Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка базы данных мероприятий по направлению «BigData»

Студент группы **ИУ7-61Б** _____ **Я. Р. Байрамгалин**
(Подпись, дата) (Н.О. Яковидис)

Руководитель _____ **Н. О. Яковидис**
(Подпись, дата) (И.О. Фамилия)

***2024** г.*

РЕФЕРАТ

Курсовая работа 35 с., 6 рис., 4 табл., 0 ист.

СУБД, БД, PostgreSQL, C++, Linux, userver.

Цель работы: спроектировать и разработать базу данных мероприятий по направлению «BigData».

Результат работы:

— формализована задача и определен необходимый функционал;
— описана структура объектов базы данных, а также сделан выбор СУБД для ее хранения и взаимодействия;
— спроектирован и разработан триггер, поддерживающий время обновления записи в актуальном состоянии;
— спроектирован и разработан интерфейс взаимодействия с базой данных;
— проведено исследование времени работы операции с использованием индексом, а также без таковых.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире огромное количество данных генерируется ежедневно, и их анализ становится все более важным для принятия обоснованных решений в различных областях науки и бизнеса. Термин «Big Data»[1] определяется как данные огромных объемов и разнообразия, а также методы их обработки, которые помогают в распределенном масштабе анализировать информацию[1]. Обработка данных значительного объема требует использования специальных инструментов и знаний. Именно поэтому передача такого опыта имеет ключевое значение в современном образовании.

Для передачи знаний, а также обсуждения и решения актуальных задач в области обработки больших объемов данных в России проводятся следующие мероприятия[2]: вебинары, воркшопы, выставки, конгрессы, конференции. Ораганизация таких мероприятий связана с рассмотрением организаторами заявок от потенциальных участников. Автоматизация данного процесса может сократить нагрузку на организаторов, а также сделать процесс более понятным и прозрачным для участников.

Целью курсовой работы является проектирование и разработка базы данных мероприятий по направлению «BigData».

Для достижения поставленной цели, необходимо решить следующие задачи:

- определить необходимый функционал приложения, предоставляющего доступ к базе данных;
- выделить роли пользователей, а также формализовать данные;
- проанализировать системы управления базами данных и выбрать подходящую систему для хранения данных;
- спроектировать базу данных, описать ее сущности и связи, спроектировать триггер;
- реализовать интерфейс для доступа к базе данных.

Итогом работы станет приложение, предоставляющее интерфейс доступа к базе данных.

1 Аналитический раздел

В данном разделе рассмотрены различные можели данных, их достоинства и недостатки, а также проведен анализ существующих систем управления базами данных. На основе результатов выбрана наиболее подходящая модель данных для реализации приложения. Кроме того, в данной секции будут определены акторы системы, будут формализованы данные, будут представлены отношения между сущностями, что поможет более луяше определить требования к приложению и спроектировать его структуру.

1.1 Требования к приложению

Подача заявки на конференцию, а также модерация может происходить с разных платформ, решено создать серверное приложение, предоставляющее интерфейс для взаимодействия с данными. Таким образом в дальнейшем для любой платформы можно будет создать клиент, который будет иметь возможность использовать всю функицональность системы.

Система должна поддерживать следующий функционал:

— возможность авторизации через sso[3] для каждого пользователя;
— создание, редактирование, удаление заявки на участие для потенциального докладчика на различные события;
— просмотр списка мероприятий для каждого пользователя;
— добавление приложений объемом до 10МБ каждое к заявке;
— просмотр статуса своих заявок;
— возможность взаимодействия докладчика и модератора через комментарии к заявке;
— утверждение или отколнения заявки модероатором.

1.2 Анализ существующих решений

Для анализа существубщих решений были рассмотрены следующие веб-приложения:

1) Онтико[4] — веб-приложение, используемое при проведении конферен-

ций highload++, а также некоторых других. Не имеет возможности авторизации через sso, имеет возможность записи на разные конференции.

2) YaConf[5] — веб-приложение, используемое при проведении конференций YaConf. Имеется возможность авторизации через аккаунт Яндекса, однако через подать заявку можно только на конференции, проводимые Яндексом.
3) Jugru[6] — веб-приложение, используемое для подачи заявок на конференцию cppcon и некоторые другие. Не имеет входа через sso, есть возможность записи на разные конференции.

Таким образом, ни одно из существующих решений не удовлетворяет всем требованиям, выдвинутым выше.

1.3 Пользователи системы

В приложении выделены следующие роли пользоваталей:

1) Потенциальный докладчик (далее просто докладчик) — пользователь, имеющий возможность авторизоваться, создать заявку на мероприятие, обновить текущую заявку (добавлением комментария к ней), отклонить собственную заявку. Также докаладчик может посмотреть список доступных для подачи заявок мероприятий и добавить приложений (файл) к своей заявке.
2) Модератор — пользователь, который обладаем правами докладчика, а также может просмтаривать весь список заявок на мероприятий, создавать, редактировать и удалять мероприятия и одобрять или отклонять заявки.
3) Администратор — пользователь, обладающий возможностями модератора, а также возможностью назначать и удалять модераторов.

На рисунке 1 представлена диаграмма использования приложения.

Рисунок 1 – Диаграмма использования приложения

1.4 Формализация данных

База данных должна хранить информацию о следующих сущностях:

— мероприятие;

— пользователь;

— группа доступов;

— заявка;

— комментарий к заявке;

— приложение (бинарный файл);
— мета-информация о приложении.

В таблице 1 представлены сущности и сведения о них.

Таблица 1 – Сущности и сведения о них

Сущность	**Сведения**
Мероприятие	Название, описание
Пользователь	Имя, телефон, email-адрес, метка времени первого входа
Группа доступов	Название, описание
Заяка	Событие, автор, описание, статус, метка времени создания, метка времени обновления
Комментарий к заявке	Текст, событие, автор, время создания
Приложение	Бинарные данные
Мета-информация о приложении	Имя файла, хеш, размер, метка времени загрузки

Данные сущности представлены на диаграмме на рисунке 2.

1.5 Анализ моделей баз данных

Перед выбором подходящей системы управления базами данных были рассмотрены 3 модели базы данных.

1.5.1 Объектная модель данных

Объектные хранилища (Object Stores) представляют собой специализированные хранилища данных, в которых информация хранится в виде объектов. Эти объекты могут содержать не только сами данные, но и дополнительные метаданные, методы и связи с другими объектами. Объектные хранилища широко применяются в различных областях информационных технологий, таких как разработка программного обеспечения, облачные вычисления, хранение дан-

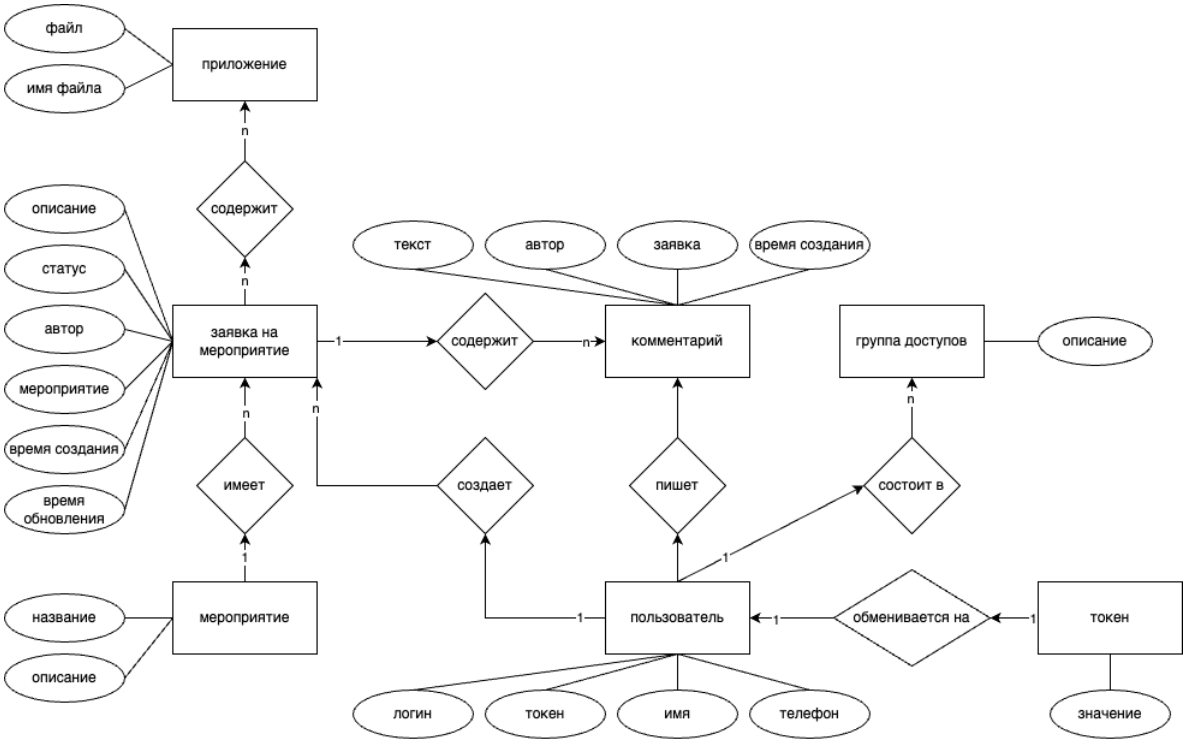

Рисунок 2 – ER-диаграмма в нотации Чена

ных больших объемов и др.

В данной работе объектная можель данных подходит для хранения приложений (бинарных файлов). Ключем (путем к файлу) может выступать уникальный идентификатор файла, создаваемый на уровне приложением, а объектом — сами набор байтов (содержимое файла).

1.5.2 Иерархическая модель

Иерархическая модель данных представляет базу данных как древовидную структуру, где объекты различных уровней связаны между собой. Каждый объект может содержать несколько объектов более низкого уровня, устанавливая отношение предка к потомку. Близнецами называются объекты, имеющие общего предка, а в программировании структура дерева также известна как ”братья”.

Иерархическую модель данных стоит использовать в случаях, когда данные могут быть организованы в виде древовидной структуры с четко выраженными отношениями предок-потомок. Это может быть полезно, когда данные имеют естественную иерархию, например, в организационных структурах,

классификациях товаров, семейных деревьях и т. д. Иерархическая модель также подходит для ситуаций, где данные имеют фиксированную структуру и не требуют сложных запросов или изменений в структуре данных. Однако следует учитывать, что иерархическая модель может ограничить гибкость и эффективность обработки данных в случае изменений в их структуре или потребности в сложных запросах.

В данных формализованных выше отношение предок-потомок имеют только сущности заяка-комментарий, поэтому иерархическая модель не подходит для применения в данной курсовой работе.

1.5.3 Реляционная модель

Реляционная модель данных[8] основана на понятии математических отношений. В реляционной модели данные и связи представлены в виде таблиц, каждая из которых имеет несколько столбцов с уникальными именами[2]. Основной идеей такой модели является использование ключей для связывания таблиц между собой. Ключ - это уникальный идентификатор, который позволяет однозначно определить каждую строку в таблице. Путем использования ключей можно устанавливать связи между таблицами для получения более сложной информации.

Модель баз данных в форме таблиц обладает рядом преимуществ. Во-первых, при правильном использолвании гарантируется высокая надежность и целостность данных. Во-вторых большое количество самых популярных современных СУБД (Postgres, Oracle, MySql и другие) основаны на реляционной моделе данных.

Однако у этой модели также есть недостатки. Например, она не всегда эффективна при работе с большими объемами информации, так как выполнение сложных запросов может занимать много времени. Кроме того, модель в форме таблиц не всегда подходит для хранения и обработки неструктурированных данных, таких как длинные тексты и бинарные файлы.

В рамках данной работы реляционная модель данных подходит лучше

остальным рассмотренных для всех данных кроме сущности «Приложение». Во-первых, все оставшиеся сущности легко представимы в табличной форме, во-вторых использование внешних ключей поможет гарантировать целостность данных.

1.5.4 Вывод

Для хранения сущности «Приложение» лучше подходит объектная модель содержание файла не находится в каких-либо отношениях с другими сущностями, а значит использование реляционной модели данных не является целесообразным. Также содержимое файла не образует какую-либо иерархию, что говорит о невозможности применения иерархической модели.

Для хранения остальных сущностей реляционная модель подходит лучшим образом, так как данные легко представимы в табличном виде, а также реляционная модель лучше остальных рассмотренных гарантирует целостность данных, что позволит избежать ошибок во время эксплуатации приложения.

1.6 Обзор объектных хранилищ

При выборе объектного хранилища данных стоит помнить, что использоваться она будет для хранения приложений (бинарных файлов) объемом до 10МБ. Ниже представлен обзор существующих решений и обоснование выбора базы данных.

1.6.1 Microsoft Azure Blob Storage

Microsoft Azure Blob Storage - это облачное хранилище данных, предоставляемое компанией Microsoft в рамках облачного сервиса Azure. Azure Blob Storage предназначено для хранения больших объемов неструктурированных данных, таких как файлы, изображения, видео, аудио, резервные копии, журналы и другие типы информации. Это расширяемое и высокодоступное хранилище, которое обеспечивает надежное хранение данных в облаке с возможностью масштабирования по требованию.

Основной недостаток – недоступна для использования в России.

1.6.2 Yandex S3

Amazon Simple Storage Service (Amazon S3) — это облачное хранилище данных, предоставляемое Amazon Web Services (AWS). S3 предоставляет возможность хранить и извлекать любое количество данных в Интернете, обеспечивая высокую доступность, надежность и масштабируемость.

Основные характеристики Amazon S3:

1) доступность и надежность: S3 обеспечивает высокую доступность данных и надежность, гарантируя, что ваши данные будут доступны в любое время;
2) масштабируемость: S3 позволяет хранить огромные объемы данных без необходимости заботиться о масштабировании инфраструктуры;
3) управление данными: Вы можете управлять доступом к вашим данным, устанавливать права доступа, шифровать данные и использовать другие функции для обеспечения безопасности информации;
4) стоимость: Вы платите только за использование, что делает S3 экономически выгодным решением для хранения данных.

Amazon S3 широко используется для хранения резервных копий, статических сайтов, медиафайлов, архивов данных, а также для обработки и анализа больших объемов информации.

Yandex S3 предоставляет совместимый с «оригинальным» Amazon S3 программный интерфейс и доступен в России. В современном мире s3 является де-факто стандартом хранения бинарных файлов.

1.7 Обзор реляционных баз данных

Ниже представлен обзор существующих решений обоснование выбора базы данных.

1.7.1 Oracle Database

Oracle Database - это одна из самых популярных и мощных реляционных баз данных, разработанная компанией Oracle Corporation. Она широко используется в корпоративных средах для хранения и управления данными, обеспе-

чивая высокую производительность, надежность и масштабируемость. Основным недостатком, который делает невохможным применения данной базы данных является отсутствие поддержки этой базы данных выбранным для разработки фреймворком userver.

1.7.2 MySQL

MySQL - это одна из самых популярных открытых реляционных баз данных, широко используемая веб-разработчиками и предприятиями для хранения и управления данными. Тем не менее, в рамках данном работы получилось выделить следующие минусы данной системы управления базами данных:

1) ограниченная поддержка даты и времени: MySQL имеет ограниченный набор типов данных для работы с датой и временем, что может вызывать сложности при работе с различными форматами даты и времени или при работе с часовыми поясами;
2) ограниченные возможности хранения текстовых данных: MySQL имеет ограничения на размер текстовых полей, что может быть проблемой при работе с большими объемами текстовых данных, таких как скрипты выступлений, отправленные через комментарии.

Хотя MySQL является мощной и гибкой базой данных, ограниченный набор типов данных может стать препятствием к применению в данном проекте.

1.7.3 PostgreSQL

PostgreSQL - это мощная и расширяемая объектно-реляционная система управления базами данных (СУБД), которая широко используется в различных проектах и приложениях. Она является открытой и бесплатной для использования, что делает ее популярным выбором среди разработчиков и организаций.

PostgreSQL предлагает широкий набор встроенных и сторонних расширений, которые позволяют расширять функциональность базы данных с помощью пользовательских функций, типов данных, операторов и других возможностей. PostgreSQL обеспечивает высокий уровень транзакционной безопасности благодаря поддержке ACID-свойств (атомарность, согласованность, изолирован-

ность, долговечность) и механизмам контроля целостности данных.

Из минусов PostgreSQL обычно выделяют отсутствие встроенной возможности шардирования, а также сложность настройки кластера базы данных. В рамках данной работы шардирование не планируется к использованию, а PostgreSQL будет развернута на единичной инсталляции, а не в кластере.

2 Конструкторский раздел

В данном разделе курсовой работы будет рассмотрен процесс проектирования базы данных для разрабатываемого приложения. Будут выделены ключевые этапы проектирования, подробно проанализированы действия в рамках ролевой модели, спроектирован триггер и описаны основные принципы проектирования приложения. Целью данного раздела будет создание базы данных, которая обеспечит стабильную работу приложения и удовлетворит потребности пользователей.

На основе выделенных ранее сущностей спроектированы следующие объекты базы данных.

1) Users — содержит информацию об участнике и включает следующие поля:
   - user_id — уникальный идентификатор пользователя;
   - token — авторизационный токен пользователя;
   - login — логин пользователя;
   - name — имя пользователя;
   - phone — телефон пользователя;
   - created_at — метка времени первой авторизации.
2) Events — содержит информацию о событиях и включает слудующие поля:
   - event_id — уникальный идентификатор события;
   - name — название события.
3) Requests — содержит информацию заявках на участие и включает слудующие поля:
   - request_id — уникальный идентификатор заявки;
   - event_id — идентификтор конференции, на которую подана заявка;
   - user_id — пользователь, подавший заявку;
   - description — описание заяки;
   - status — статус заявки;

— created_at — метка времени создания заявки;

— updated_at — метка времени обновления заявки.

4) Permissions — содержит информацию о правах, доступных пользователям, и включает слудующие поля:
   — slug — строковый идентификатор заявки;
   — user_id — идентификатор пользователя
5) FileMeta — содержит мета-информация о файлах и включает слудующие поля:
   — uuid — уникальный идентификатор файла;
   — source_name — изначальное имя файла;
   — hash — хеш файла;
   — created_at – время загрузки файла.
6) File— содержит информацию об улове участника и включает слудующие поля:
   — uuid — уникальный идентификатор файла;
   — binary_data — бинарные данные.

Диаграмма проектируемой базы данных представлена на рисунке 3.

2.1 Ролевая модель

Ролевая модель предполагает наличие трех ролей: участника, модератора и администратора. Стоит отметить, что модератор обладает всеми правами участника, а администратор — всеми правами модератора.

Для реализации поставленной выше задачи программа должна предоставлять следующие возможности всем пользователям:

— авторизация через sso;

— сеоздание, редактирование, удаление заявки на участие;

— просмотр списка мероприятий;

— добавление приложений к заявке;

— просмотр статуса своих заявок;

— общение через комментарии к заявке.

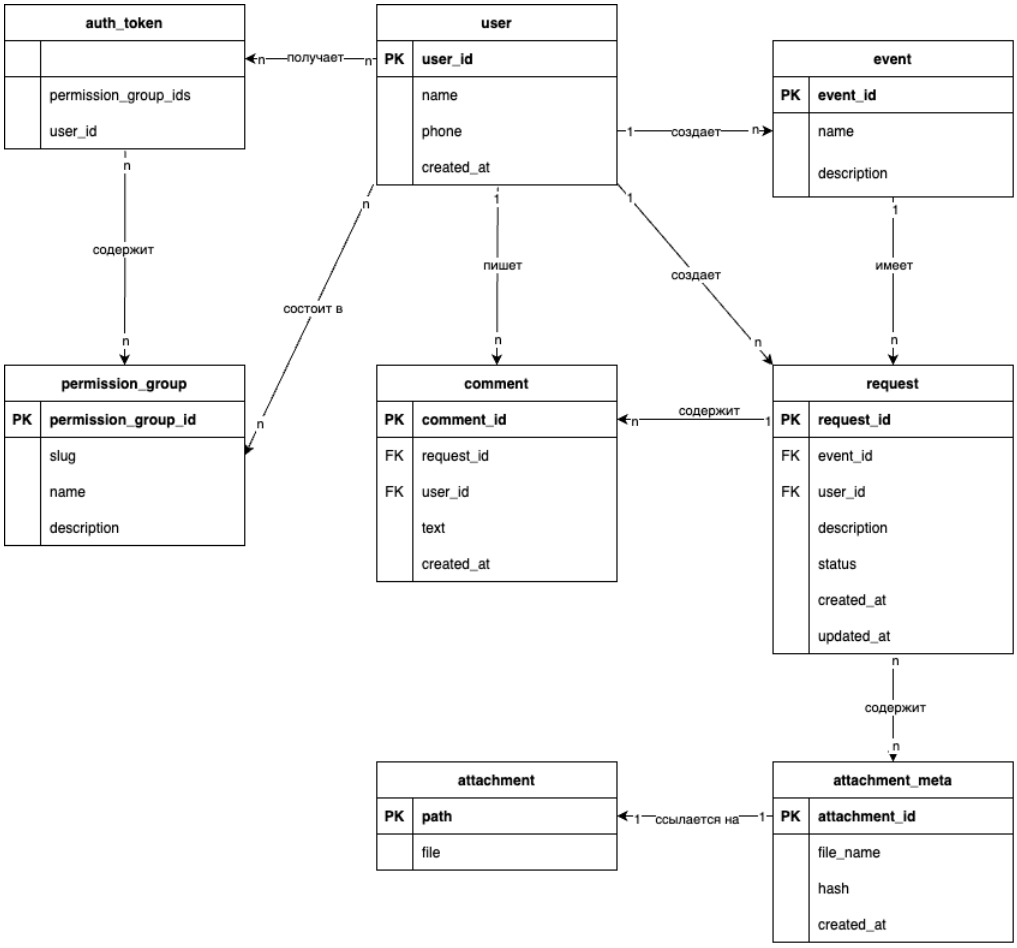

Рисунок 3 – Диаграмма проектируемой базы данных

Модератор также имеет право:

— просмтаривать все заявки;

— создавать, редактировать, удалять мероприятия;

— одобрять и отклонять заявки;

— назначать и удалять модераторов, администраторов;

Администратор обладаем всеми правами модератора, но кроме того может начначать и удалять модераторо, других администраторов.

2.2 Триггер

Триггер — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по изменению данных: добавлением, модификацией, удалением строки в заданной таблице.

В сущности «Request» имеется поле updated_at — время посоеднего изме-

нения заявки. Для гарантии корректности данного поля и исключения ошибок при update - запросах разумно создать триггер, который будет проставлять данному полю текущее время после совершения операции update над записью.

2.3 Проектиоврание приложения

При проектиорвании приолжения использован подход «чистой»[10] архитектуры, состоящей из следующих слоев:

1) слой бизнес логики — обработка основной логики работы приложения;
2) слой доступа к данным — подключение к базе данных, отправка запросов, получение информации из базы данных;
3) слой программного интерфеса — обработка запросов от пользователей, делегирование выполнения соответвующим сервисам бизнес-логики.

2.4 Вывод

В результате проектирования базы данных для разрабатываемого приложения спроектированы сущности базы данных и их связи, ролевая модель и триггер.

3 Технологический раздел

В данном разделе выбраны и обоснованы средства реализации, работана база данных, а также будет разработан интерфецс доступа к данным. Выполнено покрытие тестами публичного API приложение, протестирован триггер.

3.0.1 Резальтат выбора объектной базы данных

Для реализации проекта среди 2 рассмотренных вариантов была выбрана база данных Yandex S3, так как она доступна в России и удовлетворяет всем требованиям.

3.0.2 Результат выбора реляционной базы данных

В качестве основной базы данных принято использовать PostgreSQL, так как она удовлетворяет заданным требованиям и, кроме того, выбранный для разработки фреймворк userver имеет лучшую поддержку именно этой базы данных.

3.1 Сущности объектной базы данных

Единственная сущность объектной базы данных — бинарный файл. Доступ к нему осуществляется по имени файла. Имя файла — его идентификатор (полу в таблице file_meta).

Проектирование ролевой модели для объектной базы данных разрабатываемого приложения нецелесообрвзно.

3.2 Сущности реляционной базы данных

Ниже представлены разработанные сущности базы данных.

3.2.1 Таблицы

На рисунке 4 представлены сущности-таблицы полученной базы данных. Для обеспечения целостности данных используются внешние ключи, ограничения на уникальность первичного ключа.

3.2.2 Индекс

В таблице requests используется стандартный b-tree индекс на поле user_id. Использование необходимо обеспечения быстродействия запроса из листинга

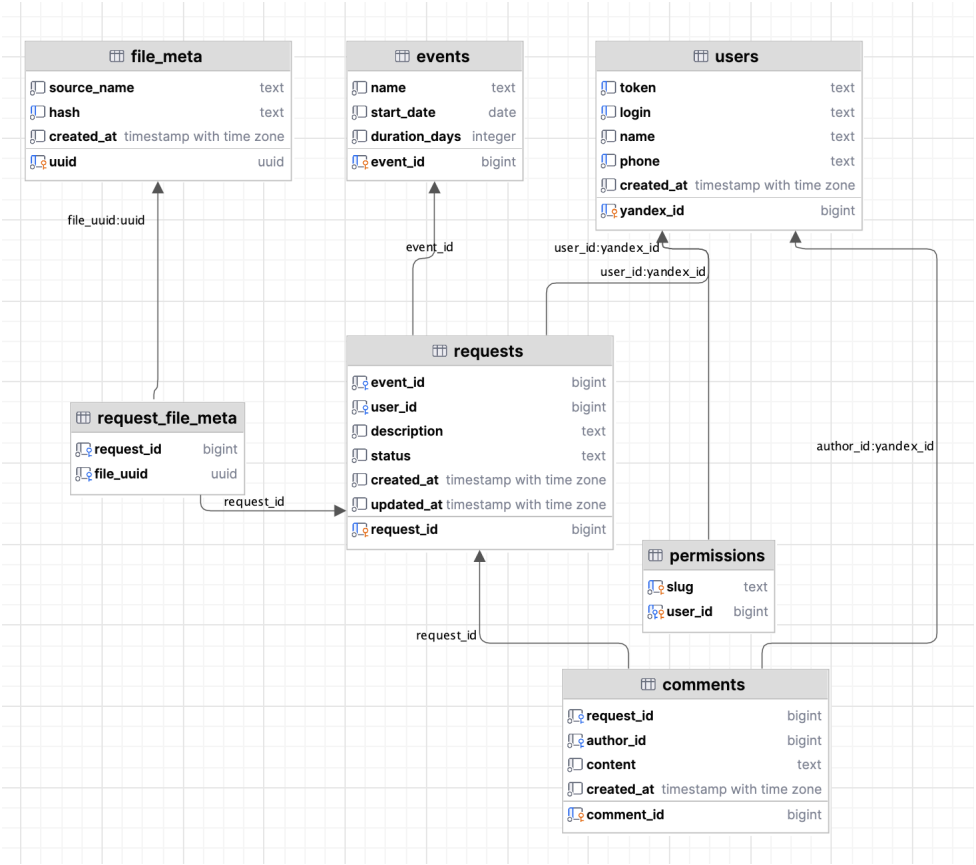

Рисунок 4 – ER-диаграмма базы данных

1 и для проведения исследования.

3.2.3 Триггер

На рисунке 5 представлена диаграмма спроектированного триггера.

Триггер гарантирует корректность метки времени updated_at таблицы requests. Для тестирования триггера написаны функциональные тесты на языке Python с использованием библиотеки pytest. В таблице 2 приведены сценарии тестирования.

Таблица 2 – Сценарии тестирования триггера

Событие	**Ожидаемый результат**	**Рельный результат**
Изменилось поле description	updated_at = now()	updated_at = now()
Изменилось поле status	updated_at = now()	updated_at = now()
Попытка изменить updated_at	updated_at = now()	updated_at =now()

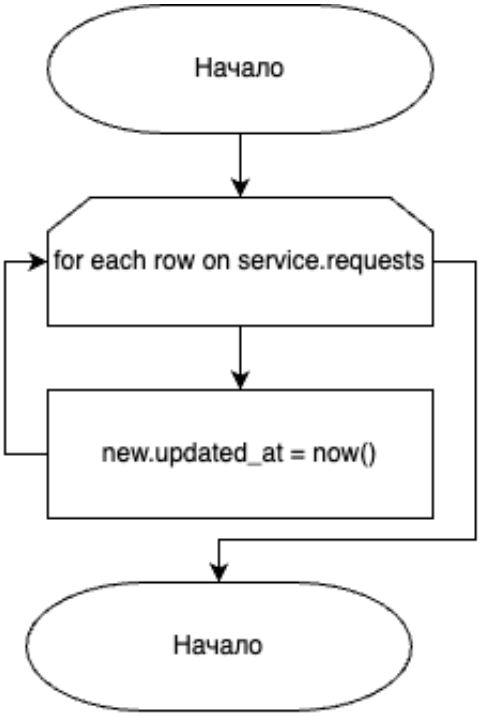

Рисунок 5 – Блок-схема триггера

3.2.4 Роли

Помимо ролевой модели на уровне приложения разработана для дополнительных гарантий безопасности дополнительно разраотана ролевая модель на уровне приложения. Права пользователей представлены в таблице 3.

Таблица 3 – Ролевая модель на уровне базы данных

Действие	**Таблица**	**everybody**	**modarator**	**admin**
select	все таблицы	да	да	да
insert, update	requests	да	да	да
insert, update, delete	events	нет	да	да
insert, update	users	да	да	да
insert, update,	requests_file_meta	да	да	да
insert, update	file_meta	да	да	да
insert, update, delete	permisisions	нет	нет	да
insert, update	comments	да	да	да

Такая ролевая модель на уровне базы данных обеспечит пользователей

минимальными необходимыми правами для доступа к просмотру и модификации данных.

3.3 Обеспечение целостности данных

Для обечспечения целостности предприняты следующие меры:

1) добавлены внешние ключи (полный список можно увидеть на er-диаграмме на рисунке 4);
2) доавлено ограничение на уникальность хеша файла, это позволит гарантировать отсутсвие в системе двух одинаковых файлов;
3) создан и протестирован триггер на поле updated_at таблицы requests.

3.4 Интерфейс доступа к данным

Доступ к данным осуществляется методом отправки http запроса на сервер, на котором развернуто приложение. Ниже приведено опсание эндпоинтов.

1) /v1/request GET — получение информации о запросе по его идентификатору.
2) /v1/requests GET — получение информации о всех запросах, доступных для просмотра.
3) /v1/requests POST — создание/обновление запроса.
4) /v1/request/status PATCH — обновление статуса заявки.
5) /v1/request/comment POST — добавление комментария к заявке.
6) /v1/file GET — получение файла по его идентификтору.
7) /v1/file POST — добавление файла (приложения).
8) /v1/events GET — просмотр списка мероприятий.
9) /v1/event POST — создание мероприятия.
10) /v1/event DELETE — удаление мероприятия.

3.5 Вывод

Разработаны и протестированы сущности базы данных, разработан интерфейс взаимодействия с данными.

4 Исследовательский раздел

4.1 Цель исследования

Целью исследования является оценка зависимости времени выполнения запроса от наличия индексов в базе данных. Для оценки использован запрос на языке sql[9] из листинга 1. Необходимо определеить влияет ли использование стандартного b-tree индекса на user_id на время выполнения данного запроса при различном количестве записей.

Листинг 1 – Запрос для исследования

```
1 select event_id, description, status, created_at, updated_at
2 from service.requests
3 where user_id={user_id};
```

4.2 Описание исследования

На первом этапе исследования производится замер времени выполнения запросов при количестве записей от 100 до 1000 с шагом 100 в таблице requests без использовния индекса на user_id.

На втором этапе исследования производится замер времени выполнения запросов при том же количестве записей, что и на предыдущем этапе, однако с использованием индексов.

Для уменьшения погрешности вычислений каждый запрос отправляется 1000 раз, а за результат берется медианное время 1000 запросов.

Для уменьшения временных издержек, связанных с работой сети база данных и скрипт отправки запросов звпускаются на одной физической машине.

Результаты измерений приведены в таблице 4. График изображен на рисунке 6.

Таблица 4 – Результат исследования

Количество записей	**Без индексов, мс**	**С индексами, мс**
100	9.03	6.91
200	12.50	8.73
300	13.41	10.3
400	14.12	11.78
500	15.46	12.13
600	16.12	12.72
700	16.42	12.34
800	16.91	13.4
900	17.07	14.23
1000	17.39	14.10

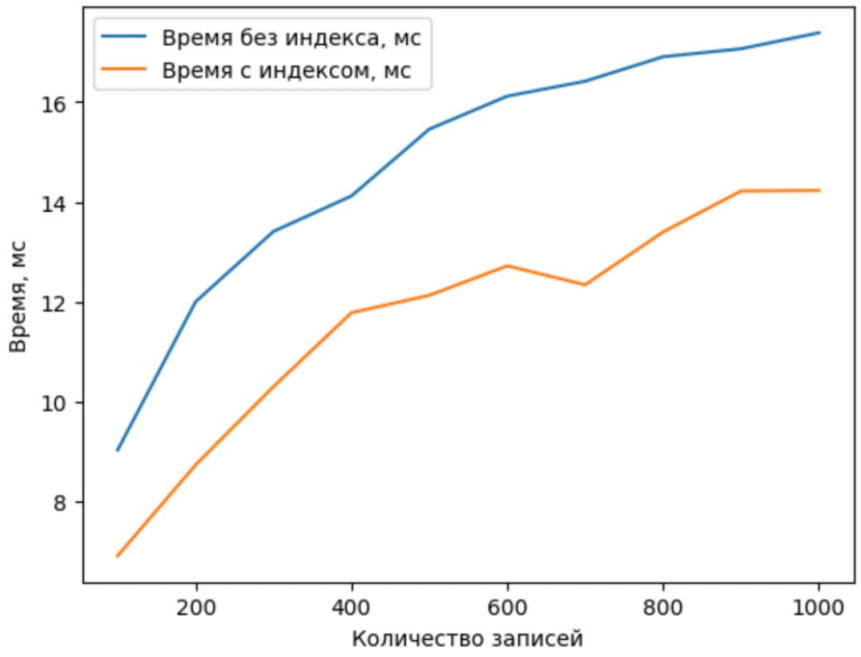

Рисунок 6 – График измерений

4.3 Вывод

В результате исследования установлено, что для запроса из листинга 1 время выполнения уменьшается при использовании индексов.

ЗАКЛЮЧЕНИЕ

В ходе разработки данного курсового проекта решены следующие задачи:

— определен необходимый функционал приложения, предоставляющего доступ к базе данных;
— выделены роли пользователей, а данные формализованы;
— выбраны 2 системы управления бащами данных;
— спроектирована база данных, описаны сущности и связи, разработан тригер;
— реализован интерфейс доступа к данным.

Достигнута цель — спроектирована и разработана база данных мероприятий по направлению «Big Data».

Проведено исследование — установлено уменьшение времени выполнения простого (с точки зрения количества условий) запроса, при наличии индекса в базе данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фангаццини Д., Шаклеина М.В., Юрас И.А. BIG DATA В ОПРЕДЕЛЕНИИ СОЦИАЛЬНОГО САМОЧУВСТВИЯ НАСЕЛЕНИЯ РОССИИ // Прикладная эконометрика. 2018. №2 (50). URL: https://cyberleninka.ru/article/n/big-data-v-opredelenii-sotsialnogo-samochuvstviya-naseleniya-rossii (дата обращения: 21.05.2024).

2. Национальный портал в сфере искусственного интеллекта [Электронный ресурс]. — Режим доступа: `https://ai.gov.ru/`, свободный (дата обращения: 18.04.2024)

3. De Clercq, J. (2002). Single Sign-On Architectures. In: Davida, G., Frankel, Y., Rees, O. (eds) Infrastructure Security. InfraSec 2002. Lecture Notes in Computer Science, vol 2437. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45831-X_4

4. Онтико [Электронный ресурс]. — Режим доступа: `https://conf.ontico.ru/`, свободный (дата обращения: 16.04.2024)

5. YaConf | Yandex [Электронный ресурс]. — Режим доступа: `https://yace.yandex.ru/`, свободный (дата обращения: 16.04.2024)

6. Jugru [Электронный ресурс]. — Режим доступа: `https://jugru.org/callforpapers-speaker/#conferences`, свободный (дата обращения: 30.04.2024)

7. M. Mesnier, G. R. Ganger and E. Riedel, ”Object-based storage,”in IEEE Communications Magazine, vol. 41, no. 8, pp. 84-90, Aug. 2003, doi: 10.1109/MCOM.2003.1222722. keywords: Secure storage;Data security;Disk drives;Stability;Multimedia databases;Operating systems;Costs;File servers;File systems;Storage area networks,

8. Жалолов Озод Исомиддинович, Хаятов Хуршид Усманович Понятие SQL и реляционной базы данных // Universum: технические науки. 2020. №6-1 (75). URL: https://cyberleninka.ru/article/n/ponyatie-sql-i-relyatsionnoy-bazy-dannyh (дата обращения: 26.05.2024).

9. SQL — Structured Query Language [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?view=sql-server-ver15, свободный (дата обращения: 16.04.2023)

10. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2018. – 352 с.

ПРИЛОЖЕНИЕ А

Листинг 2 – Схема базы данных

```
1
2 drop schema if exists service cascade;
3
4 create schema service;
5
6 create table service.users(
7 yandex_id bigint primary key,
8 token text not null unique,
9 login text unique not null,
10 name text not null,
11 phone text unique not null,
12 created_at timestamptz not null default now()
13 );
14
15 create table service.events(
16 event_id bigserial not null primary key,
17 name text not null
18 );
19
20 create table service.requests(
21 request_id bigserial not null primary key,
22 event_id bigint not null references service.events(event_id),
23 user_id bigint not null references service.users(yandex_id),
24 description text not null,
25 status text not null default \'new\',
26 created_at timestamptz not null default now(),
27 updated_at timestamptz not null default now()
28 );
29
30 create table service.permissions(
31 slug text not null,
32 user_id bigint not null references service.users(yandex_id),
33
34 primary key (user_id, slug)
35 );
36
```

```
create table service.file_meta(
uuid uuid not null primary key,
source_name text not null,
hash text not null unique,
created_at timestamptz not null default now()
);

create table service.request_file_meta(
request_id bigint not null references service.requests(request_id),
file_uuid uuid not null references service.file_meta(uuid),

unique (request_id, file_uuid)
);

create table service.comments(
comment_id bigserial not null primary key,
request_id bigint not null references service.requests(request_id),
author_id bigint not null references service.users(yandex_id),
content text not null,
created_at timestamptz not null default now()
);

create or replace function service.update_updated_at()
returns trigger as $$
begin
new.updated_at = now();
return new;
end;
$$ language plpgsql;

drop trigger if exists update_updated_at on service.requests;
create trigger update_updated_at
before update on service.requests
for each row
execute function service.update_updated_at();
```