных, необходимо использовать индексы для оптимизации выполнения поисковых запросов и процедуры для логической обработки хранимой информации. Применение этих объектов БД значительно улучшает производительность и возможности базы данных.

Индексы позволяют СУБД мгновенно находить нужные записи, действуя по принципу предметного указателя в книге. В рамках современного приложения скорость поиска любой информации очень важна, поэтому индексы – важный и полезный инструмент в работе с БД такого приложения. СУБД PostgreSQL автоматически создает индексы для первичных ключей таблиц, поэтому здесь мы рассмотрим только те, что будут создаваться вручную. В таблице 2.8 в общем виде описаны дополнительные индексы, создаваемые для каждой таблицы.

Таблица 2.8 – Индексы базы данных

Название таблицы	Столбцы, на которые создаются индексы
MENU_ITEMS	user_id
WEEKLY_MENU	user_id
DAILY_MENU	menu_date
WEIGHT_HISTORY	rser_id, record_date

Одним из ключевых элементов архитектуры проектируемой базы данных являются хранимые процедуры, благодаря которым реализуется вся бизнес-логика и взаимодействие с данными.

Для большинства сущностей достаточно будет предоставить реализацию базовых CRUD-операций и процедур для решения специфических задач при необходимости. Логически их можно разбить на две группы: процедуры администратора и процедуры пользователя. В таблице 2.9 представлено обобщённое описание этих двух групп.

Таблица 2.9 – Обобщённое описание логических групп хранимых процедур

Имя группы	Обобщённое описание назначения
ADMIN	Функции управления пользователями, блюдами, экспорта и импорта блюд и пользователей в JSON

Окончание таблицы 2.9

Имя группы	Обобщённое описание назначения
USER	Функции добавления, удаления собственных блюд, формирование меню, расчёт калорий, отслеживание прогресса в весе и экспорт своих учётных данных и данных о прогрессе в массе тела в JSON

2.4 Выводы по разделу

Подводя итог по данному подразделу, можно заключить, что проектирование базы данных включает не только создание таблиц, но и использование дополнительных объектов СУБД. В данной базе данных были определены следующие объекты для будущего использования: индексы, с целью оптимизации скорости выполнения поисковых запросов и повышения общей производительности системы, а также хранимые процедуры.

Таким образом, эти объекты позволят создать надежную и эффективную архитектуру базы данных.

3 Разработка объектов базы данных

3.1 Роли и пользователи

В разрабатываемой базе данных присутствуют 2 типа пользователей, поэтому для дальнейшего их взаимодействия с базой данных нам необходимо создать для них соответствующие роли. В листинге 3.1 представлен скрипт создания ролей.

```
CREATE ROLE app_admin
    LOGIN
    PASSWORD 'Strong_admin_pass123!';

CREATE ROLE app_user
    LOGIN
    PASSWORD 'Strong_user_pass123!';
```

Листинг 3.1 – Создание ролей

Поскольку весь доступ к данным будет скрыт за процедурами, то для начала мы отзываем пава доступа на эти процедуры у всех, а после – выдаём администратору и пользователю необходимые наборы привилегий. На листинге 3.2 представлен скрипт для выдачи прав пользователям на соответствующие процедуры.

```
GRANT EXECUTE ON FUNCTION nutrition.create_product(TEXT, INT, NUMERIC, TEXT, NUMERIC, NUMERIC, NUMERIC, BOOLEAN) TO app_admin;
GRANT EXECUTE ON FUNCTION nutrition.update_product(INT, TEXT, INT, NUMERIC, TEXT, NUMERIC, NUMERIC, NUMERIC, BOOLEAN) TO app_admin;
GRANT EXECUTE ON FUNCTION nutrition.delete_product(INT) TO app_admin;
GRANT EXECUTE ON FUNCTION nutrition.import_products(JSONB) TO app_admin;
GRANT EXECUTE ON FUNCTION nutrition.export_products() TO app_admin;

GRANT EXECUTE ON FUNCTION nutrition.get_my_profile() TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.update_my_profile(INT, INT) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_weight_history() TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.add_weight_record(DATE, NUMERIC) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_available_products() TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.add_product_to_menu(INT) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.remove_product_from_menu(INT) TO app_user;
```

```
GRANT EXECUTE ON FUNCTION nutrition.add_consumed_food(INT, NUMERIC, DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.remove_consumed_food(INT) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_daily_consumption(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_calorie_progress(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.generate_weekly_menu(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_weekly_menu(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_daily_menu(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.regenerate_day(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_daily_report(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_weekly_report(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.get_weight_report(DATE) TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.export_user_data() TO app_user;
GRANT EXECUTE ON FUNCTION nutrition.import_user_data(JSONB) TO app_user;
```

Листинг 3.2 – Выдача прав пользователям на выполнение соответствующих процедур

Так как мы работаем в СУБД PostgreSQL, то создание ролей, представленное на листинге 3.1 является и созданием пользователей для БД.

3.2 Процедуры и функции

В разделе проектирования были выделены две основные логически разделённые группы функций. Всего было разработано 25 процедур и функций, выполняющий различные операции.

В данном разделе рассмотрим самые интересные и наглядные примеры процедур, которые дадут представление об общем уровне реализации и общем состоянии этого архитектурного компоненты базы данных.

Для начала рассмотрим базовые CRUD-функции на примере функции для получение всех доступных блюд (nutrition.get_available_products()). В листинге 3.3 представлен код данной функции.

```
--GET /api/products
CREATE OR REPLACE FUNCTION nutrition.get_available_products()
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
   RETURN jsonb_build_object(
```

```
        'success', true,
            'data', (
                SELECT jsonb_agg(
                    jsonb_build_object(
                        'id', p.id,
                        'name', p.name,
                        'caloriesPerPortion', p.calories_per_portion,
                        'portionSize', p.portion_size,
                        'portionUnit', p.portion_unit,
                        'protein', p.protein,
                        'fat', p.fat,
                        'carbs', p.carbs,
                        'isPublic', p.is_public
                    )
                )
                FROM nutrition.products p
            )
        );
    EXCEPTION WHEN OTHERS THEN
        RETURN jsonb_build_object(
            'success', false,
            'error', 'Failed to get products: ' || SQLERRM
        );
    END;
    $$;
```

Листинг 3.3 – Скрипт функции nutrition.get_available_products()

Данная функция устроена довольно просто и логично, что упрощает читаемость кода. Также эта функция изначально спроектирована для взаимодействия со сторонним приложением, поэтому возвращаемые данные и даже сообщение об ошибке – в JSON одинакового для всех CRUD-функций этого проекта формата. Важной деталью также является блок EXCEPTION который позволяет избежать непредвиденных ситуаций при взаимодействии с данными и не допускает непредвиденного поведения стороннего приложения.

Далее рассмотрим ещё один важный пример использованных в проекту CRUD-функций – функцию для добавления данных на примере функции добавления потреблённой пользователем еды (nutrition.add_consumed_food). В листинге 3.4 представлен скрипт для создания данной функции.

```
    --POST /api/consumed-food
    CREATE OR REPLACE FUNCTION nutrition.add_consumed_food(
        p_product_id INT,
        p_quantity NUMERIC,
        p_consumed_at DATE
    )
    RETURNS JSONB
    LANGUAGE plpgsql
```

```
    SECURITY DEFINER
    AS $$
    DECLARE
      calories NUMERIC;
    BEGIN
      SELECT calories_per_portion * p_quantity
      INTO calories
      FROM nutrition.products
      WHERE id = p_product_id;

      INSERT INTO nutrition.consumed_food(user_id, product_id, quantity, consumed_at,
calories)
      VALUES (
        current_setting('app.current_user_id')::int,
        p_product_id,
        p_quantity,
        p_consumed_at,
        calories
      );

      RETURN jsonb_build_object(
        'success', true,
        'message', 'Consumed food added',
        'caloriesAdded', calories
      );
    EXCEPTION WHEN OTHERS THEN
      RETURN jsonb_build_object(
        'success', false,
        'error', 'Failed to add consumed food: ' || SQLERRM
      );
    END;
    $$;
```

Листинг 3.4 – Скрипт функции nutrition.add_consumed_food

В этой функции мы можем видеть ту же структуру и принцип обработки исключений, что и в предыдущей. Также формат возвращаемых данных (в этом случае – сообщение об успешном или не успешном добавлении данных в таблицу) – также JSON, что позволяет стороннему приложению легко взаимодействовать с базой данных и отслеживать результаты выполненных операций.

Также стоит рассмотреть функции, важные для логики работы самой системы. Так была разработана функция nutrition.generate_weekly_menu(p_week_start DATE), генерирующая для пользователя меню на неделю, начиная с определённой даты. Скрипт для создания данной функции представлен на листинге 3.5.

```
--POST /api/menu/generate-week
CREATE OR REPLACE FUNCTION nutrition.generate_weekly_menu(p_week_start DATE)
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
DECLARE
   i INT;
   day_date DATE;
   product RECORD;
BEGIN
   FOR i IN 0..6 LOOP
      day_date := p_week_start + i;

      INSERT INTO nutrition.weekly_menu(user_id, menu_date, product_id)
      SELECT current_setting('app.current_user_id')::int, day_date, p.id
      FROM nutrition.menu_items mi
      JOIN nutrition.products p ON p.id = mi.product_id
      ORDER BY random()
      LIMIT 5; --daily products limit
   END LOOP;

   RETURN jsonb_build_object(
      'success', true,
      'message', 'Weekly menu generated'
   );
EXCEPTION WHEN OTHERS THEN
   RETURN jsonb_build_object(
      'success', false,
      'error', 'Failed to generate weekly menu: ' || SQLERRM
   );
END;
$$;
```

Листинг 3.5 – Скрипт функции nutrition.generate_weekly_menu(p_week_start DATE)

Как можем видеть, данная функция семантически устроена тажке как и предыдущие аналоги и формат возвращаемых данных тоже идентичен, что важно для взаимодействия со сторонним приложением и избежения разработки различных способов обработки данных, возвращённых функциями.

3.3 Вывод по разделу

В данном разделе была описана реализация некоторых спроектированных ранее объектов базы данных. Все рассмотренные объекты базы данных вместе создают сложную, но гибкую и эффективную структуру, что делает базу данных не просто хранилищем, а настоящим вычислительным ядром приложения

4 Описание процедур импорта и экспорта

В данном разделе будет предоставлено краткое описание процедур экспорта и импорта данных из файла с данными в формате JSON в таблицуPRODUCTS базы данных и наоборот. Скрипты процедур для экспорта/импорты представлены в приложении Б.

4.1 Описание процедуры экспорта данных

Для экспорта данных из таблицы PRODUCTS в файл в формате JSON был реализована процедура nutrition.export_products. Данная процедура не принимает никаких параметров и возвращает сообщение об успешном или не успешном выполнении, что важно для взаимодействия со сторонним приложением.

В данном случае сериализация данных в JSON выполняется наполовину вручную. PostgreSQL предоставляет полезные утилиты для работы с JSON, такие как jsonb_agg и jsonb_build_object. Они помогают сагрегировать данные и на их основе построить объект нужного формата.

Также, как и в примерах функций, рассмотренных в предыдущем разделе, структура данной функции не отличается от других. Формат возвращаемых данных один и тот же (сообщение об успешном или не успешном выполнении и данные, если предусмотрены), а также имеется блок обработки исключений, позволяющий предотвращать непредвиденное поведение и неправильные операции доступа к данным.

Стоит отдельно упомянуть, что формат возвращаемых данных – стандартные пары ключ:значение, где ключу соответствует имя столбца базы данных, а значению – данные, находящиеся в этом столбце.

4.2 Описание процедуры импорта данных

Для импорта из файла с данными в формате JSON в таблицу PRODUCTS разработана процедура nutrition.import_products. Она принимает входной параметр в формате JSONB и возвращает сообщение об успешном или не успешном выполнении (также в формате JSONB).

Здесь десериализация данных их JSON формата происходит вручную, однако PostgreSQL и в этом случае предоставляет инструменты для работы с JSON (jsonb_array_elements), которые помогают явно определить входящие данные как массив JSON-объектов и удобно взаимодействовать с полями этих элементов при десериализации и вставке элементов в таблицу. Данные инструменты сильно облегчают работы с данными такого формата, так как инкапсулируют в себе большое количество сложной реализации, которую теперь не нужно переопределять самостоятельно, что значительно уменьшает количество точек отказа в приложении и позволяет сделать код более читаемым, масштабируемым и модифицируемым.

Структура данной функции идентична предыдущей и формат возвращаемых данных также общий для всего приложения. Блок обработки исключений

EXCEPTION также присутствует и помогает предотвращать непредвиденное поведение приложения и неправильные операции доступа к данным таблиц.

4.3 Выводы по разделу

В рамках данного раздела были представлены и разобраны процедуры, позволяющие осуществлять экспорт/импорт данных в/из формата JSON. Также для этой цели были использованы встроенные пакеты, что значительно улучшило читаемость кода, помогло значительно сократить количество точек отказа в приложении и сделало его более масштабируемым и модифицируемым.

5 Тестирование производительности

Для проверки производительности базы данных было решено заполнить одну из таблиц большим количеством строк. Для автоматизации данного процесса была разработана соответствующая процедура. В листинге 5.1 представлен скрипт создания процедуры заполнения таблицы рандомно генерируемыми данными в заданном количестве.

```
CREATE OR REPLACE FUNCTION nutrition.generate_test_products(
    p_count INT DEFAULT 100000
)
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
DECLARE
    i INT;
    product_name TEXT;
    base_names TEXT[] := ARRAY[
        'Курица', 'Говядина', 'Свинина', 'Рыба', 'Лосось',
        'Тунец', 'Рис', 'Гречка', 'Овсянка', 'Макароны',
        'Картофель', 'Брокколи', 'Сыр', 'Творог', 'Яйца',
        'Молоко', 'Йогурт', 'Хлеб', 'Яблоко', 'Банан'
    ];
    modifiers TEXT[] := ARRAY[
        'отварной', 'жареный', 'запечённый', 'на пару',
        'обезжиренный', 'классический', 'домашний'
    ];
    portion_size NUMERIC;
    calories INT;
    protein NUMERIC;
    fat NUMERIC;
    carbs NUMERIC;
BEGIN
    FOR i IN 1..p_count LOOP
        product_name :=
            base_names[1 + floor(random() * array_length(base_names, 1))] || ' ' ||
            modifiers[1 + floor(random() * array_length(modifiers, 1))] || ' #' || i;

        portion_size := round((30 + random() * 270)::numeric, 1);
        calories := (50 + random() * 550)::INT;

        protein := round((calories * (0.1 + random() * 0.4) / 4)::numeric, 1);
        fat     := round((calories * (0.1 + random() * 0.4) / 9)::numeric, 1);
        carbs   := round((calories * (0.2 + random() * 0.6) / 4)::numeric, 1);

        INSERT INTO products (
            name,
            calories_per_portion,
            portion_size,
            portion_unit,
```

```
        protein,
                fat,
                carbs,
                is_public,
                created_by
            )
            VALUES (
                product_name,
                calories,
                portion_size,
                'g',
                protein,
                fat,
                carbs,
                (random() > 0.3),
                current_setting('app.current_user_id')::INT
            );
        END LOOP;

        RETURN jsonb_build_object(
            'success', true,
            'message', format('%s products generated successfully', p_count)
        );
    EXCEPTION WHEN OTHERS THEN
        RETURN jsonb_build_object(
            'success', false,
            'error', 'Failed to generate test products: ' || SQLERRM
        );
    END;
    $$;
```

Листинг 5.1 – Скрипт создания функции заполнения таблицы случайно генерируемыми значениями

После заполнения таблицы данными нужно протестировать ее на типовых для этой таблицы запросах. Так как в приложении используется фильтрация по названию блюда, то в качестве примера был выбран данный тип запроса.

Был выполнен запрос для поиска записей о блюдах, в названии которых фигурирует слово йогурт. На рисунке 5.1 представлен план выполнения данного запроса.

	QUERY PLAN text
1	Result (cost=0.00..0.26 rows=1 width=32) (actual time=534.740..534.876 rows=1.00 loop...
2	Buffers: shared hit=1446
3	Planning Time: 0.050 ms
4	Execution Time: 534.909 ms

Рисунок 5.1 – План выполнения запроса к таблице PRODUCTS с заданием конкретного вхождения в имя блюда

Как видим, поиск осуществляется достаточно быстро, хотя и используется полное сканирование. Теперь создадим индекс на столбец NAME для блюда и проанализируем план запроса после создания индекса. Данный план запроса представлен на рисунке 5.2.

	QUERY PLAN text
1	Result (cost=0.00..0.26 rows=1 width=32) (actual time=357.964..358.011 rows=1.00 loop...
2	Buffers: shared hit=1487 read=1
3	Planning Time: 0.051 ms
4	Execution Time: 358.037 ms

Рисунок 5.2 – План выполнения запроса после создания индекса на столбец NAME таблицы PRODUCT

Как видим, время выполнения запроса снизилось даже по сравнению с достаточно быстрым предыдущим показателем. На основании этих данных можно сделать вывод, что индексация нужных столбцов таблиц базы данных значительно ускоряет время выполнения запроса и разгружает систему, а таже предоставляет пользователю более приятный пользовательский опыт.

6 Описание технологии и ее применение в базе данных

В данном разделе будет описана технология Row-Level Security (RLS)**.** Данная технология позволяет реализовать тонкий и гибкий механизм контроля доступа к данным прямо на уровне базы данных. В отличие от традиционных методов, которые чаще всего ограничивают доступ на уровне таблиц (GRANT, REVOKE), данная технология позволяет автоматически фильтровать строки в таблице для каждого пользователя или роли в зависимости от заданных политик безопасности. Это означает, что разные пользователи, выполняя один и тот же запрос, будут видеть совершенно разные наборы строк.

В СУБД PostgreSQL данная технология реализована с помощью политик безопасности (SECURITY POLICIES), которые можно привязывать к таблицам и иным объектам БД. Для работы данных технологии необходимо выполнить два основных шага: включить RLS для целевой таблицы и создать одну или несколько политик.

Политики, в свою очередь, бывают двух основных типов:

–Разрешающие. Политики по умолчанию. Разрешают доступ к строке, если она проходит проверку по хотя бы одной из разрешающих политик;

–Ограничивающие. Более строгие политики. Они могут использоваться совместно с разрешающими. Для доступа к строке необходимо выполнения всех применимых политик.

Наиболее часто будет использоваться политика для оператора SELECT, которая и будет ключевой для обеспечения конфиденциальности данных пользователей. В листинге 6.1 представлен скрипт для создания политик безопасности на примере таблицы USERS/

```
ALTER TABLE nutrition.users ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.products ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.menu_items ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.weekly_menu ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.daily_menu ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.consumed_food ENABLE ROW LEVEL SECURITY;
ALTER TABLE nutrition.weight_history ENABLE ROW LEVEL SECURITY;


--nutrition.users policies
CREATE POLICY user_own_profile_select
ON nutrition.users
FOR SELECT
USING (id = current_setting('app.current_user_id')::INT);

CREATE POLICY user_own_profile_update
ON nutrition.users
FOR SELECT
USING (id = current_setting('app.current_user_id')::INT);
CREATE POLICY admin_all_users
```

```
ON nutrition.users
FOR ALL
USING (current_user='app_admin');
--end of nutrition.users policies
```

Листинг 6.1 – Скрипт для создания политик безопасности для таблицы USERS

Таким образом, технология RLS **в** PostgreSQL предоставляет мощный, декларативный и централизованный способ реализации сложных бизнес-правил доступа к данным, что критически важно для многопользовательских приложений, требующих изоляции данных между клиентами. Также стоит отметить, что эта технология сильно упрощает написание кода и делает его гораздо более читаемым, так как позволяет избавиться от дополнительных проверок и соединений таблиц, что не только повышает количество точек отказа в системе, но и нагружает её, напрямую влияя на производительность.

7 Краткое описание приложения для демонстрации

Дополнительно для более удобной демонстрации работы базы данных было спроектировано и реализовано приложение. Приложение разработано с использованием фреймворка Node.JS на стороне сервера и HTML+CSS+JS на стороне клиента. Такой стек технологий был выбран по нескольким причинам, таким как быстрая разработка, так как порог вхождения в Node.JS меньше, чем у некоторых других популярных фреймворков и технологий, также это позволяет в какой-то степени унифицировать интерфейс благодаря тому, что и для клиента, и для сервера используется JavaScript. Что касается клиентской части, то тут тоже выбор был сделан в сторону того, чтобы не выбирать слишком сложные решения, не оправдываемые задачей (например, Angular или React) и требующие для изучения много времени, а взять уже изученные веб-технологии, которые к тому же не требуют достаточно серьезной подготовки для их использования и соответственно не требуют дополнительных временных затрат.

Приложение предоставляет следующие возможности:

–регистрация и авторизация для пользователей и администраторов. Пользователи могут зарегистрироваться в приложении, а также в дальнейшем осуществлять вход под только что созданной учетной записью, учетная запись админа заранее определена и используя эти данные можно войти под учетной записью администратора, создание новых учетных записей для администратора;

–Просмотр, фильтрация блюд в меню. Пользователи и администратор могу просматривать, фильтровать и добавлять свои продукты;

–Управление профилем. Пользователь может изменять свои лимиты калорий, администратор может изменять информацию о пользователях;

–Отслеживание веса. Пользователь может делать записи о своём весе и отслеживать прогресс;

–Просмотр меню. Пользователь может просматривать меню на день и на неделю, а также генерировать и перегенерировать его;

–Генерация и просмотр отчётов. Более персонализированная версия для пользователя и более расширенная версия для администратора;

–Добавление собственных блюд в меню;

–Экспорт и импорт данных в JSON. Пользователь и администратор могут делать это для своих таблиц;

–Отслеживаение прогресса по калориям.

Приложение позволяет продемонстрировать функционал разработанной базы данных, упростить взаимодействие с ней через удобный UI и UX. Также позволяет осуществлять удобный и комплексный поиск для более удобного и быстрого получения нужных товаров.

8 Руководство пользователя

При входе в приложение пользователь попадает на главную страницу. В верхней части расположена панель навигации, на которой видно название сервиса и краткие сведения о текущем статусе (гость или авторизованный пользователь). Здесь же есть кнопка «Войти», которая открывает блок авторизации и регистрации. На стартовом экране по центру находится карточка с двумя вкладками: «Вход» и «Регистрация». Вход требует указать логин и пароль; после успешного входа пользовательские данные сохраняются в локальном хранилище, а интерфейс переключается в рабочий режим. Регистрация доступна гостю и позволяет создать новую учётную запись: достаточно ввести логин и пароль и подтвердить действие.

После регистрации можно сразу перейти на вкладку «Вход» и авторизоваться. Если пользователь остаётся гостем, доступны только публичные данные, а административные функции скрыты.

После авторизации верхняя панель показывает имя и роль текущего пользователя. В личном кабинете видны разделы: профиль, вес, продукты, меню, потребление и отчёты. В разделе профиля отображаются логин и лимиты калорий; здесь же можно задать новые дневные и недельные лимиты.

В разделе веса отображается таблица с историей измерений и форма для добавления новой записи с датой и значением веса. Раздел продуктов показывает таблицу доступных продуктов; здесь можно искать по названию, просматривать состав, а также создавать собственные продукты (с указанием порции, калорий и БЖУ) и добавлять их в личное меню. Раздел меню позволяет добавить или убрать продукт из меню, сгенерировать меню на неделю, показать меню за день или неделю и перегенерировать конкретный день; меню выводится таблицей с датой, порцией, калорийностью и количеством. Раздел потребления позволяет фиксировать, что и сколько было съедено за выбранную дату, и просматривать список съеденного за день в табличном виде. Раздел отчётов выводит сводку по калориям за выбранный день и агрегированный недельный отчёт по датам, а также краткий отчёт по весу за неделю; данные показываются таблицами и текстовым резюме. Если роль пользователя — «app_admin», становится доступна вкладка «Админ-панель». В ней есть таблица всех пользователей с ролями и лимитами, формы для создания пользователя и администратора, обновления лимитов и удаления записей. Также доступен экспорт пользователей в JSON. В админской части для продуктов можно выгрузить все продукты в JSON или загрузить их из JSON. Чтобы выйти из системы, в правой части верхней панели есть кнопка «Выйти», которая очищает сохранённые данные и возвращает интерфейс в состояние гостя.

Заключение

В рамках курсового проекта была разработана база данных для системы учёта питания и физической активности с использованием СУБД PostgreSQL. В ней хранится вся необходимая информация о пользователях, продуктах питания, рецептах, дневниковых записях, тренировках и других сущностях, выделенных в ходе анализа предметной области и процесса проектирования. Для создания всей структуры базы данных (таблиц, индексов, связей) и настройки вспомогательных механизмов был написан соответствующий SQL-код.

Была успешно внедрена и протестирована технология безопасности на уровне строк (Row-Level Security, RLS), которая стала основой для обеспечения конфиденциальности и целостности данных в многопользовательской среде. RLS позволила реализовать строгое и прозрачное разграничение доступа, гарантируя, что каждый пользователь может работать исключительно со своими собственными данными (дневником питания, историей тренировок, личными метриками). Это решение значительно повысило безопасность системы, минимизировав риски несанкционированного доступа на уровне базы данных и упростив логику контроля прав в клиентском приложении.

Также были реализованы процедуры для экспорта и импорта пользовательских данных (например, дневника за определённый период или библиотеки личных рецептов) в формат JSON. Это расширяет возможности системы, позволяя пользователям создавать резервные копии, переносить данные и проводить внешний анализ. Для администраторов это упрощает процессы миграции, тестирования и интеграции с аналитическими инструментами.

По итогам разработки были выполнены все поставленные функциональные и нефункциональные требования. Спроектированная база данных обладает высокой степенью нормализации, гибкостью для внесения изменений (например, добавления новых типов метрик или категорий упражнений) и оптимизирована для типовых операций — добавления записей в дневник и формирования аналитических отчётов. Использование индексов и эффективных типов данных обеспечивает необходимую производительность даже при росте объёма информации.

В целом, разработанное решение полностью соответствует исходным задачам. Оно обеспечивает надёжное, безопасное и производительное хранение, обработку и извлечение данных, что является фундаментом для создания удобного и функционального приложения, способного помочь пользователям в достижении их фитнес-целей и целей по здоровью.

Список использованных источников

1 MyFitnessPall [Электронный ресурс] – Режим доступа: https://MyFitnessPall.com – Дата доступа: 12.09.2025

2 YOUZIO [Электронный ресурс] – Режим доступа: https://YPUZIO.com – Дата доступа: 12.09.2025

3 PosgreSQL [Электронный ресурс] – Режим доступа: https://www.postgresql.ord / – Дата доступа: 10.10.2025

ПРИЛОЖЕНИЕ А

```
/*
CREATE SCHEMA nutrition;
*/

select current_schema();

CREATE TABLE nutrition.users(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	username TEXT UNIQUE NOT NULL,
	password_hash TEXT NOT NULL,
	daily_cal_limit INT NOT NULL CHECK(daily_cal_limit>0),
	weekly_cal_limit INT NOT NULL CHECK(weekly_cal_limit>0),
	create_at TIMESTAMP DEFAULT now(),
	role TEXT NOT NULL DEFAULT 'app_user' CHECK(role in ('app_admin','app_user'))
);

select * from nutrition.users;
alter table nutrition.users add column role TEXT NOT NULL DEFAULT 'app_user' CHECK(role in ('app_admin','app_user'));


CREATE TABLE nutrition.products(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	name TEXT NOT NULL,
	calories_per_portion INT NOT NULL CHECK(calories_per_portion>0),
	portion_size NUMERIC(6,2) NOT NULL CHECK(portion_size>0),
	portion_unit TEXT NOT NULL CHECK(portion_unit in('g','kg','ml','l','pcs','г','кг','мл','л','шт')),
	protein NUMERIC(5,2) NOT NULL CHECK(protein>=0),
	fat NUMERIC(5,2) NOT NULL CHECK(fat>=0),
	carbs NUMERIC(5,2) NOT NULL CHECK(carbs>=0),
	created_by INT REFERENCES nutrition.users(id),
	is_public BOOLEAN DEFAULT FALSE
);



CREATE TABLE nutrition.menu_items(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	user_id INT NOT NULL REFERENCES nutrition.users(id) ON DELETE CASCADE,
	product_id INT NOT NULL REFERENCES nutrition.products(id),
	UNIQUE(user_id,product_id)
);


CREATE TABLE nutrition.weekly_menu(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	user_id INT NOT NULL REFERENCES nutrition.users(id) ON DELETE CASCADE,
	week_start DATE NOT NULL,
	created_at TIMESTAMP DEFAULT now(),
	UNIQUE(user_id, week_start)
```

```
);


CREATE TABLE nutrition.daily_menu(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	weekly_menu_id INT NOT NULL REFERENCES nutrition.weekly_menu(id) ON DELETE CASCADE,
	menu_date DATE NOT NULL,
	UNIQUE(weekly_menu_id,menu_date)
);

CREATE TABLE nutrition.consumed_food(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	daily_menu_id INT NOT NULL REFERENCES nutrition.daily_menu(id) ON DELETE CASCADE,
	product_id INT NOT NULL REFERENCES nutrition.products(id),
	quantity NUMERIC(6,2) NOT NULL DEFAULT 1 CHECK (quantity>0)
);

CREATE TABLE nutrition.weight_history(
	id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
	user_id INT NOT NULL REFERENCES nutrition.users(id) ON DELETE CASCADE,
	weight NUMERIC(5,2) NOT NULL CHECK (weight>0),
	record_date DATE NOT NULL,
	UNIQUE(user_id,record_date)
);

CREATE INDEX idx_menu_items_user ON nutrition.menu_items(user_id);
CREATE INDEX idx_weekly_menu_user ON nutrition.weekly_menu(user_id);
CREATE INDEX idx_daily_menu_date ON nutrition.daily_menu(menu_date);
CREATE INDEX idx_weight_user_date ON nutrition.weight_history(user_id, record_date);
```

ПРИЛОЖЕНИЕ Б

```
-- импорт продуктов из JSON (админ), принимает JSON-массив объектов
CREATE OR REPLACE FUNCTION nutrition.admin_import_products(p_products JSONB)
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
DECLARE rec JSONB;
BEGIN
  IF current_setting('app.current_user_role', true) <> 'app_admin' THEN
    RAISE EXCEPTION 'Access denied: admin only';
  END IF;

  FOR rec IN SELECT * FROM jsonb_array_elements(p_products)
  LOOP
    INSERT INTO nutrition.products(name, calories_per_portion, portion_size, portion_unit,
                                   protein, fat, carbs, is_public, created_by)
    VALUES (
      rec->>'name',
      COALESCE((rec->>'calories_per_portion')::INT, rec->>'calories')::INT,
      (rec->>'portion_size')::NUMERIC,
      rec->>'portion_unit',
      (rec->>'protein')::NUMERIC,
      (rec->>'fat')::NUMERIC,
      (rec->>'carbs')::NUMERIC,
      COALESCE((rec->>'is_public')::BOOLEAN, true),
      current_setting('app.current_user_id')::INT
    )
    ON CONFLICT (id) DO NOTHING;
  END LOOP;

  RETURN jsonb_build_object('success', true, 'message', 'Products imported');
EXCEPTION WHEN OTHERS THEN
  RETURN jsonb_build_object('success', false, 'error', 'Import products failed: ' || SQLERRM);
END;
$$;

CREATE OR REPLACE FUNCTION nutrition.user_export_products()
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
BEGIN
  RETURN jsonb_build_object(
    'success', true,
    'data', (
      SELECT jsonb_agg(row_to_json(p))
      FROM nutrition.products p
      WHERE p.created_by = current_setting('app.current_user_id')::INT
```

```
      OR p.is_public = TRUE
    )
  );
EXCEPTION WHEN OTHERS THEN
  RETURN jsonb_build_object('success', false, 'error', 'Export products failed: ' || SQLERRM);
END;
$$;


CREATE OR REPLACE FUNCTION nutrition.admin_export_users()
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
BEGIN
  IF current_setting('app.current_user_role', true) <> 'app_admin' THEN
    RAISE EXCEPTION 'Access denied: admin only';
  END IF;

  RETURN jsonb_build_object(
    'success', true,
    'data', (SELECT jsonb_agg(jsonb_build_object(
      'id', id, 'username', username, 'role', role,
      'dailyCalorieLimit', daily_cal_limit, 'weeklyCalorieLimit', weekly_cal_limit,
      'createdAt', create_at)) FROM nutrition.users)
  );
EXCEPTION WHEN OTHERS THEN
  RETURN jsonb_build_object('success', false, 'error', 'Export users failed: ' || SQLERRM);
END;
$$;

-- экспорт всех продуктов (админ)
CREATE OR REPLACE FUNCTION nutrition.admin_export_products()
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
BEGIN
  IF current_setting('app.current_user_role', true) <> 'app_admin' THEN
    RAISE EXCEPTION 'Access denied: admin only';
  END IF;

  RETURN jsonb_build_object(
    'success', true,
    'data', (SELECT jsonb_agg(row_to_json(p)) FROM nutrition.products p)
  );
EXCEPTION WHEN OTHERS THEN
  RETURN jsonb_build_object('success', false, 'error', 'Export products failed: ' || SQLERRM);
END;
$$;
```

ПРИЛОЖЕНИЕ В

```
-- Расширение для crypt/gen_salt
CREATE EXTENSION IF NOT EXISTS pgcrypto;

SET search_path = nutrition, pg_temp;

-------------------------------------------------
-- ВСПОМОГАТЕЛЬНАЯ ФУНКЦИЯ ДЛЯ daily_menu
-------------------------------------------------
CREATE OR REPLACE FUNCTION nutrition.ensure_daily_menu(
    p_user_id INT,
    p_date    DATE
) RETURNS INT
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
DECLARE
    v_week_start  DATE;
    v_weekly_id   INT;
    v_daily_id    INT;
BEGIN
    -- начало недели (понедельник)
    v_week_start := p_date - ((EXTRACT(ISODOW FROM p_date)::INT) - 1);

    -- weekly_menu
    SELECT id INTO v_weekly_id
    FROM nutrition.weekly_menu
    WHERE user_id   = p_user_id
      AND week_start = v_week_start;

    IF v_weekly_id IS NULL THEN
        INSERT INTO nutrition.weekly_menu(user_id, week_start)
        VALUES (p_user_id, v_week_start)
        RETURNING id INTO v_weekly_id;
    END IF;

    -- daily_menu
    SELECT id INTO v_daily_id
    FROM nutrition.daily_menu
    WHERE weekly_menu_id = v_weekly_id
      AND menu_date      = p_date;

    IF v_daily_id IS NULL THEN
        INSERT INTO nutrition.daily_menu(weekly_menu_id, menu_date)
        VALUES (v_weekly_id, p_date)
        RETURNING id INTO v_daily_id;
    END IF;

    RETURN v_daily_id;
END;
```

```
$$;

-------------------------------------------------
-- АУТЕНТИФИКАЦИЯ
-------------------------------------------------

-- Регистрация пользователя
CREATE OR REPLACE FUNCTION nutrition.user_register(
    p_username TEXT,
    p_password TEXT
)
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
DECLARE
    new_id INT;
BEGIN
    IF EXISTS (SELECT 1 FROM nutrition.users WHERE username = p_username) THEN
        RETURN jsonb_build_object('success', false, 'error', 'Username already exists');
    END IF;

    INSERT INTO nutrition.users (username, password_hash, role, daily_cal_limit, weekly_cal_limit)
    VALUES (p_username, crypt(p_password, gen_salt('bf')), 'app_user', 2000, 14000)
    RETURNING id INTO new_id;

    RETURN jsonb_build_object(
        'success', true,
        'user', jsonb_build_object(
            'id', new_id,
            'username', p_username,
            'role', 'app_user'
        )
    );
EXCEPTION WHEN OTHERS THEN
    RETURN jsonb_build_object('success', false, 'error', 'Registration failed: ' || SQLERRM);
END;
$$;

-- Логин
CREATE OR REPLACE FUNCTION nutrition.user_login(
    p_username TEXT,
    p_password TEXT
)
RETURNS JSONB
LANGUAGE plpgsql
SECURITY DEFINER
SET search_path = nutrition, pg_temp
AS $$
DECLARE
    stored_hash TEXT;
    user_id     INT;
```

```
    user_role   TEXT;
BEGIN
    SELECT id, password_hash, role
    INTO user_id, stored_hash, user_role
    FROM nutrition.users
    WHERE username = p_username;

    IF NOT FOUND THEN
        RETURN jsonb_build_object('success', false, 'error', 'Invalid username or password');
    END IF;

    IF crypt(p_password, stored_hash) <> stored_hash THEN
        RETURN jsonb_build_object('success', false, 'error', 'Invalid username or password');
    END IF;

    PERFORM set_config('app.current_user_id', user_id::TEXT, false);
    PERFORM set_config('app.current_user_role', user_role, false);

    RETURN jsonb_build_object(
        'success', true,
        'message', 'Login successful',
        'user', jsonb_build_object(
            'id', user_id,
            'username', p_username,
            'role', user_role
        )
    );
EXCEPTION WHEN OTHERS THEN
    RETURN jsonb_build_object('success', false, 'error', 'Login failed: ' || SQLERRM);
END;
$$;
```