Реферат

Расчетно-пояснительная записка состоит из 44 страниц и содержит 15 рисунков, 24 таблицы, 22 источника и 3 приложения.

Ключевые слова: база данных, учет времени, PostgreSQL, Java, Swagger, индексы базы данных.

Цель работы — Разработать базу данных для хранения и обработки данных для приложения учета и аудита времени, потраченного на рабочие и личные задачи.

В работе осуществляется проектирование и реализация базы данных для учета и аудита времени, затраченного на рабочие и личные задачи. Была выбрана объектно-реляционная модель данных и СУБД PostgreSQL. Приложение, реализованное на языке Java, поддерживает клиент-серверную архитектуру и взаимодействие через Swagger. Разработанная серверная часть может быть интегрирована в корпоративные системы для управления временем сотрудников, а также использована индивидуальными пользователями для личного тайм-менеджмента.

В ходе работы было проведено исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в базе данных.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

База данных (БД) — это систематически организованный набор данных, хранящийся в памяти вычислительной системы, который представляет состояние объектов и их взаимосвязи в предметной области [1].

Система управления базами данных (СУБД) — это набор программных и языковых инструментов, разработанных для создания, управления и обеспечения доступа к базам данных для множества пользователей [1].

Программное обеспечение — ПО.

ВВЕДЕНИЕ

В современном мире, где время является ценным ресурсом, его эффективное управление становится критически важным как для индивидуальных пользователей, так и для организаций. Знание того, сколько времени требуется на выполнение каждой задачи, критически важно для оптимизации рабочих процессов. Именно поэтому разработка приложения для учета и анализа времени, затраченного на выполнение задач является актуальной задачей [2].

Цель курсовой работы — разработка базы данных для хранения и обработки данных приложения учета и аудита времени, потраченного на рабочие и личные задачи.

Для достижения этой цели необходимо выполнить следующие задачи:

- сформулировать описание пользователей проектируемого приложения и провести анализ существующих решений;
- спроектировать БД и приложение для учета и аудита времени;
- реализовать БД и приложение;
- провести исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД.

1 Аналитический раздел

В данном разделе будет рассмотрено следующее: анализ предметной области, формулировка требований к БД и приложению, формализация и описание информации, анализ моделей данных, ER-диаграмма, формализация и описание пользователей и диаграмма вариантов использования.

1.1 Анализ предметной области

Предметная область «эффективное управление временем и задачами» охватывает планирование, отслеживание и анализ времени, затрачиваемого на различные задачи и проекты. В этой области применяются как классические, так и современные методологии.

Существует множество теорий и методологий, способствующих эффективному управлению временем. Среди наиболее известных подходов выделяются Матрица Эйзенхауэра, которая помогает приоритизировать задачи по их срочности и важности, метод Помодоро, предлагающий разбивку работы на короткие интервалы с перерывами, и система *GTD*, нацеленная на снижение стресса за счет переноса задач из головы в внешнюю систему организации [3].

Современные методики, такие как *Agile*, *Scrum* и *Kanban*, вносят гибкость в управление проектами. *Scrum* использует короткие итеративные циклы, в течение которых команда выполняет заранее определенный объем работы, обеспечивая регулярную обратную связь и быструю адаптацию к изменениям. *Kanban* управляет задачами в реальном времени с помощью визуализации на доске *Kanban*, что способствует равномерному распределению работы и выявлению узких мест. Эти подходы позволяют командам эффективно реагировать на изменения и поддерживать цикличность в деятельности [4].

Эти методики позволяют командам быстро реагировать на изменения, эффективно управлять рабочим процессом, обеспечивая постоянную обратную связь и поддерживая цикличность в деятельности.

Анализ аналогичных решений

На рынке существует большое количество приложений, которые имеют функциональность замера времени для задач, наиболее известные: *clockify* [5], *toggl* [6], *timecamp* [7], также есть приложения по планированию задач в стиле канбан-досок, например как *trello* [8]. Выделим следующие критерии для сравнения:

– наличие замера времени для задачи;

– ограниченность в создание проектов;

– наличие аналитики;

– ограниченность в количестве пользователей у проекта;

– наличие канбан-досок для удобной организации задач.

В таблице 1.1 представлено сравнение существующих решений для бесплатных версий приложений.

Таблица 1.1 – Сравнение существующих решений

Критерий	**Clockify**	**Toggl**	**TimeCamp**	**Trello**
Замер времени	Есть	Есть	Есть	Нет
Количество проектов	Неограниченно	Неограниченно	Ограничено	Ограничено
Аналитика	Есть	Есть	Есть	Нет
Количество пользователей	Неограниченно	Ограничено	Ограничено	Ограничено
Канбан-доски	Нет	Нет	Нет	Есть

Как видно из таблицы 1.1 ни одно из рассмотренных решений не удовлетворяет всем критериям сравнения, кроме того, данные приложения являются зарубежными, следовательно, мое решение будет актуальным.

1.2 Требования к разрабатываемой базе данных и приложению

База данных должна содержать информацию о пользователях, группах, проектах, задачах, карточках, тегах и делах. Должны быть определены роли с соответствующими ограничениями, чтобы гарантировать целостность данных.

Приложение должно включать следующую функциональность:

– регистрация и аутентификация пользователей;

– изменение проектов, карточек, задач, дел и тегов;

– замер времени для конкретной задачи;

– просмотр аналитики проекта;

– создание и изменение групп пользователей.

1.3 Формализация и описание информации

Разрабатываемая БД для приложения по учету и аудиту затраченного времени на определенные задачи, должна содержать информацию о пользователях, проектах, задачах, карточках, тегах, делах и группах. *Проекты* могут содержать набор задач, карточек, тегов объединенных общей целью. *Карточки* — это визуальные представления задач в рамках проектов. Использование карточек позволит применить методики управления проектами, такие как *Kanban* и *Scrum*. *Теги* можно будет использовать для категоризации и фильтрации задач внутри проектов. Они облегчат организацию данных и ускорят поиск необходимой информации по ключевым словам или признакам. Под *делами* понимаются конкретная запись в списке дел пользователя. *Группы* создаются для организации совместной работы нескольких пользователей над общим проектом. Это позволит управлять доступом к проектам и задачам, распределять обязанности между членами группы и следить за ходом выполнения работ.

В таблице 1.2 представлены необходимые сущности и информация, которая должна содержаться в этой сущности для разрабатываемой БД.

Таблица 1.2 – Необходимые сущности и их информация

Сущность	**Информация**
Пользователь	Имя, фамилия, роль, почта, дата последнего входа и дата регистрации
Проект	Название, описание
Карточка	Название, описание, статус
Задача	Название, статус, описание, дата начала и завершения задачи, начало записи таймера, сумма времени таймера
Группа	Название, описание
Тег	Название, цвет
Дело	Статус, приоритет, дата завершения, контент

На рисунке 1.1 приведена ER диаграмма в нотации Чена.

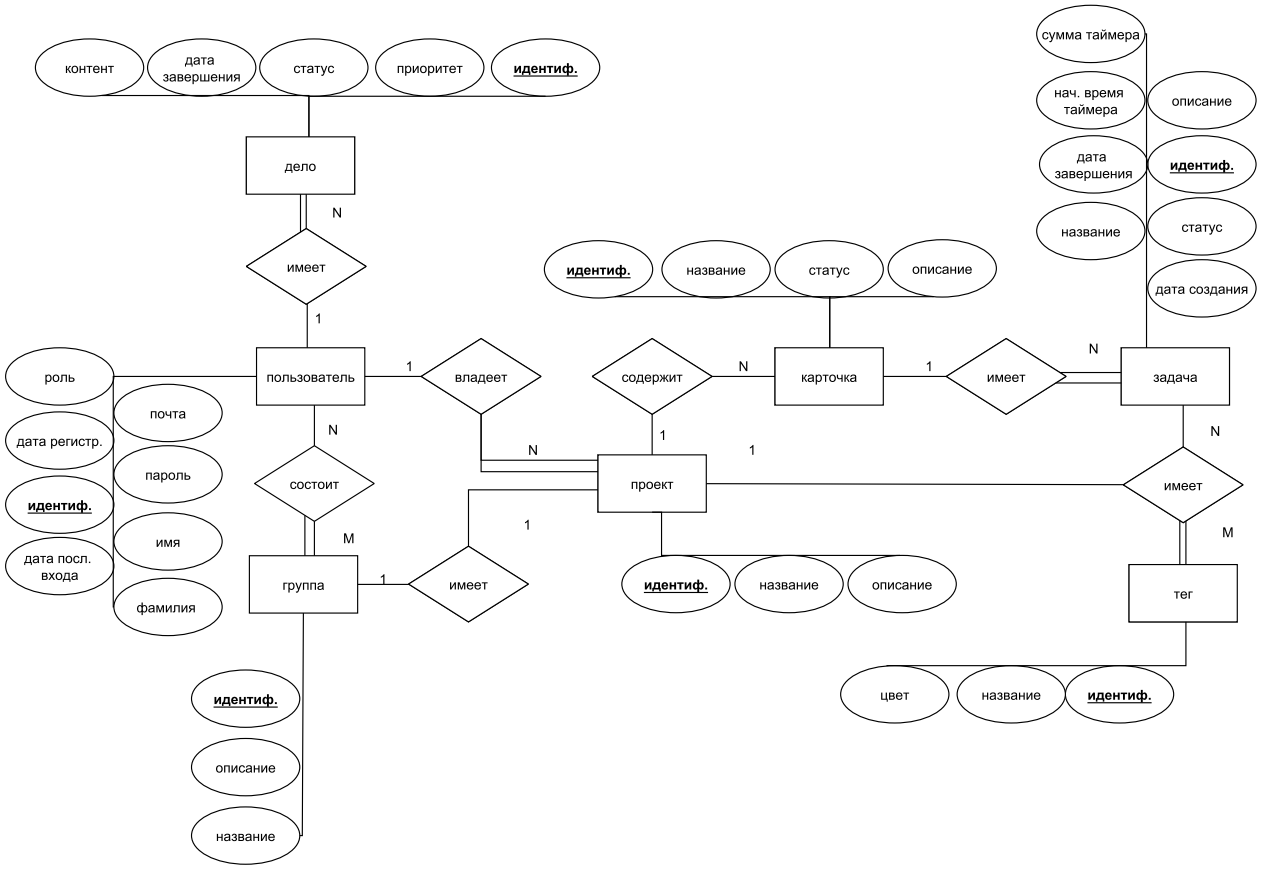


Рисунок 1.1 – ER диаграмма

1.4 Формализация и описание пользователей

Для взаимодействия с приложением по учету и аудиту времени, было выделено три роли пользователей: администратор, менеджер групп и пользователь.

В таблице 1.3 представлена функциональность для администратора, менеджер групп и пользователя. Также в таблице 1.3 используются следующие сокращения: 1 означает роль администратор, 2 — менеджер групп, 3 —пользователь.

Таблица 1.3 – Функциональность администратора, менеджера групп и пользователя

Функциональность	**1**	**2**	**3**
Регистрация и аутентификация	+	+	+
Добавить и удалить пользователя	+		
Получить информацию о пользователях	+		
Создать или изменить дело, проект, карточку, задачу или тег	+	+	+
Получить информацию о списке дел, проекте, карточках, задачах и тегах	+	+	+
Замерить время для конкретной задачи	+	+	+
Получить аналитику по проекту	+	+	+
Создать и изменить группу	+	+	

1.5 Диаграмма вариантов использования

На рисунке 1.2 приведена диаграмма вариантов использования приложения, для ранее выделенных ролей.

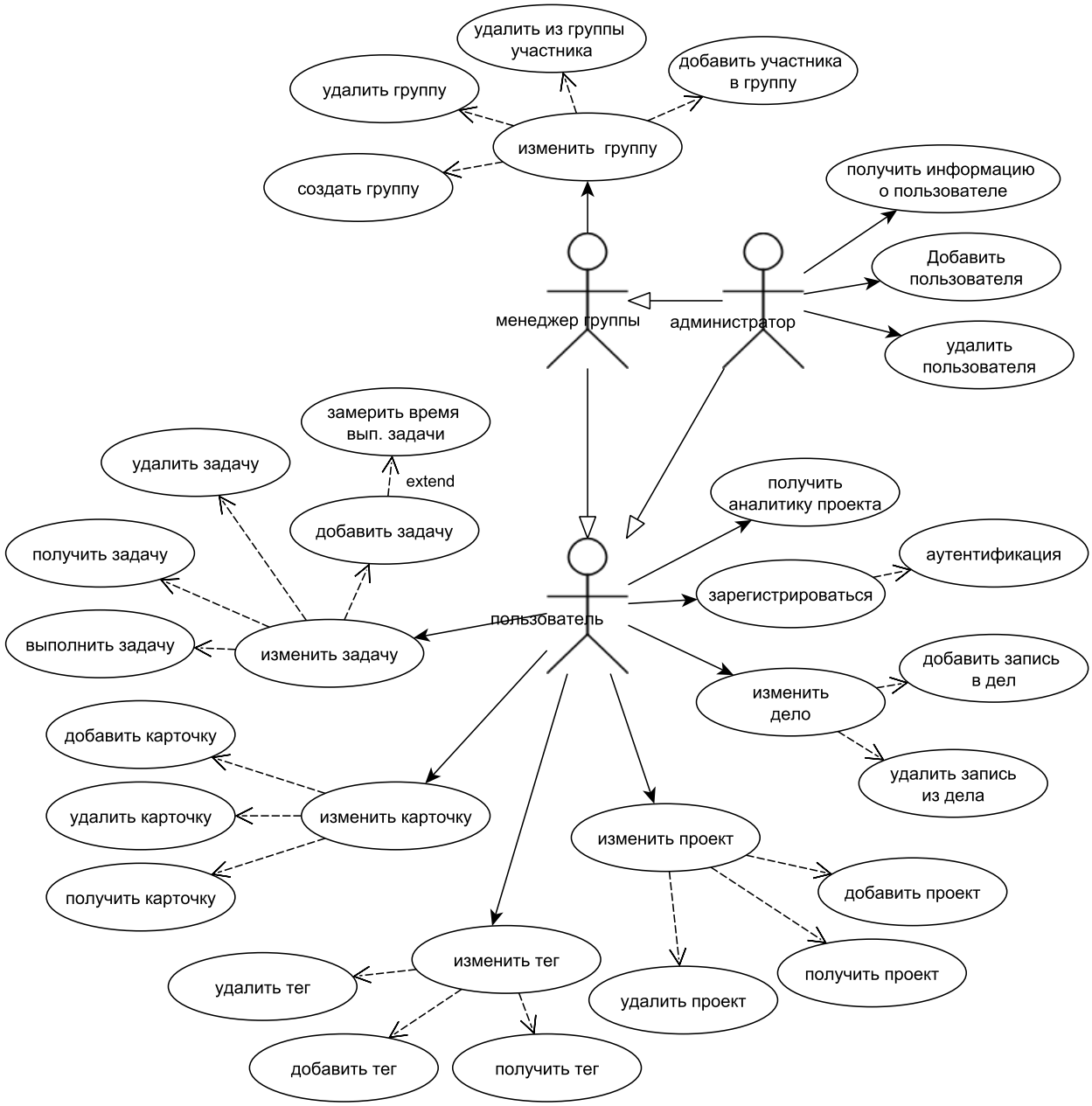


Рисунок 1.2 – Диаграмма использования

1.6 Модели данных

Модель данных представляет собой абстрактное описание структуры и организации данных, определяющее правила хранения, управления и обработки данных в БД. К основным моделям представления данных отно-

сятся следующие: реляционные, постреляционные, иерархические, сетевые, объектно-ориентированные каждая из которых предлагает различные подходы к организации данных в системе [1; 9].

Иерархическая модель данных организует информацию в виде древовидной структуры, где каждая запись связана с одним родителем и может иметь много дочерних элементов. Это эффективно для данных с четкой иерархией, но ограничивает гибкость, так как любые изменения в структуре данных требуют значительных усилий и могут нарушить целостность данных [1; 9].

Сетевая модель данных представляет данные в виде графа, позволяя элементам иметь множество связей и предков. Это обеспечивает гибкость и быстродействие, но модель сложна в понимании и управлении, что может затруднить ее использование без специализированных знаний. Также сложность управления связями может привести к ослаблению контроля за целостностью данных [1; 9].

Реляционная модель данных организует информацию в форме двухмерных таблиц, которые представляют собой наборы отношений, изменяющиеся со временем. Эти таблицы позволяют хранить данные о различных объектах предметной области и моделировать связи между ними. Главные *преимущества* реляционной модели включают ее простоту, понятность и удобство в физической реализации, что сделало ее широко распространенной. Однако, среди *недостатков* модели — отсутствие стандартных методов для идентификации отдельных записей и сложности в описании иерархических и сетевых связей [10].

В классической реляционной модели данные в таблицах представлены так, что каждое поле записи является атомарным, то есть не может быть дополнительно разделено на подзначения. Это требование соответствует первой нормальной форме, которая может ограничивать гибкость при разработке приложений. В отличие от этого, **постреляционные модели данных** устраняют эти ограничения, позволяя использовать многозначные поля. В таких полях значения могут содержать подзначения, которые рассматриваются как отдельная встроенная таблица, что обеспечивает более гибкую структуру данных для сложных приложений [11].

Объектно-ориентированная модель данных интегрирует концепции объектно-ориентированного программирования с управлением базами

данных, позволяя каждой записи в базе данных быть уникально идентифицированной и управляемой через методы и свойства, аналогичные тем, что используются в объектно-ориентированных языках. Эта модель позволяет структурировать данные как взаимосвязанные объекты, обладающие состоянием и поведением. *Преимущества* объектно-ориентированной модели включают лучшее управление сложными данными и улучшенную модульность кода. *Недостатки* включают высокую сложность освоения и потенциально ниже производительность по сравнению с реляционными моделями [12].

Объектно-реляционная модель данных интегрирует объектно-ориентированные особенности, такие как наследование, полиморфизм, и инкапсуляция, в реляционную модель данных. *Преимущества:* благодаря поддержке сложных типов данных и наследования, модель позволяет более точно отражать реальные объекты и отношения, поддержка объектно-ориентированных концепций облегчает разработку, т.к. модель данных ближе к используемым в приложениях объектам, возможность переиспользования кода и логики наследования способствует повышению эффективности разработки и поддержки. *Недостатки:* интеграция объектно-ориентированных элементов может привести к снижению производительности по сравнению с чисто реляционными моделями [13].

После анализа различных моделей данных и проведенной формализации задачи, данных и пользователей, была выбрана объектно-реляционная модель данных, это обусловлено ее удобством в интеграции, возможностью масштабирования и поддержкой пользовательских и сложных типов данных.

2 Конструкторский раздел

В данном разделе будет представлено описание сущностей и ограничений целостности в проектируемой БД, диаграмма БД, схемы функций БД и описание ролевой модели на уровне БД.

2.1 Проектирование базы данных

Исходя из ER диаграммы, приведенной в предыдущем разделе (см. рисунок 1.1), были выделены следующие таблицы :

- таблица пользователей (user_app);
- таблица дел (todo_node);
- таблица проектов (project);
- таблица ролей (roles);
- таблица групп пользователей (group_user);
- таблица для связи группы с пользователем (group_member);
- таблица карточек (table_app);
- таблица тегов (tag);
- таблица задач (task);
- таблица для связи задач с тегами (task_tag).

2.2 Сущности и ограничения целостности в базе данных

Информация о полях в таблице ролей (roles) представлена в таблице 2.1.

Таблица 2.1 – Описание таблицы ролей

Поле	**Тип данных**	**Ограничение**	**Описание**
id_role	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 16, значение должно быть 'ADMIN', 'USER' или 'MANAGER'.	Имя роли

Информация о полях в таблице пользователей (user_app) представлена в таблице 2.2.

Таблица 2.2 – Описание таблицы пользователей

Поле	**Тип данных**	**Ограничение**	**Описание**
id_user	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
email	Символьный	Уникальное, обязательное, максимальная длина — 255 символов	Адрес электронной почты пользователя (логин пользователя)
password	Символьный	Обязательное, максимальная длина — 255 символов	Хэш пароля пользователя
user_name	Символьный	Обязательное, максимальная длина — 255	Имя пользователя
user_secondname	Символьный	Обязательное, максимальная длина — 255 символов	Фамилия пользователя
data_last_log_in	Дата	—	Дата последнего входа пользователя
data_sign_in	Дата	—	Дата регистрации пользователя
role_id	Целочисленный	Обязательное	Внешний ключ, для определения роли

Информация о полях в таблице записи списка дел (todo_node) представлена в таблице 2.3.

Таблица 2.3 – Описание таблицы записи списка дел

Поле	**Тип данных**	**Ограничение**	**Описание**
id_node	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
content	Символьный	Максимальная длина — 255 символов	Описание записи
priority	Символьный	Максимальная длина — 255 символов	Приоритет записи
status	Символьный	Максимальная длина — 255 символов, обязательное, значение должно быть 'ACTIVE' или 'NO_ACTIVE',	Статус записи
due_data	Дата	—	Дата и время срока выполнения записи
user_id	Целочисленный	Обязательное	Идентиф. пользователя, создавшего запись, внешний ключ

Информация о полях в таблице проекта (project) представлена в таблице 2.4.

Таблица 2.4 – Описание таблицы проекта

Поле	**Тип данных**	**Ограничение**	**Описание**
id_project	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 255 символов	Название проекта
description	Символьный	Обязательное, максимальная длина — 255 символов	Описание проекта
user_id	Целочисленный	Обязательное	Идентификатор пользователя, создавшего проект, внешний ключ

Информация о полях в таблице карточек (table_app) представлена в таблице 2.5.

Таблица 2.5 – Описание таблицы карточек

Поле	**Тип данных**	**Ограничение**	**Описание**
id_project	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 255 символов	Название таблицы
description	Символьный	Максимальная длина — 255 символов	Описание таблицы
status	Символьный	Максимальная длина — 255 символов, обязательное, значение должно быть 'ACTIVE' или 'NO_ACTIVE'	Статус таблицы
project_id	Целочисленный	Обязательное	Идентификатор проекта, к которому относится таблица задач, внешний ключ

Информация о полях в таблице задачи (task) представлена в таблице 2.6.

Таблица 2.6 – Описание таблицы задачи

Поле	**Тип данных**	**Ограничение**	**Описание**
id_task	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 255 символов	Название задачи
description	Символьный	Максимальная длина — 255 символов	Описание задачи
status	Символьный	Максимальная длина — 255 символов, обязательное, значение должно быть 'ACTIVE' или 'NO_ACTIVE'	Статус задачи
sum_timer	Целочисленный	—	Общее время выполнения задачи
start_timer	Дата	—	Время начала выполнения задачи
timer_add_task	Дата	—	Дата начала выполнения задачи
time_end_task	Дата	—	Дата завершения выполнения задачи
table_id	Целочисленный	Обязательное	Идентификатор проекта, к которому относится таблица задач, внешний ключ

Информация о полях в таблице тегов (tag) представлена в таблице 2.7.

Таблица 2.7 – Описание таблицы тегов

Поле	**Тип данных**	**Ограничение**	**Описание**
id_tag	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 255 символов	Название тега
color	Символьный	Максимальная длина — 255 символов	Цвет тега
project_id	Целочисленный	Обязательное	Идентификатор проекта, к которому относится тег, внешний ключ

Информация о полях в таблице связи задач с тегами (task_tag) представлена в таблице 2.8.

Таблица 2.8 – Описание таблицы связи задачи и тега

Поле	**Тип данных**	**Ограничение**	**Описание**
id	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
tag_id	Целочисленный	Обязательное	Идентификатор тега, внешний ключ
task_id	Целочисленный	Обязательное	Идентификатор задачи, внешний ключ

Информация о полях в таблице групп (group_user) представлена в таблице 2.9.

Таблица 2.9 – Описание таблицы групп

Поле	**Тип данных**	**Ограничение**	**Описание**
id	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
name	Символьный	Обязательное, максимальная длина — 255 символов	Название группы
description	Символьный	Максимальная длина — 255 символов	Описание группы
project_id	Целочисленный	Уникальное, обязательное	Идентификатор проекта, внешний ключ

Информация о полях в таблице связи группы и пользователя (group_member) представлена в таблице 2.10.

Таблица 2.10 – Описание таблицы связи группы и пользователя

Поле	**Тип данных**	**Ограничение**	**Описание**
id	Целочисленный	Уникальное, обязательное	Первичный ключ таблицы
role	Символьный	Обязательное, максимальная длина — 255 символов	Роль пользователя в группе
user_id	Целочисленный	Обязательное	Идентификатор пользователя, внешний ключ
group_id	Целочисленный	Обязательное	Идентификатор группы, внешний ключ

На рисунке 2.1 приведена диаграмма разрабатываемой базы данных.

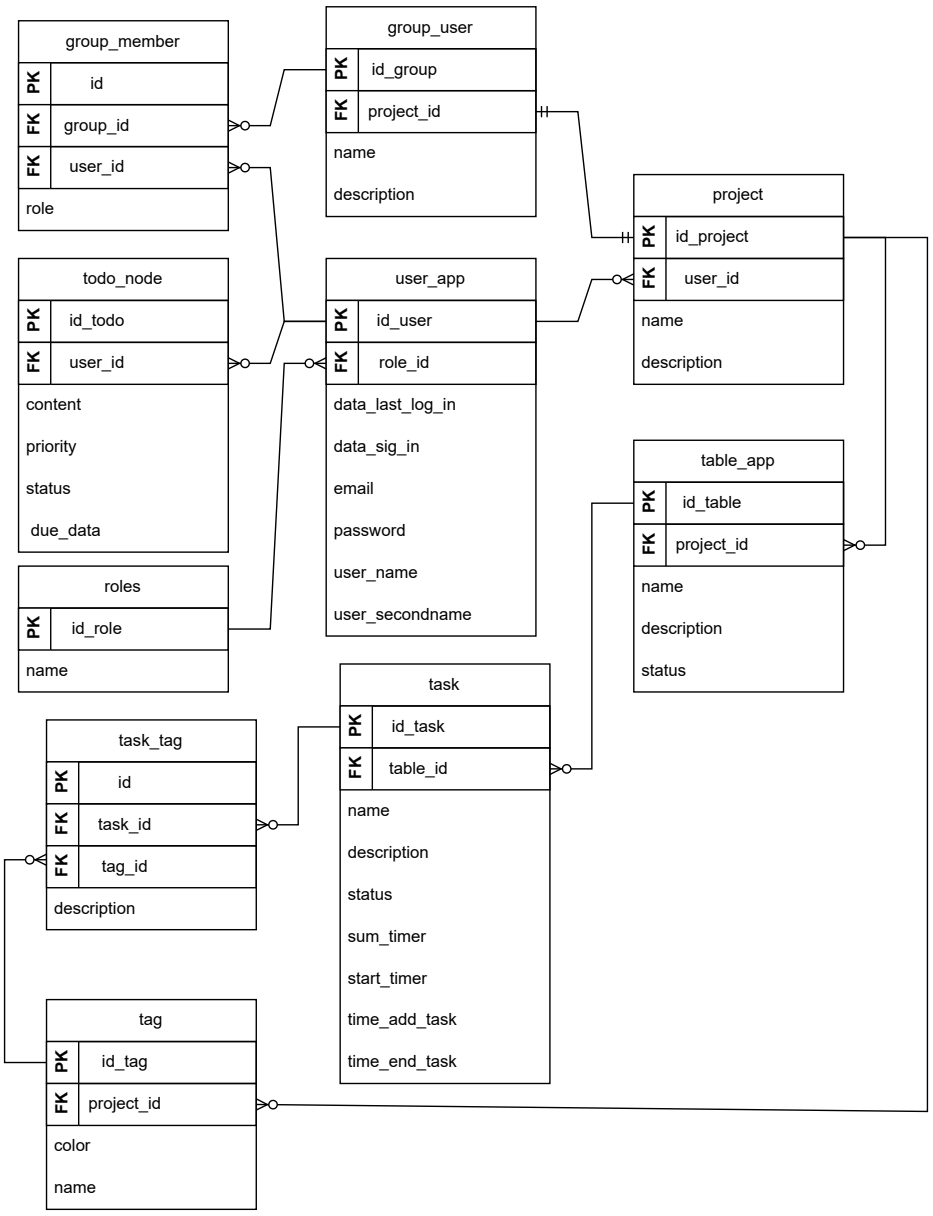


Рисунок 2.1 – Диаграмма разрабатываемой базы данных

2.3 Разработка функций базы данных

Была спроектирована функция БД для триггера, которая производит обновление статуса связанных таблиц задач (task), если статус в таблице карточки (table_app) неактивен. Триггер, использующий данную функцию, будет производить автоматическое обновление после каждого обновления в таблице карточек. Данный триггер обеспечит согласованность и целостность данных. Схема функции приведена на рисунке 2.2.

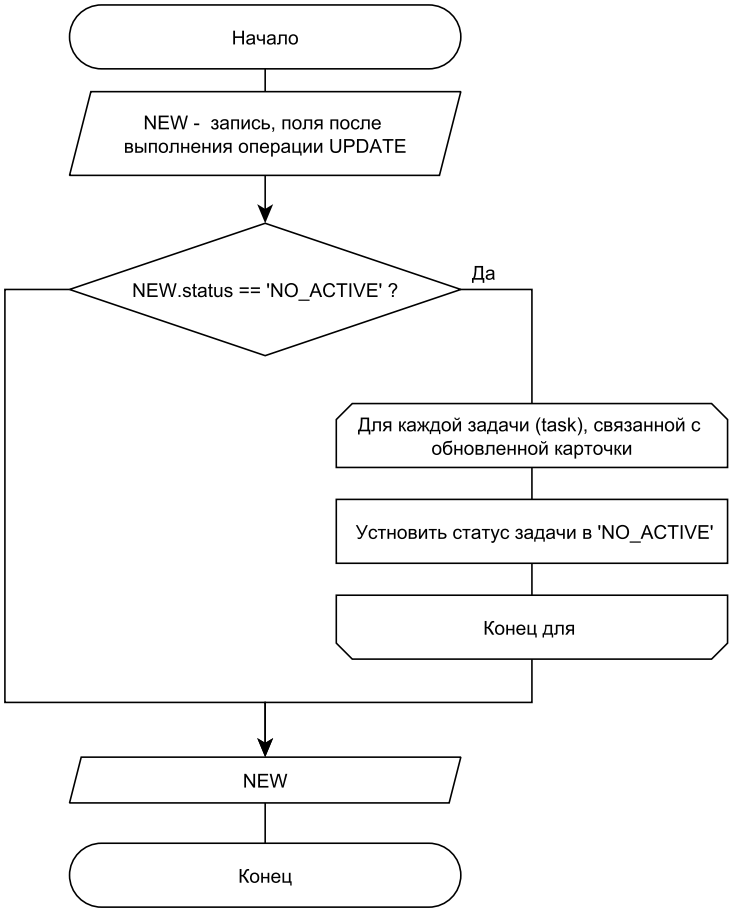


Рисунок 2.2 – Схема алгоритма функции обновления статуса задачи

Также, была спроектирована функция для логирования изменений в БД. В данной функции используется таблица 2.11, для сохранения лога.

Таблица 2.11 – Описание таблицы для сохранения лога

Поле	**Тип данных**	**Описание**
id	Целочисленный	Первичный ключ таблицы
table_name	Символьный	Имя таблицы, в которой произведена операция
operation	Символьный	тип операции (INSERT, UPDATE, DELETE)
data	JSONB	Объект, содержащий детали изменений
logged_at	Дата	Временная метка операции

Схема функции приведена на рисунке 2.3.

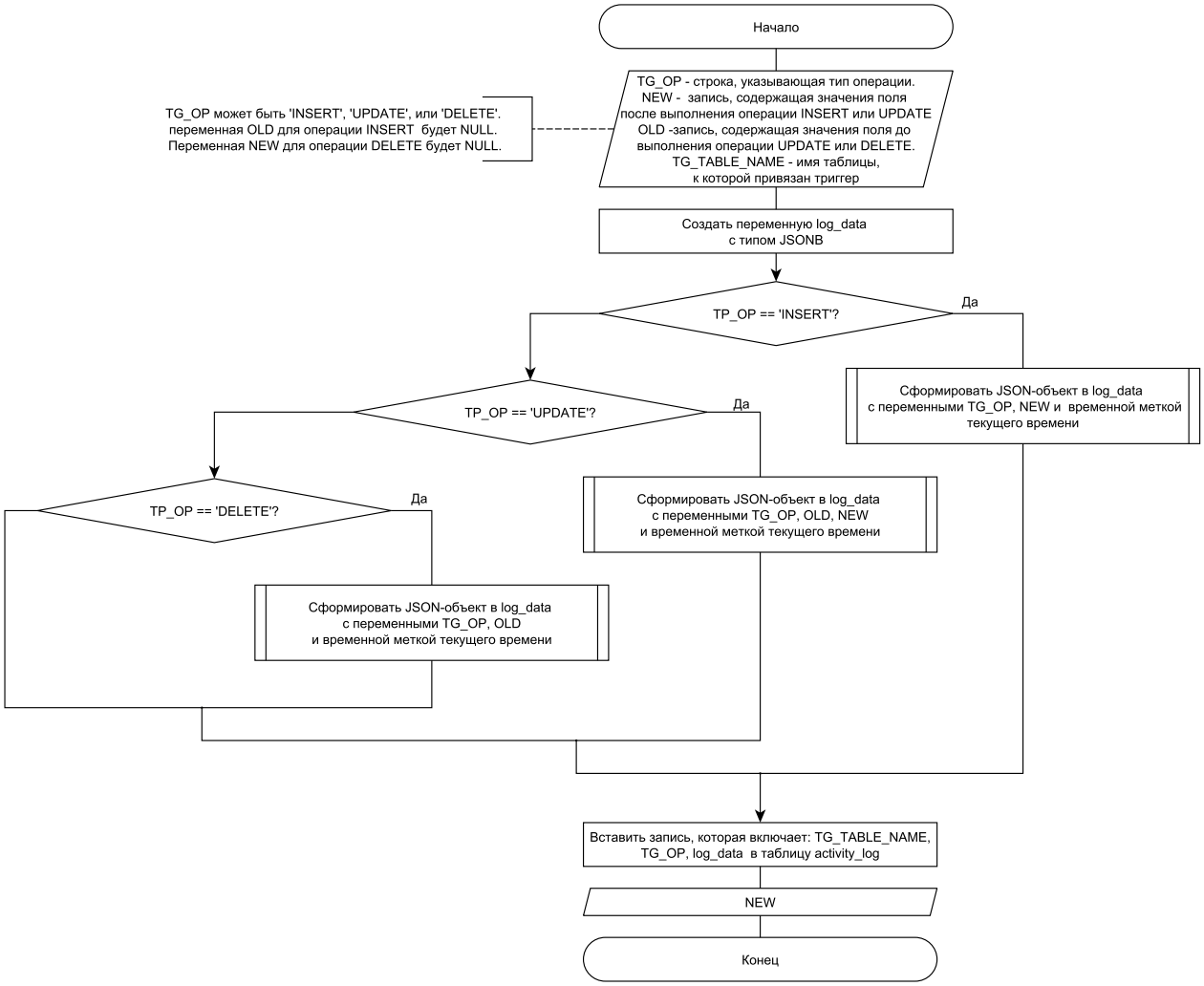


Рисунок 2.3 – Схема алгоритма функции

2.4 Ролевая модель на уровне базы данных

В аналитическом разделе были выделены роли для взаимодействия с приложением (см. таблицу 1.3). На уровне базы данных также было выделено три роли:

- администратор обладает полным доступом ко всем операциям и таблицам базы данных;
- менеджер групп имеет права на выборку, вставку, обновление и удаление данных из всех таблиц, за исключением таблицы пользователя (user_app);
- пользователь имеет права на выборку, вставку, обновление и удаление данных из всех таблиц, за исключением таблицы пользователя и группы (group_user).

3 Технологический раздел

В данном разделе будут представлены средства реализаций, листинги кода для создания таблиц и ограничений целостности БД, реализация функций и ролевой модели БД, а также тестирование и описание интерфейса доступа к БД.

3.1 Средства реализации

Выбор СУБД

В аналитическом разделе была выбрана объектно-реляционная модель данных. Наиболее известные СУБД с поддержкой данной моделью данных: *Informix* [14], *PostgreSQL* [15] и *Oracle* [16].

Таблица 3.1 – Сравнение объектно-реляционных СУБД

Критерий	**Informix**	**PostgreSQL**	**Oracle**
Модель данных	Объектно-реляционная	Объектно-реляционная	Объектно-реляционная
Пользовательские типы	Поддерживается	Поддерживается	Поддерживается
Открытость кода	Проприетарная	Открытый исходный код	Проприетарная
Поддержка JSON	Полная	Полная	Полная
Тип хранения	Клиент-сервер	Клиент-сервер	Клиент-сервер

В качестве СУБД была выбрана *PostgreSQL*. Данный выбор обусловлен следующим:

– эта СУБД поддерживает все типы определенные в ходе проектирования;

– *PostgreSQL* предоставляет необходимые ограничения для обеспечения целостности данных;

– при помощи данной СУБД можно реализовать ролевую модель;

– СУБД работает на основе объектно-реляционной модели данных;

– *PostgreSQL* обладает открытым исходным кодом.

Выбор средства реализации приложения

Для реализации приложения была выбрана клиент-серверная архитектура, она обеспечит четкое разделение между обработкой данных на сервере и их представлением на клиенте. Это разделение позволит управлять бизнес-логикой и данными в одном месте.

Приложение структурировано по шаблону MVC (Model-View-Controller), который делит его на три части: модель, представление и контроллер. Модель управляет данными и бизнес-логикой, представление показывает данные пользователю, а контроллер обрабатывает ввод и координирует работу модели и представления [17].

В качестве элемента взаимодействия между клиентом и сервером был выбран *API*. Этот выбор позволяет стандартизировать доступ к серверным функциям и упрощает интеграцию с другими системами и сервисами [18]. Для реализации и документирования *API* использовался *Swagger* [19].

качестве языка программирования для разработки ПО был выбран язык *Java* [20]. Данный выбор обусловлен следующим:

- *Java* позволяет реализовать клиент-сервисное приложение;
- язык имеет поддержку транзакций *PostgreSQL*;
- *Java* является кроссплатформенным языком, а также имеет обширный выбор библиотек для разработки клиент-сервисного приложения.

3.2 Создание таблиц для сущностей и реализация ограничений целостности базы данных

Ниже описано создание всех необходимых таблиц для сущностей, включая соответствующие ограничения целостности БД.

1) **Пользователи.** В листинге Б.1 приведено создание таблицы *user_app*, которая хранит данные пользователей, включая ограничения целостности.
2) **Дела.** Листинг Б.2 описывает структуру таблицы *todo_node* для сущности дел, с ограничениями на данные.

3) **Проекты.** Листинг Б.3 демонстрирует создание таблицы *project*, включая ограничения целостности для проектов.

4) **Группы и члены групп.** В листинге Б.4 показано создание таблиц *group_user* и *group_member*, а также соответствующие ограничения целостности.

5) **Карточки.** Листинг Б.5 содержит информацию о создании таблицы *table_app* для карточек, включая ограничения.

6) **Задачи.** Создание таблицы *task* описано в листинге Б.6, с подробностями ограничений целостности.

7) **Теги и их связи с задачами.** В Листинге Б.7 описано создание таблиц *tag* и *task_tag*, с необходимыми ограничений на данные.

3.3 Реализация триггеров и функций базы данных

В листинге 3.1 приведена реализация функции для логирования. А также приведена реализация триггра в листинге 3.2, данный триггер будет срабатывать после изменений в таблицах проектов или задач.

Листинг 3.1 – Реализация функции для логирования

```
CREATE OR REPLACE FUNCTION log_activity() RETURNS trigger AS
$$
DECLARE
    log_data JSONB;
BEGIN
    -- Формирование JSON данных для логирования
    IF TG_OP = 'INSERT' THEN
        log_data := jsonb_build_object(
                'operation', TG_OP,
                'new_data', row_to_json(NEW),
                'changed_at', current_timestamp
                    );
    ELSIF TG_OP = 'UPDATE' THEN
        log_data := jsonb_build_object(
                'operation', TG_OP,
                'old_data', row_to_json(OLD),
                'new_data', row_to_json(NEW),
                'changed_at', current_timestamp
                    );
    ELSIF TG_OP = 'DELETE' THEN
        log_data := jsonb_build_object(
                'operation', TG_OP,
                'old_data', row_to_json(OLD),
                'changed_at', current_timestamp
                    );
    END IF;

    -- Вставка логов в таблицу activity_log
    INSERT INTO public.activity_log (table_name, operation, data)
    VALUES (TG_TABLE_NAME, TG_OP, log_data);

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

Листинг 3.2 – Реализация триггера для логирования

```
CREATE TRIGGER log_project_changes
    AFTER INSERT OR UPDATE OR DELETE ON public.project
    FOR EACH ROW EXECUTE FUNCTION log_activity();
CREATE TRIGGER log_task_changes
    AFTER INSERT OR UPDATE OR DELETE ON public.task
    FOR EACH ROW EXECUTE FUNCTION log_activity();
```

В функции 3.1 используется таблица для сохранения лога, реализация создания данной таблицы, приведена в листинге 3.3.

Листинг 3.3 – Создание таблицы для лога

```
CREATE TABLE public.activity_log
(
    id          SERIAL PRIMARY KEY,
    table_name  VARCHAR(255)  NOT NULL,
    operation   VARCHAR(255)  NOT NULL,
    data        JSONB,
    logged_at   TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

В листинге 3.4 приведена реализация функции изменения статуса у задачи, если был изменен статус у карточки, также в этом листиге придена реализация триггера, который будет использовать эту функцию.

Листинг 3.4 – Реализация функции и триггера для изменения статуса задачи

```
CREATE OR REPLACE FUNCTION public.func_update_task_table_status()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.status = 'NO_ACTIVE' THEN
        UPDATE public.task
        SET status = 'NO_ACTIVE'
        WHERE NEW.id_table = public.task.table_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_task_table_status
AFTER UPDATE ON public.table_app
FOR EACH ROW
EXECUTE FUNCTION public.func_update_task_table_status();
```

3.4 Реализация ролевой модели на уровне базы данных

В листинге 3.5 приведена реализация ролевой модели на БД.

Листинг 3.5 – Реализация ролевой модели

```
-- Создание ролей
CREATE ROLE admin_role;
CREATE ROLE manager_role;
CREATE ROLE user_role;

-- Привилегии для admin_role
GRANT ALL PRIVILEGES ON SCHEMA public TO admin_role;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO
   admin_role;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO
   admin_role;


-- Привилегии для manager_role
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO
   manager_role;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA
   public TO manager_role;

-- Исключения для manager_role: отсутствие всех операций над
   таблицей user_app
REVOKE ALL PRIVILEGES ON TABLE public.user_app FROM manager_role;


-- Привилегии для user_role
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO
   user_role;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA
   public TO user_role;

-- Исключения для user_role: отсутствие всех операций над
   таблицами user_app и group_user
REVOKE ALL PRIVILEGES ON TABLE public.user_app FROM user_role;
REVOKE ALL PRIVILEGES ON TABLE public.group_user FROM user_role;
```

3.5 Тестирование

Были протестированы функции и триггеры работающие с этими функциями.

Для функции, которая изменяет статус задачи при изменение статуса карточки, было выделено два класса эквивалентности: статус карточки изменился, статус карточки не изменился, но произошло изменение таблицы. Тесты для данной функции приведены ниже.

1) Начальное состояние таблицы карточки приведено в таблице 3.2. Начальное состояние таблицы задач приведено в таблице 3.3. Состояние таблицы, после изменения статуса карточки, представлено в таблице 3.4. Состояние таблицы задачи, после изменения статуса карточки, представлено в таблице 3.5. *Итог*, фактический результат совпал с ожидаемым, тест пройден.

2) Начальное состояние таблицы карточки приведено в таблице 3.6. Начальное состояние таблицы задач приведено в таблице 3.7. Состояние таблицы после изменения имени карточки, представлено в таблице 3.8. Состояние таблицы задачи после изменения имени карточки, представлено в таблице 3.9. *Итог*, фактический результат совпал с ожидаемым, тест пройден.

Таблица 3.2 – Начальное состояние таблицы карточки для первого теста

id_table	**description**	**name**	**status**	**project_id**
1	Table for tracking issues	Issue	ACTIVE	1
2	Sample Description	Sample Table	ACTIVE	1

Таблица 3.3 – Начальное состояние таблицы задачи для первого теста

id_task	**description**	**name**	**status**	**...**	**table_id**
1	Fix all UI bugs	UI Bugfix	ACTIVE	...	1

Таблица 3.4 – Состояние таблицы карточки после изменения статуса для первого теста

id_table	**description**	**name**	**status**	**project_id**
1	Table for tracking issues	Issue	NO_ACTIVE	1
2	Sample Description	Sample Table	ACTIVE	1

Таблица 3.5 – Состояние таблицы задачи после изменения для первого теста

id_task	**description**	**name**	**status**	**...**	**table_id**
1	Fix all UI bugs	UI Bugfix	NO_ACTIVE	...	1

Таблица 3.6 – Начальное состояние таблицы карточки для второго теста

id_table	**description**	**name**	**status**	**project_id**
1	Table for tracking issues	Issue	ACTIVE	1
2	Sample Description	Sample Table	ACTIVE	1

Таблица 3.7 – Начальное состояние таблицы задачи для второго теста

id_task	**description**	**name**	**status**	**...**	**table_id**
1	Fix all UI bugs	UI Bugfix	ACTIVE	...	1

Таблица 3.8 – Состояние таблицы карточки после изменения имени для второго теста

id_table	**description**	**name**	**status**	**project_id**
1	New-Name	Issue	ACTIVE	1
2	Sample Description	Sample Table	ACTIVE	1

Таблица 3.9 – Состояние таблицы задачи после изменения для второго теста

id_task	**description**	**name**	**status**	**...**	**table_id**
1	Fix all UI bugs	UI Bugfix	ACTIVE	...	1

Для функции и триггера, который производит логирование при любом изменение в таблицах проекта или в таблицах задач, было выделены следующие классы эквивалентности: произведена операция *INSERT* в таблицу задач или проекта, произведена операция *UPDATE* в таблицу задач или проекта, произведена операция *DELETE* в таблицу задач или проекта, произведена любая из выше перечисленных операций другие таблицы. Тесты для данной функции приведены ниже, для этих тестов использовался код представленный в листинге Б.8 .

1) Действие: произведены операции *INSERT*, *UPDATE*, *DELETE* над таблицей проектов, состояние таблицы лога после данных действий приведено в таблице В.1. *Итог*, после каждой операции над таблицей проектов, было произведено сохранение в таблицу лога, тест пройден.

2) Действие: произведены операции *INSERT*, *UPDATE*, *DELETE* над таблицей задач, состояние таблицы лога, после данных операций приведено в таблице В.2. *Итог*, после каждой операции над таблицей задач, было произведено сохранение в таблицу лога, тест пройден.

3) Действие: произведены операции *INSERT*, *UPDATE*, *DELETE* над таблицей карточек, состояние таблицы лога не изменилось,тест пройден.

3.6 Интерфейс доступа к базе данных

На рисунках 3.1–3.8 представлен программный интерфейс, обеспечивающий доступ к БД.

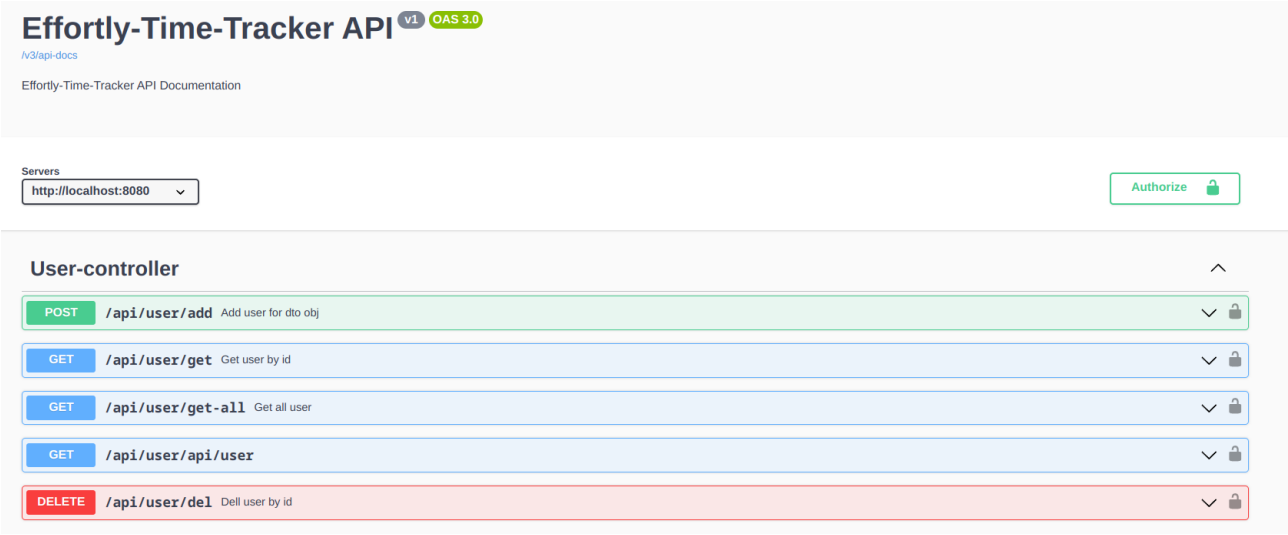


Рисунок 3.1 – Демонстрация интерейса

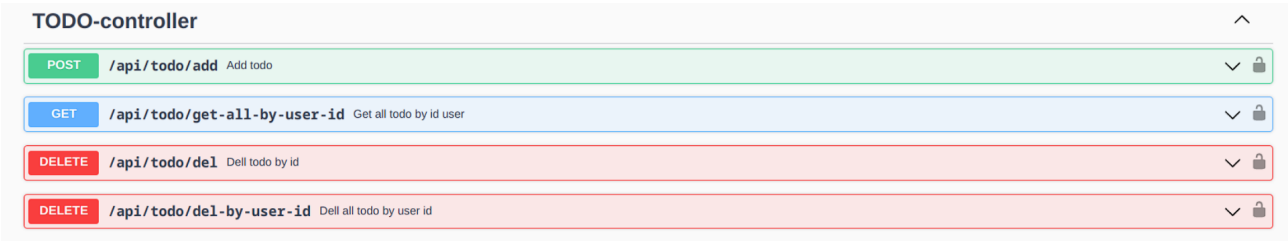


Рисунок 3.2 – Демонстрация интерейса

Рисунок 3.3 – Демонстрация интерейса

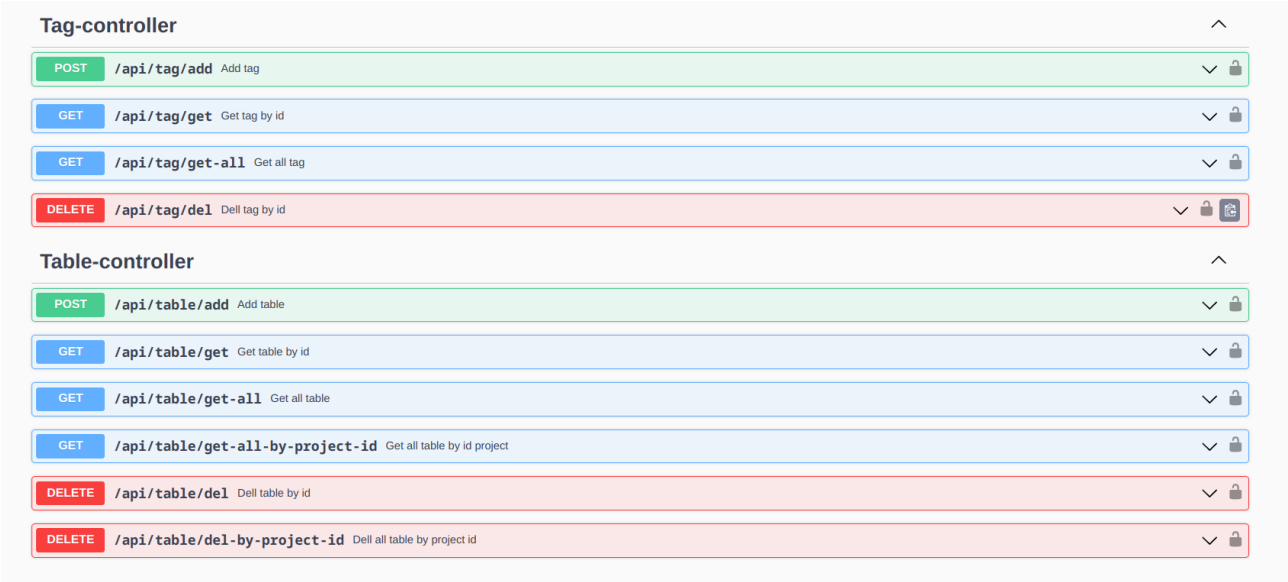


Рисунок 3.4 – Демонстрация интерейса

Рисунок 3.5 – Демонстрация интерейса

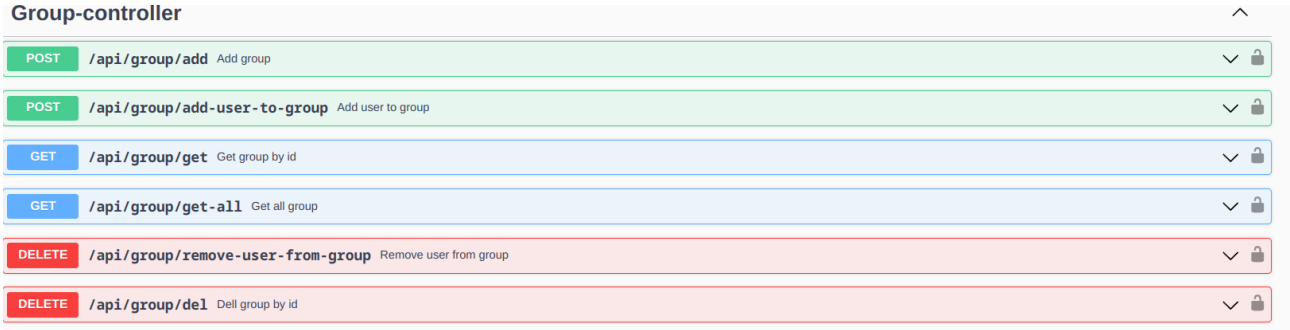


Рисунок 3.6 – Демонстрация интерейса

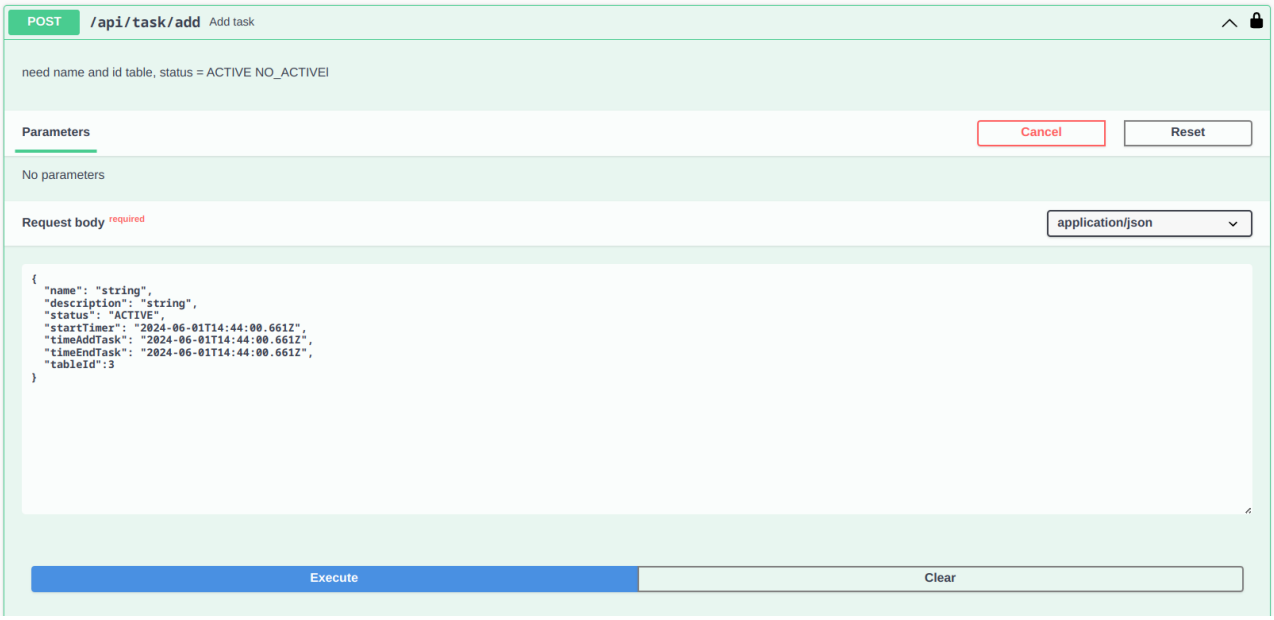


Рисунок 3.7 – Пример входных данных для запроса содания задачи

Рисунок 3.8 – Полученный ответ после запроса создания задачи

4 Исследовательская часть

В данном разделе будет представлено исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

– Процессор: Ryzen 5 4600H, 6 процессорных ядер архитектуры Zen 2 и 12 потоков, работающих на базовой частоте в 3.0 ГГц (до 4.0 ГГц в Turbo режиме), 12 логических ядер [21].

– Оперативная память: 16 ГБайт.

– Операционная система: Windows 10 Pro 64-разрядная система [22].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями. Каждый замер проводился 100 раз, после чего рассчитывалось их среднее арифметическое значение.

4.2 Времени выполнения запросов

Для исследования зависимости времени выполнения запросов с использованием индексов и без них, было выбрана два типа индекса: B-дерево и Хеш-индекс [15]. Были выбраны столбцы *email* и *id_user* для создания индексов, в листинге 4.1 представлен код, который создает для данных столбцов индекс.

Листинг 4.1 – Создание индексов

```
CREATE INDEX b-tree-id-user-indx ON user_app USING BTREE
   (id_user)
CREATE INDEX b-tree-id-user-indx ON user_app USING BTREE  (email)

CREATE INDEX b-tree-id-user-indx ON user_app  USING HASH
   (id_user)
CREATE INDEX b-tree-id-user-indx ON user_app  USING HASH (email)
```

Результаты замеров времени выполнения запросов зависимости от числа записей в БД при использование индекса на столбец *id* и без него, приведены в таблице В.3. На рисунке 4.1 изображен график для этой таблицы. Листинг запроса Б.9 для данного замера.

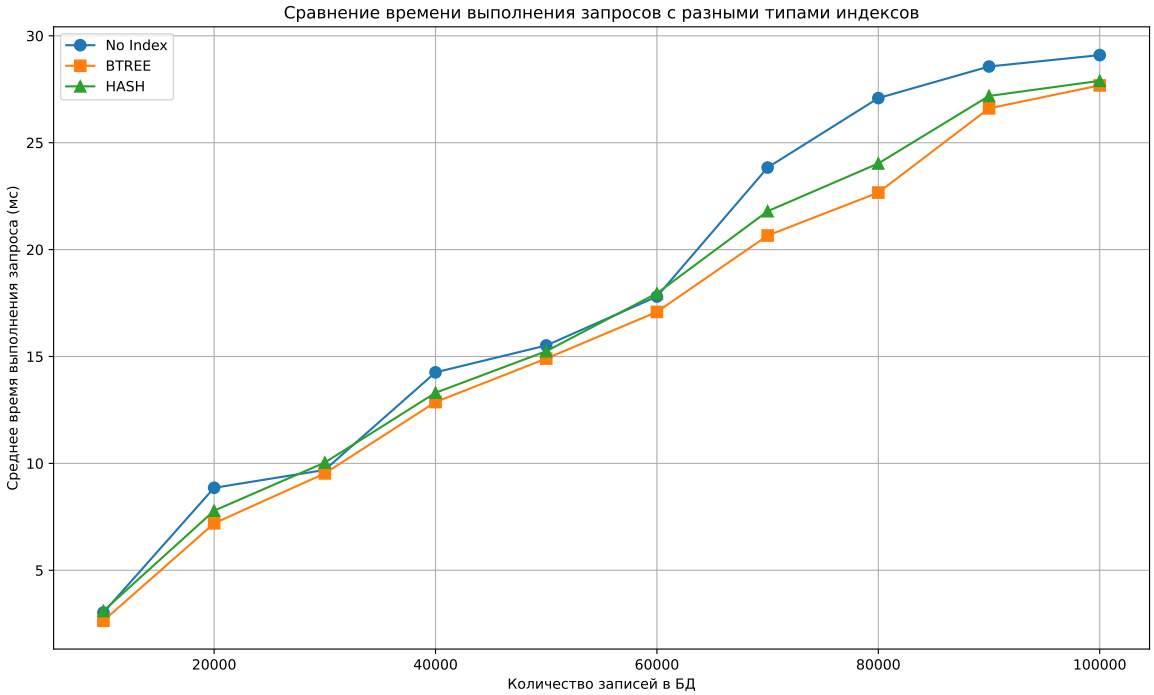


Рисунок 4.1 – График зависимости времени выполнения запросов от числа записей в БД (индекс для столбца id)

Результаты замеров времени выполнения запросов в зависимости от числа записей в БД при использование индекса на столбец *email* и без него, приведены в таблице В.4. На рисунке 4.2 изображен график для данной таблицы. Листинг запроса Б.10 для данного замера.

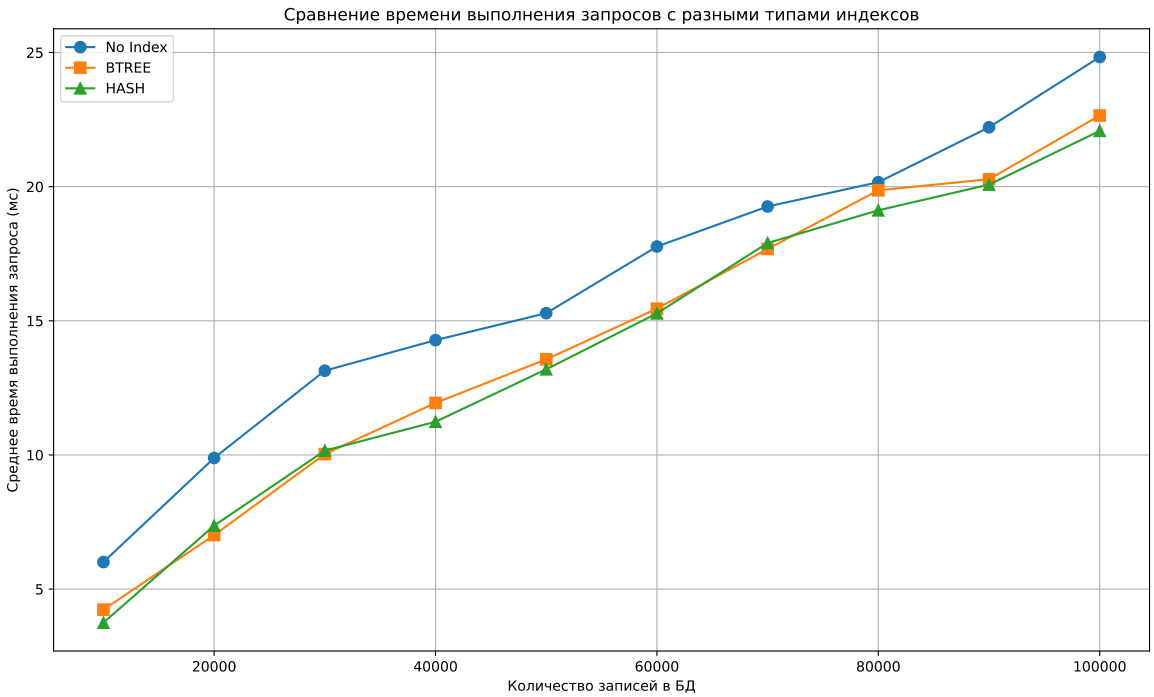


Рисунок 4.2 – График зависимости времени выполнения запросов от числа записей в БД (индекс для столбца email)

Вывод

В результате исследования можно сделать следующие выводы.

– С увеличением количествам записей в БД, время выполнения запроса увеличивается.

– Из рисунка 4.1 видно, что при выполнении запроса Б.9 использование B-дерева для индексации столбца *id* оказывается наиболее эффективной. При 100000 записей в каждой таблице, B-дерево на 4.87% быстрее по сравнению с отсутствием индекса и на 0.77% быстрее по сравнению с хеш-индексом (см. таблицу В.3).

– Из рисунка 4.2 видно, что при выполнении запроса Б.10 использование хеш-индекса для столбца *email* оказывается наиболее эффективной. При

100000 записей в каждой таблице, хеш-индекс на 11.08% быстрее по сравнению с отсутствием индекса и на 2.53% быстрее по сравнению с B-деревом (см. таблицу B.4).

В результате проделанного исследования можно сделать вывод, что B-деревья наиболее эффективно обрабатывают запросы с диапазонным значением. А хеш-индексы, в свою очередь эффективнее в запросах с точным совпадением.

ЗАКЛЮЧЕНИЕ

Цель курсовой работы достигнута, разработана база данных для хранения и обработки данных приложения учета и аудита времени, потраченного на рабочие и личные задачи.

Для достижения поставленной цели были выполнены следующие задачи:

- сформулировано описание пользователей проектируемого приложения и проведен анализ существующих решений;
- спроектирована БД для учета и аудита времени;
- реализована БД;
- проведен исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД.

В результате исследования можно сделать следующие выводы.

- С увеличением количествам записей в БД, время выполнения запроса увеличивается.
- Из рисунка 4.1 видно, что при выполнении запроса Б.9 использование B-дерева для индексации столбца *id* оказывается наиболее эффективной. При 100000 записей в каждой таблице, B-дерево на 4.87% быстрее по сравнению с отсутствием индекса и на 0.77% быстрее по сравнению с хеш-индексом (см. таблицу В.3).
- Из рисунка 4.2 видно, что при выполнении запроса Б.10 использование хеш-индекса для столбца *email* оказывается наиболее эффективной. При 100000 записей в каждой таблице, хеш-индекс на 11.08% быстрее по сравнению с отсутствием индекса и на 2.53% быстрее по сравнению с B-деревом (см. таблицу В.4).

В результате проделанного исследования можно сделать вывод, что B-деревья наиболее эффективно обрабатывают запросы с диапазонным значением. А хеш-индексы, в свою очередь эффективнее в запросах с точным совпадением.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Хомопепко А. Д., Цыганков В. М., Мальцев М. Г.* Базы данных, учебник для высших учебных заведений. — 2009. — С. 1—734.
2. Time Management - A Case Study [Электронный ресурс]. — Режим доступа: `https://www.researchgate.net/publication/323825275_Time_Management_-_A_Case_Study` (дата обращения: 17.03.2024).
3. Introducing the Eisenhower Matrix [Электронный ресурс]. — Режим доступа: `https://www.eisenhower.me/eisenhower-matrix/` (дата обращения: 17.03.2024).
4. *Брайан Трейси.* Тайм-менеджмент. — 2015. — С. 1—144.
5. Clockify. — Режим доступа: `https://clockify.me/` (дата обращения: 30.05.2024).
6. Toggl. — Режим доступа: `https://toggl.com/` (дата обращения: 30.05.2024).
7. Timecamp. — Режим доступа: `https://www.timecamp.com/` (дата обращения: 30.05.2024).
8. Trello. — Режим доступа: `https://trello.com/` (дата обращения: 30.05.2024).
9. *Шустова Л. И., Тараканов О. В.* Базы данных. — 2016. — С. 1—304.
10. *Жалолов О. И., Хаятов Х. У.* Понятие SQL и реляционной базы данных // Universum: Технические науки: электрон. научн. журн. — 2020. — Т. Выпуск 6 (75). — С. 27—29.
11. *Парфенов Ю. П.* Постреляционные хранилища данных. — 2016. — С. 1—120.
12. *Емельченков Е. П., Савченков К. П.* объектно-ориентированный подход к представлению баз данных // системы компьютерной математики и их приложения. — 2016. — С. 47—48.
13. *Сорокин В. Е., Хаятов Х. У.* Об эффективности наследования таблиц в СУБД PostgreSQL // НИИ «Центрпрограммсистем». — 2016. — С. 15—23.
14. IBM Informix. — Режим доступа: `https://www.ibm.com/products/informix` (дата обращения: 30.05.2024).

15. PostgreSQL 16.3 Documentation. — Режим доступа: https://www.postgresql.org/docs/current/index.html (дата обращения: 30.05.2024).

16. Oracle Database. — Режим доступа: https://www.oracle.com/ (дата обращения: 30.05.2024).

17. *Ищанова С. Г.* Применение схемы разделения данных и управляющей логики MVC в разработке веб-приложений // III Международная научно-практическая конференция «Современные стратегии и цифровые трансформации устойчивого развития общества, образования и науки». — 2024. — С. 69—72.

18. *Кошевой С. Р.* api как способ обслуживания клиентов // главные характеристики современного этапа развития мировой науки. — 2018. — С. 76—79.

19. Swagger RESTful API Documentation Specificationn. — Режим доступа: https://docs.swagger.io/spec.html (дата обращения: 31.05.2024).

20. Python 3.12.3 documentation. — Режим доступа: https://docs.oracle.com/en/java/javase/17/index.html (дата обращения: 31.05.2024).

21. Ryzen 4600H. — Режим доступа: https://www.amd.com/en/products/apu/amd-ryzen-5-4600h (дата обращения: 20.05.2024).

22. Windows 10 Pro 2h21 64-bit. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 25.05.2024).

ПРИЛОЖЕНИЕ А

Рисунок А.1 – Презентация, слайд первый

Цель и задачи работы

Цель работы — разработка базы данных для хранения и обработки данных приложения учета и аудита времени, потраченного на рабочие и личные задачи.

- Задачи:
  - сформулировать описание пользователей проектируемого приложения и провести анализ существующих решений
  - спроектировать БД для учета и аудита времени
  - реализовать БД
  - провести исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД

Рисунок А.2 – Презентация, слайд второй

Анализ аналогичных решений

Критерий	**Clockify**	**Toggl**	**TimeCamp**	**Trello**
Замер времени	Есть	Есть	Есть	Нет
Количество проектов	Неограниченно	Неограниченно	Ограничено	Ограничено
Аналитика	Есть	Есть	Есть	Нет
Количество пользователей	Неограниченно	Ограничено	Ограничено	Ограничено
Канбан-доски	Нет	Нет	Нет	Есть

Рисунок А.3 – Презентация, слайд третий

Формализация и описание пользователей

Функциональность	администратор	менеджер групп	пользователь
Регистрация и аутентификация	+	+	+
Добавить и удалить пользователя	+		
Получить информацию о пользователях	+		
Создать или изменить дело, проект, карточку, задачу или тег	+	+	+
Получить информацию о списке дел, проекте, карточках, задачах и тегах	+	+	+
Замерить время для конкретной задачи	+	+	+
Получить аналитику по проекту	+	+	+
Создать или изменить группу	+	+	

4

Рисунок А.4 – Презентация, слайд четвертый

Виды пользователей приложения

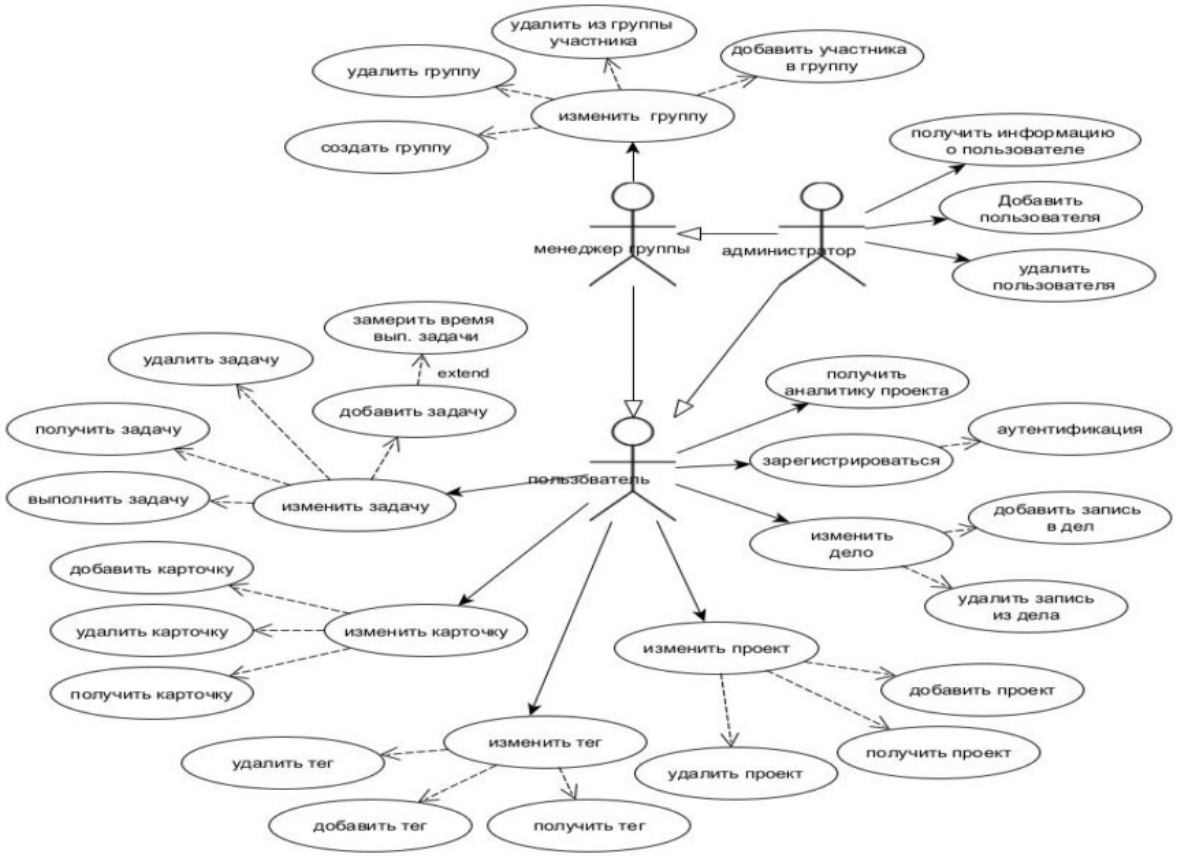


Рисунок А.5 – Презентация, слайд пятый

Рисунок А.6 – Презентация, слайд шестой

Диаграмма БД

Таблицы базы данных:

- user_app — таблица пользователей
- todo_node — таблица дел
- project — таблица проектов
- roles — таблица ролей
- group_user — таблица групп пользователей
- group_member — таблица для связи группы с пользователем
- table_app — таблица карточек
- tag — таблица тегов
- task — таблица задач
- task_tag — таблица для связи задач с тегами

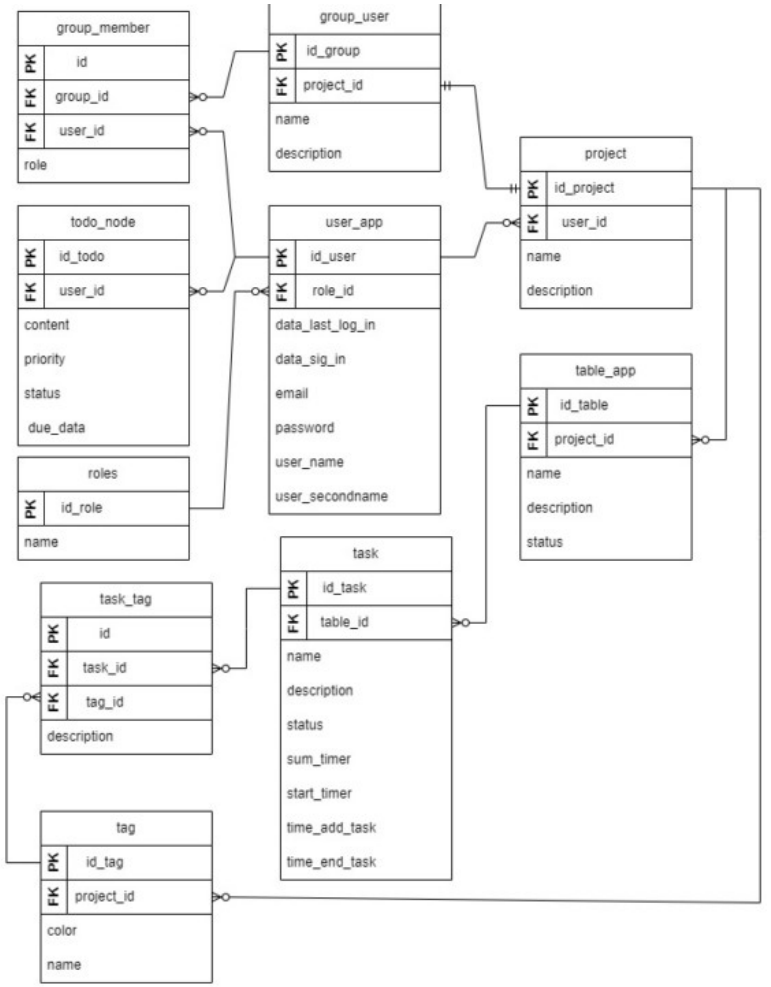


Рисунок А.7 – Презентация, слайд седьмой

Рисунок А.8 – Презентация, слайд восьмой

Интерфейс доступа к базе данных

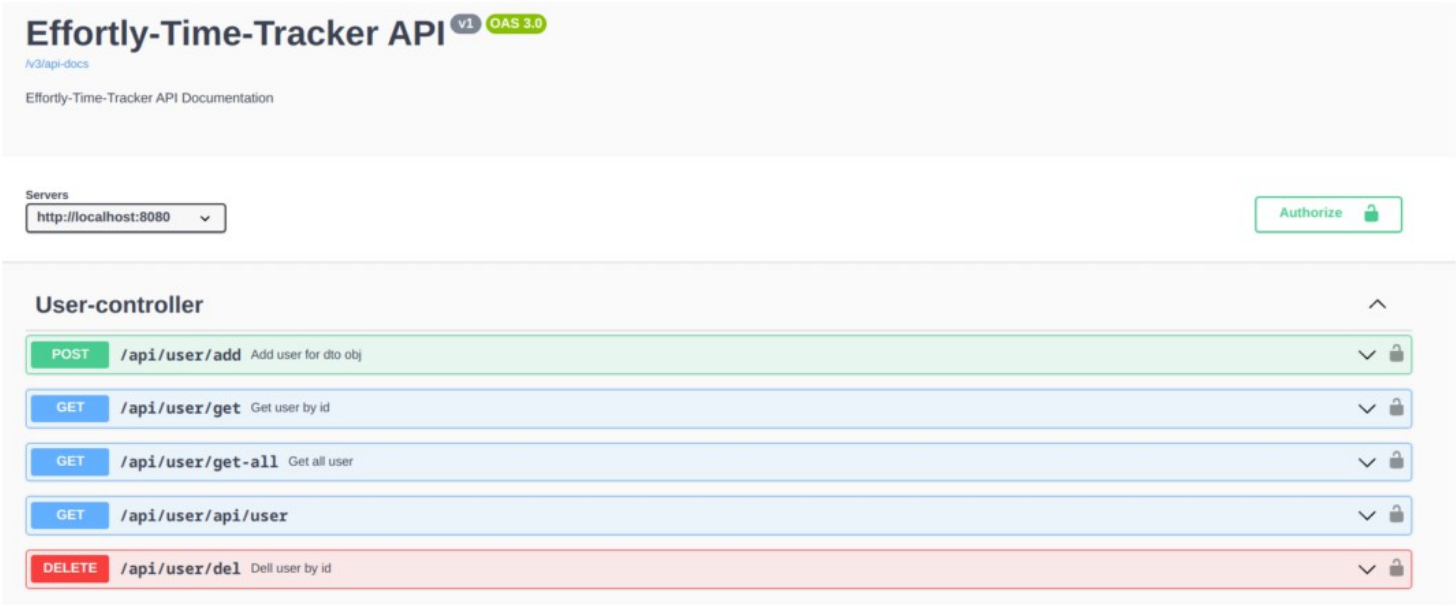


Рисунок А.9 – Презентация, слайд девятый

Интерфейс доступа к базе данных

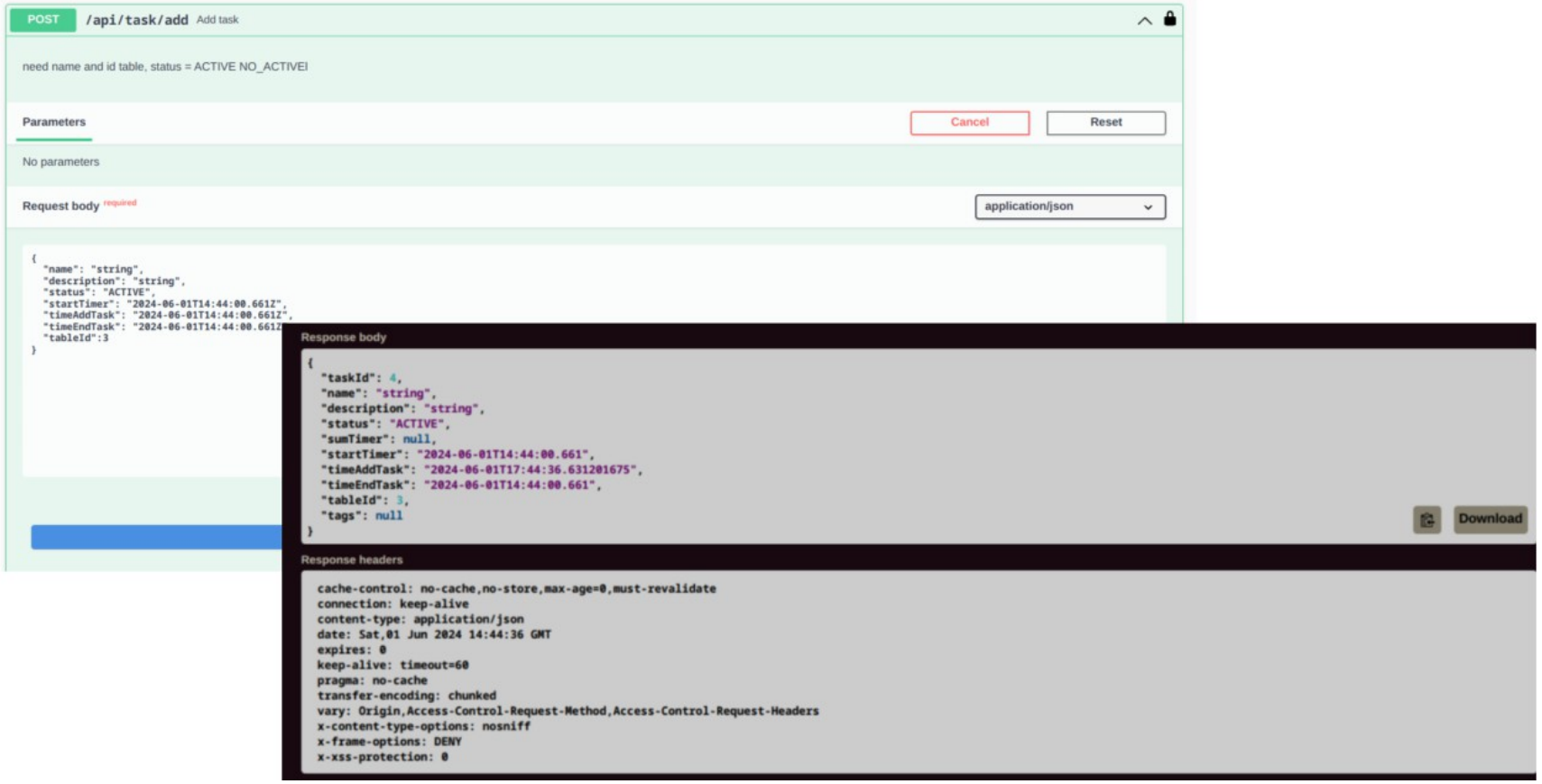
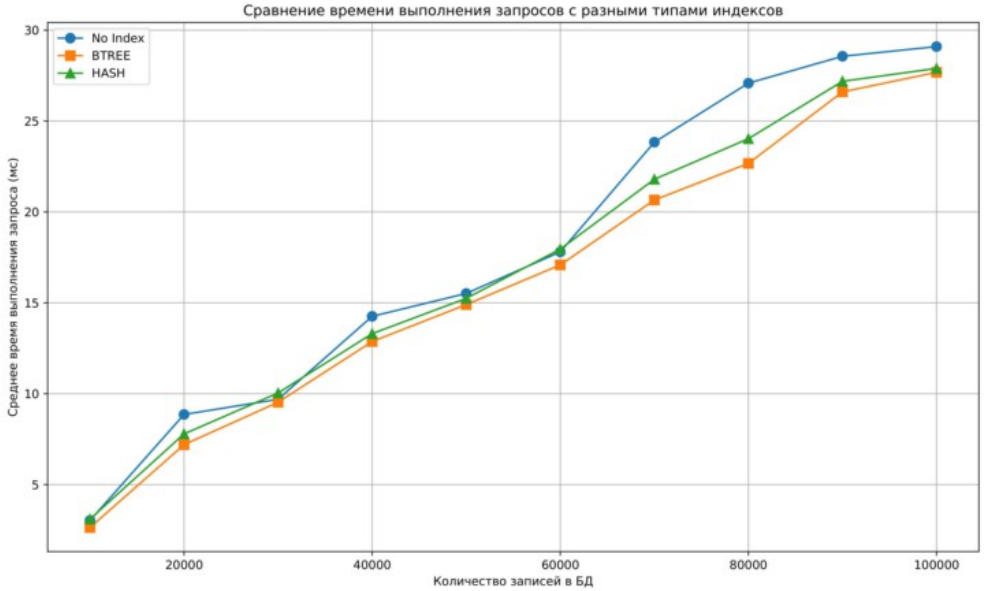


Рисунок А.10 – Презентация, слайд десятый

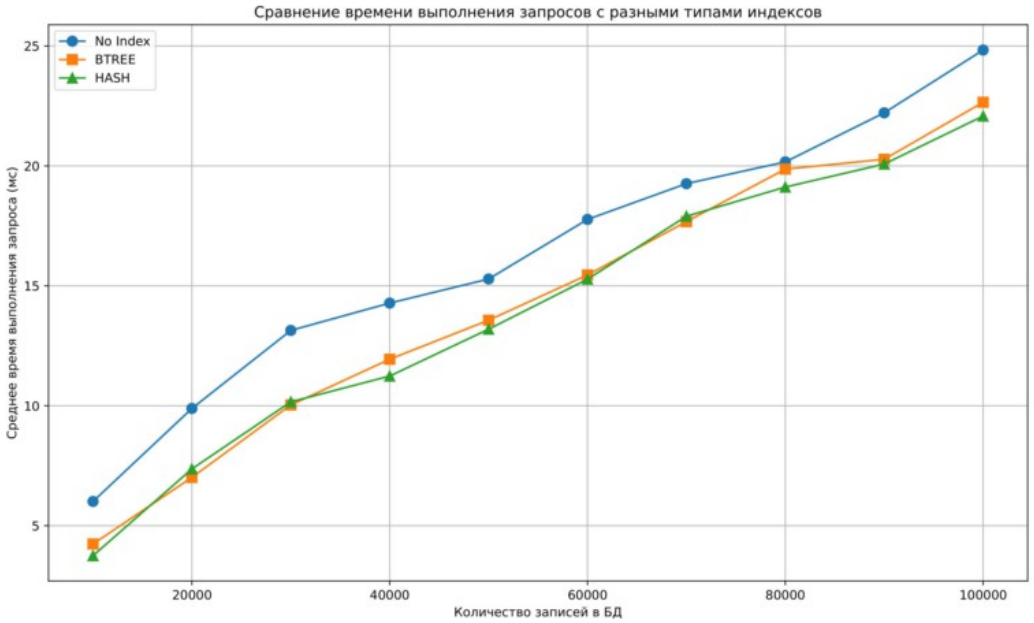
Исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД



Использование индекса для столбца id

Рисунок А.11 – Презентация, слайд одиннадцатый

Исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД



Использование индекса для столбца email

Рисунок А.12 – Презентация, слайд двенадцатый

Заключение

Цель курсовой работы достигнута, разработана база данных для хранения и обработки данных приложения учета и аудита времени, потраченного на рабочие и личные задачи

- Для достижения поставленной цели были выполнены следующие задачи:
  - сформулировано описание пользователей проектируемого приложения и проведен анализ существующих решений
  - спроектирована БД для учета и аудита времени
  - реализована БД
  - проведено исследование зависимости времени выполнения запросов с использованием индексов и без них от количества записей в БД

Рисунок А.13 – Презентация, слайд тринадцатый

Рисунок А.14 – Презентация, слайд четырнадцатый

ПРИЛОЖЕНИЕ Б

Листинг Б.1 – Реализация сущности и ограничения целостности для сущности пользователя

```
CREATE TABLE public.user_app
(
    id_user           SERIAL       NOT NULL,
    PRIMARY KEY (id_user),

    data_last_log_in TIMESTAMP(6),
    data_sign_in     TIMESTAMP(6),
    email            VARCHAR(255) NOT NULL,
    password         VARCHAR(255) NOT NULL,
    user_name        VARCHAR(255) NOT NULL,
    user_secondname  VARCHAR(255) NOT NULL,

    role_id          INTEGER NOT NULL,
    FOREIGN KEY (role_id) REFERENCES public.roles (id_role)
);

ALTER TABLE public.user_app
    ADD CONSTRAINT uniq_user_app_email UNIQUE (email);
ALTER TABLE public.user_app
    ADD CONSTRAINT chk_user_app_email_length CHECK
       (CHAR_LENGTH(email) <= 255);
ALTER TABLE public.user_app
    ADD CONSTRAINT chk_user_app_password_length CHECK
       (CHAR_LENGTH(password) <= 255);
ALTER TABLE public.user_app
    ADD CONSTRAINT chk_user_app_user_name_length CHECK
       (CHAR_LENGTH(user_name) <= 255);
ALTER TABLE public.user_app
    ADD CONSTRAINT chk_user_app_user_secondname_length CHECK
       (CHAR_LENGTH(user_secondname) <= 255);
```

Листинг Б.2 – Реализация сущности и ограничения целостности для сущности дела

```
CREATE TABLE public.todo_node
(
    id_todo  SERIAL NOT NULL,
```

```
    PRIMARY KEY (id_todo),

    content  VARCHAR(255),
    priority VARCHAR(32),
    status   VARCHAR(16),
    due_data TIMESTAMP(6),

    user_id  INTEGER NOT NULL,
    FOREIGN KEY (user_id) REFERENCES public.user_app (id_user)
);

ALTER TABLE public.todo_node
    ADD CONSTRAINT chk_todo_node_content_length CHECK
       (CHAR_LENGTH(content) <= 255);

ALTER TABLE public.todo_node
    ADD CONSTRAINT chk_todo_node_priority CHECK (priority IN
       ('IMPORTANT_URGENTLY',
       'NO_IMPORTANT_URGENTLY','IMPORTANT_NO_URGENTLY',
       'NO_IMPORTANT_NO_URGENTLY'));

ALTER TABLE public.todo_node
    ADD CONSTRAINT chk_todo_node_status CHECK (status IN
       ('NO_ACTIVE', 'ACTIVE'));
```

Листинг Б.3 – Реализация сущности и ограничения целостности для сущности проекта

```
CREATE TABLE public.project
(
    id_project  SERIAL       NOT NULL,
    PRIMARY KEY (id_project),

    name        VARCHAR(255) NOT NULL,
    description VARCHAR(255),

    user_id     INTEGER NOT NULL ,
    FOREIGN KEY (user_id) REFERENCES public.user_app (id_user)
);

ALTER TABLE public.project
    ADD CONSTRAINT chk_project_name_length CHECK
       (CHAR_LENGTH(name) <= 255);
```

```
ALTER TABLE public.project
    ADD CONSTRAINT chk_project_description_length CHECK
       (CHAR_LENGTH(description) <= 255);
```

Листинг Б.4 – Реализация group_user и group_member

```
CREATE TABLE public.group_user
(
    id_group    SERIAL       NOT NULL,
    PRIMARY KEY (id_group),
    name        VARCHAR(255) NOT NULL,
    description VARCHAR(255),
    project_id  INTEGER UNIQUE NOT NULL,
    FOREIGN KEY (project_id) REFERENCES public.project
       (id_project)
);

ALTER TABLE public.group_user
    ADD CONSTRAINT chk_group_name_length CHECK
       (CHAR_LENGTH(name) <= 255);
ALTER TABLE public.group_user
    ADD CONSTRAINT chk_group_description_length CHECK
       (CHAR_LENGTH(description) <= 255);


CREATE TABLE public.group_member
(
    id       SERIAL  NOT NULL,
    PRIMARY KEY (id),
    group_id INTEGER NOT NULL,
    user_id  INTEGER NOT NULL,
    role     VARCHAR(16) NOT NULL,
    FOREIGN KEY (group_id) REFERENCES public.group_user
       (id_group),
    FOREIGN KEY (user_id) REFERENCES public.user_app (id_user)
);

ALTER TABLE public.group_member
    ADD CONSTRAINT chk_group_member_role CHECK (role IN
       ('ADMIN', 'MANAGER', 'USER'));
```

Листинг Б.5 – Реализация сущности и ограничения целостности для сущности карточки

```
CREATE TABLE public.table_app
(
    id_table    SERIAL       NOT NULL,
    PRIMARY KEY (id_table),

    description VARCHAR(255),
    name        VARCHAR(255) NOT NULL,
    status      VARCHAR(255) NOT NULL,

    project_id  INTEGER NOT NULL,
    FOREIGN KEY (project_id) REFERENCES project (id_project)
);

ALTER TABLE public.table_app
    ADD CONSTRAINT chk_table_name_length CHECK
       (CHAR_LENGTH(name) <= 255);

ALTER TABLE public.table_app
    ADD CONSTRAINT chk_table_description_length CHECK
       (CHAR_LENGTH(description) <= 255);

ALTER TABLE public.table_app
    ADD CONSTRAINT chk_table_status CHECK (status IN
       ('NO_ACTIVE', 'ACTIVE'));
```

Листинг Б.6 – Реализация сущности и ограничения целостности для сущности задачи

```
CREATE TABLE public.task
(
    id_task       SERIAL       NOT NULL,
    PRIMARY KEY (id_task),

    description   VARCHAR(255),
    name          VARCHAR(255) NOT NULL,
    status        VARCHAR(16) NOT NULL,

    sum_timer     BIGINT,
    start_timer   TIMESTAMP(6),
    time_add_task TIMESTAMP(6),
```

```
    time_end_task TIMESTAMP(6),

    table_id      INTEGER NOT NULL ,
    FOREIGN KEY (table_id) REFERENCES public.table_app (id_table)
);

ALTER TABLE public.task
    ADD CONSTRAINT chk_task_name_length CHECK (CHAR_LENGTH(name)
       <= 255);

ALTER TABLE public.task
    ADD CONSTRAINT chk_task_description_length CHECK
       (CHAR_LENGTH(description) <= 255);

ALTER TABLE public.task
    ADD CONSTRAINT chk_table_status CHECK (status IN
       ('NO_ACTIVE', 'ACTIVE'));
```

Листинг Б.7 – Реализация tag и task_tag

```
CREATE TABLE public.tag
(
    id_tags    SERIAL       NOT NULL,
    PRIMARY KEY (id_tags),

    color      VARCHAR(255),
    name       VARCHAR(255) NOT NULL,

    project_id INTEGER NOT NULL ,
    FOREIGN KEY (project_id) REFERENCES project (id_project)
);

ALTER TABLE public.tag
    ADD CONSTRAINT chk_tag_name_length CHECK (CHAR_LENGTH(name)
       <= 255);

ALTER TABLE public.tag
    ADD CONSTRAINT chk_tag_color_length CHECK
       (CHAR_LENGTH(color) <= 255);


CREATE TABLE public.task_tag
(
```

```
    id      SERIAL  NOT NULL,
    PRIMARY KEY (id),

    tag_id  INTEGER NOT NULL,
    task_id INTEGER NOT NULL,
    FOREIGN KEY (tag_id) REFERENCES public.tag (id_tags),
    FOREIGN KEY (task_id) REFERENCES public.task (id_task)
);
```

Листинг Б.8 – Тесты для функции логирования

```
INSERT INTO public.project (name, description, user_id) VALUES
   ('Test Project', 'This is a test project', 1);
UPDATE public.project SET description = 'Updated description'
   WHERE name = 'Test Project';
DELETE FROM public.project WHERE name = 'Test Project';

INSERT INTO public.table_app (description, name, status,
   project_id)
VALUES ('Sample Description', 'Sample Table', 'ACTIVE', 1);

INSERT INTO public.task (description, name, status, start_timer,
   time_add_task, time_end_task, table_id)
VALUES ('Test task', 'Test', 'NO_ACTIVE', NOW(), NOW(), NOW() +
   INTERVAL '1 day', 1);
UPDATE public.task SET status = 'ACTIVE' WHERE name = 'Test';
DELETE FROM public.task WHERE name = 'Test';
```

Листинг Б.9 – Запрос для замера

```
SELECT
    ua.user_name, ua.email,
    p.name AS project_name, p.description AS project_description,
    ta.name AS table_name, ta.description AS table_description,
    t.name AS task_name, t.description AS task_description,
       t.status AS task_status
        FROM
        public.user_app ua
        JOIN
        public.project p ON ua.id_user = p.user_id
        JOIN
        public.table_app ta ON p.id_project = ta.project_id
        JOIN
        public.task t ON ta.id_table = t.table_id
```

```
            WHERE
            ua.id_user BETWEEN %s AND 1;
```

Листинг Б.10 – Запрос для замера

```
SELECT
    ua.user_name, ua.email,
    p.name AS project_name, p.description AS project_description,
    ta.name AS table_name, ta.description AS table_description,
    t.name AS task_name, t.description AS task_description,
       t.status AS task_status
        FROM
        public.user_app ua
            JOIN
            public.project p ON ua.id_user = p.user_id
            JOIN
            public.table_app ta ON p.id_project = ta.project_id
            JOIN
            public.task t ON ta.id_table = t.table_id
                WHERE email = %s;
```

ПРИЛОЖЕНИЕ В

Таблица В.1 – Состояние таблицы лога после операций над таблицей проектов

ID	**table_name**	**operation**	**data**	**logged_at**
1	project	INSERT	{«new_data»: {«name»: «Test Project», «user_id»: 1, «id_project»: 2, «description»: «This is a test project»}, «operation»: «INSERT», «changed_at»: «...»}	2024-06-01 14:29:21
2	project	UPDATE	{«new_data»: {«name»: «Test Project», «user_id»: 1, «id_project»: 2, «description»: «Updated description»}, «old_data»: {«name»: «Test Project», «user_id»: 1, «id_project»: 2, «description»: «This is a test project»}, «operation»: «UPDATE», «changed_at»: «...»}	2024-06-01 14:29:21
3	project	DELETE	{«old_data»: {«name»: «Test Project», «user_id»: 1, «id_project»: 2, «description»: «Updated description»}, «operation»: «DELETE», «changed_at»: «...»}	2024-06-01 14:29:21

Таблица B.2 – Состояние таблицы лога после операций над таблицей карточек

ID	**table_name**	**operation**	**data**	**logged_at**
...	...	...	...	...
4	task	INSERT	{«new_data»: {«name»: «Test», «status»: «NO_ACTIVE», «id_task»: 3, «table_id»: 1, «sum_timer»: null, «description»: «Test task», «start_timer»: «...», «time_add_task»: «2024-06-01T14:29:30.652147», «time_end_task»: «...»}, «operation»: «INSERT», «changed_at»: «...»}	2024-06-01 14:29:30
5	task	UPDATE	{«new_data»: {«name»: «Test», «status»: «ACTIVE», «id_task»: 3, «table_id»: 1, ... }, «old_data»: {«name»: «Test», «status»: «NO_ACTIVE», «id_task»: 3, «table_id»: 1, «sum_timer»: null, «description»: «Test task», «start_timer»: «...», «time_add_task»: «...», «time_end_task»: «...»}, «operation»: «UPDATE», «changed_at»: «...»}	2024-06-01 14:29:30
6	task	DELETE	{«old_data»: {«name»: «Test», «status»: «ACTIVE», «id_task»: 3, «table_id»: 1, ... }	2024-06-01 14:29:30

Таблица В.3 – Зависимость времени выполнения запросов (мс) от количества записей и типа индекса (индекс для столбца id)

Кол-во записей	**Тип индекса**	**Время выполнения запроса (мс)**
10000	No Index	3.022
10000	BTREE	2.638
10000	HASH	3.101
20000	No Index	8.858
20000	BTREE	7.196
20000	HASH	7.787
30000	No Index	9.694
30000	BTREE	9.531
30000	HASH	10.040
40000	No Index	14.257
40000	BTREE	12.869
40000	HASH	13.303
50000	No Index	15.515
50000	BTREE	14.901
50000	HASH	15.246
60000	No Index	17.803
60000	BTREE	17.085
60000	HASH	17.955
70000	No Index	23.838
70000	BTREE	20.657
70000	HASH	21.796
80000	No Index	27.087
80000	BTREE	22.668
80000	HASH	24.029
90000	No Index	28.564
90000	BTREE	26.606
90000	HASH	27.186
100000	No Index	29.097
100000	BTREE	27.680
100000	HASH	27.895

Таблица В.4 – Зависимость времени выполнения запросов (мс) от количества записей и типа индекса (индекс для столбца email)

Кол-во записей	**Тип индекса**	**Время выполнения запроса (мс)**
10000	No Index	6.009
10000	BTREE	4.236
10000	HASH	3.748
20000	No Index	9.887
20000	BTREE	7.013
20000	HASH	7.366
30000	No Index	13.140
30000	BTREE	10.033
30000	HASH	10.162
40000	No Index	14.280
40000	BTREE	11.939
40000	HASH	11.238
50000	No Index	15.285
50000	BTREE	13.565
50000	HASH	13.192
60000	No Index	17.768
60000	BTREE	15.453
60000	HASH	15.277
70000	No Index	19.260
70000	BTREE	17.677
70000	HASH	17.901
80000	No Index	20.166
80000	BTREE	19.865
80000	HASH	19.118
90000	No Index	22.213
90000	BTREE	20.277
90000	HASH	20.077
100000	No Index	24.831
100000	BTREE	22.652
100000	HASH	22.078