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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Реализация базы данных для хранения информации об экологических инцидентах»

Студент ИУ7-66Б _____ Е. А. Мазур
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. С. Кострицкий
(Подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 46 с., 5 рис., 7 табл., 23 источн., 5 прил.

Ключевые слова: Базы Данных, PostgreSQL, NoSQL, Golang, REST

Объектом разработки является база данных для приложения, предоставляющего доступ к информации об экологических инцидентах.

Цель работы — спроектировать и разработать базу данных для хранения данных об экологических инцидентах.

Для достижения данной цели необходимо решить следующие задачи:

- формализовать задачу и сформулизовать требования к разрабатываемому ПО;
- проанализировать существующие СУБД и выбрать подходящую для решения задачи систему;
- спроектировать базу данных, описать ее сущности и связи;
- реализовать интерфейс доступа к базе данных;
- реализовать ПО для работы пользователей с базой данных.

В результате выполнения работы была спроектирована и разработана база данных для хранения данных об экологических инцидентах.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Экологический инцидент — любое происшествие, которое привело или может привести к неблагоприятным последствиям для окружающей среды.

Персистентность — возможность долговременного хранения состояния

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных

Система управления базами данных — совокупность программных средств, обеспечивающих управление созданием и использованием баз данных

Structured Query Language — язык структурированных запросов

Not only Structed Query Language — термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL

PostgreSQL — свободная объектно-реляционная система управления базами данных

MongoDB — объектно-реляционная система управления базами данных компании Oracle

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

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

ПО — Программное обеспечение

SQL — Structured Query Language

NoSQL — Not only Structed Query Language

СУБД — Система управления базами данных

ACID — atomicity, consistency, isolation, durability

ВВЕДЕНИЕ

Разлив дизельного топлива в Норильске[1], сибирские пожары[2], катастрофа на Камчатке, повлекшая за собой массовую гибель морских животных[3] – последние годы подобные инциденты потрясают экологов и всех небезразличных жителей планеты. Чаще всего первоисточником информации о происшествиях являются местные жители, которые, обнаружив проблему, начинают звонить журналистам, писать экологическим организациям, публиковать информацию в социальных сетях[4].

Необходим единый сервис для размещения информации об экологических инцидентах. Такая система не только позволила бы оперативно оповещать специальные службы и экологичсекое сообщество о происшествиях, но и повысила бы экологическую осведомленность населения. Сформированная таким образом база данных экологических инцидентов могла бы использоваться исследователями для анализа.

Цель данной работы – разработать базу данных для хранения данных об экологических инцидентах.

Для достижения данной цели, необходимо решить следующие задачи:

- формализовать задачу и сформулировать требования к разрабатываемому ПО;
- проанализировать существующие СУБД и выбрать подходящую для решения задачи систему;
- спроектировать базу данных, описать ее сущности и связи;
- реализовать интерфейс доступа к базе данных;
- реализовать ПО для работы пользователей с базой данных.

1 Аналитическая часть

В данном разделе будут рассмотрены существующие решения, формализована решаемая задача, выбран способ хнанения данных.

1.1 Обзор существующих решений

В 2020 году WWF России представил проект, который призван помочь жителям нашей страны оперативно сообщать об авариях и инцидентах [5]. Идея была размещена в рамках форума "Сильные идеи для нового времени"[6]. С помощью специальной формы на сайте каждый сможет оперативно передать фото с места аварии, координаты инцидента и снабдить свое сообщение комментарием. По информации с места событий будут запрашиваться оперативные данные космического мониторинга. Однако на момент написания курсовой работы нет сведений о судьбе данного проекта.

WWF России и Fairy, бренд компании Procter&Gamble, запустили национальную программу общественного мониторинга аварийных экологических ситуаций. Программа позволяет своевременно отслеживать и оповещать дежурные службы об инцидентах, аналогичных катастрофам на побережье Авачинской бухты на Камчатке, разливе в Норильске или аварии на продуктопроводе на реке Оби [7]. В рамках данного проекта у пользователей есть возможность получить информацию об инцидентах в с помощью интерактивной карты. Однако пользователи не могут загружать информацию об инцидентах в систему. Также данная программа узконаправлена – данные собираются только об инцидентах в нефтегазовом секторе.

1.2 Формализация задачи

Под экологическим инцидентом будем понимать любое происшествие, которое привело или может привести к неблагоприятным последствиям для окружающей среды.

Разрабатываемая система должна выделять следующие типы экологических инцидентов:

- разлив нефти или нефтепродуктов;
- выброс радиоактивных веществ;

- выброс аварийно химически опасных веществ;
- выброс биологически опасных веществ;
- пожар;
- несанкционированная свалка, скопление мусора;
- другие экологические инциденты.

Каждая запись об экологическом инциденте должна содержать следующие данные:

- название (краткое описание);
- тип;
- координаты;
- дата;
- статус (подтвержден/не подтвержден);
- пользователь, опубликовавший инцидент.

Каждая запись об экологическом инциденте может содержать следующие данные:

- комментарий.

Работа пользователей с базой экологических инцидентов должна осуществляться посредством клиент-серверного веб-приложения с возможностью авторизации. Приложение должно поддерживать работу четырех типов пользователей со следующими возможностями:

- неавторизованный пользователь:
  - просмотр записей об экологических инцидентах в виде списка;
  - просмотр записей об экологических инцидентах в виде карты;
  - регистрация нового аккаунта или вход в существующий;
- авторизованный пользователь:

  - все возможности неавторизованного пользователя;
  - добавление записи об экологическом инциденте;
  - выход из аккаунта;

- модератор:
  - все возможности авторизованного пользователя;
  - подтвеждение инцидента (установка соответствующего статуса);
  - удаление записи об инциденте;
  - редактирование записи об инциденте;

- администратор:
  - все возможности модератора;
  - назначение авторизованному пользователю роли модератора;
  - снятие авторизованного пользователя с роли модератора;
  - редактирование существующих в системе типов инцидентов;
  - редактирование существующих в системе статусов инцидентов;
  - редактирование существующих в системе ролей пользователей.

Ниже представлена диаграмма использования приложения.

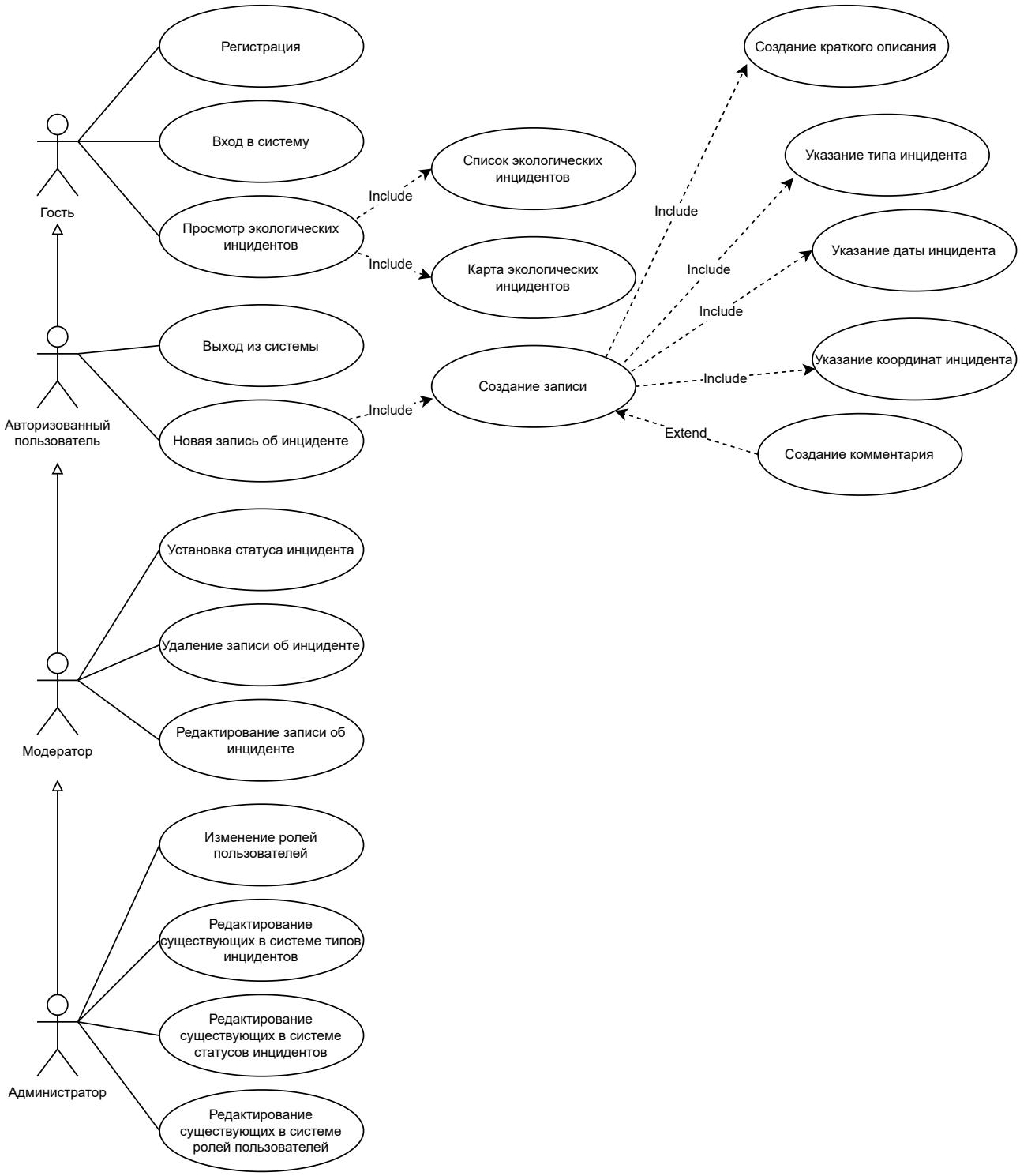


Рисунок 1.1 – Диаграмма использования приложения

1.3 Модели баз данных

Модель базы данных – это тип модели данных, которая определяет логическую структуру базы данных и в корне определяет, каким образом данные могут храниться, организовываться и обрабатываться [8]. Рассмотрим основные модели.

Иерархическая модель

В иерархической модели информация организована в виде древовидной структуры. Каждая запись имеет одного "родителя"и несколько потомков.

Такие модели данных графически могут быть представлены в виде перевернутого дерева. Оно состоит состоящее из объектов, каждый из которых имеет уровень. Корень дерево – первый уровень, его потомки второй и так далее.

Основной недостаток иерархической модели данных – невозможно реализовать отношение "many-to-many связь, при которой у потомка существует несколько родителей.

В качестве примера такой модели можно привести каталог в операционной системе Windows.

Сетевая модель

Сетевая модель – это структура, у которой любой элемент может быть связан с любым другим элементом.

Сетевая модель описывается как иерархическая, но в отличие от последней лишена недостатков, связанных с невозможностью реализовать "many-to-many"связь. Разница между сетевой и иерархической заключается в том, что в сетевой модели у потомка может быть несколько предков, когда у иерархической только один.

Основной недостаток – жёсткость задаваемых структур и сложность изменения схем БД из-за реализации связей между объектами на базе физических ссылок (через указатели на объекты).

В качестве примера такой модели можно привести WWW (World Wide Web).

Реляционная модель

Данные в реляционной модели хранятся в виде таблиц и строк, таблицы могут иметь связи с другими таблицами через внешние ключи, таким образом образуя некие отношения.

Реляционные базы данных используют язык SQL. Структура таких баз данных позволяет связывать информацию из разных таблиц с помощью внешних ключей (или индексов), которые используются для уникальной идентификации любого атомарного фрагмента данных в этой таблице. Другие таблицы могут ссылаться на этот внешний ключ, чтобы создать связь между частями данных и частью, на которую указывает внешний ключ.

SQL используют универсальный язык структурированных запросов для определения и обработки данных. Это накладывает определенные ограничения: прежде чем начать обработку, данные надо разместить внутри таблиц и описать.

Нереляционная модель

Данные нереляционных баз данных не имеют общего формата. Они могут представляться в виде документов (MongoDB [9], Tarantool [10]), пар ключ-значение (Redis [11]), графовых представляниях.

Динамические схемы для неструктурированных данных позволяют:

- ориентировать информацию на столбцы или документы;
- основывать ее на графике;
- организовывать в виде хранилища Key-Value;
- создавать документы без предварительного определения их структуры, использовать разный синтаксис;
- добавлять поля непосредственно в процессе обработки.

1.3.1 Выбор модели базы данных

Для решения задачи будет использоваться реляционная модель данных по нескольким причинам:

- изложение данных будет осуществляться в виде таблиц;
- данные структурированные, структура нечасто измяема;
- возможность исключить дублирование, используя связь между отношениями с помощью внешних ключей;
- разделение доступа к данным от способа их физической организации.

1.4 Формализация данных

Ниже представлена ER-диаграмма сущностей в нотации Чена.

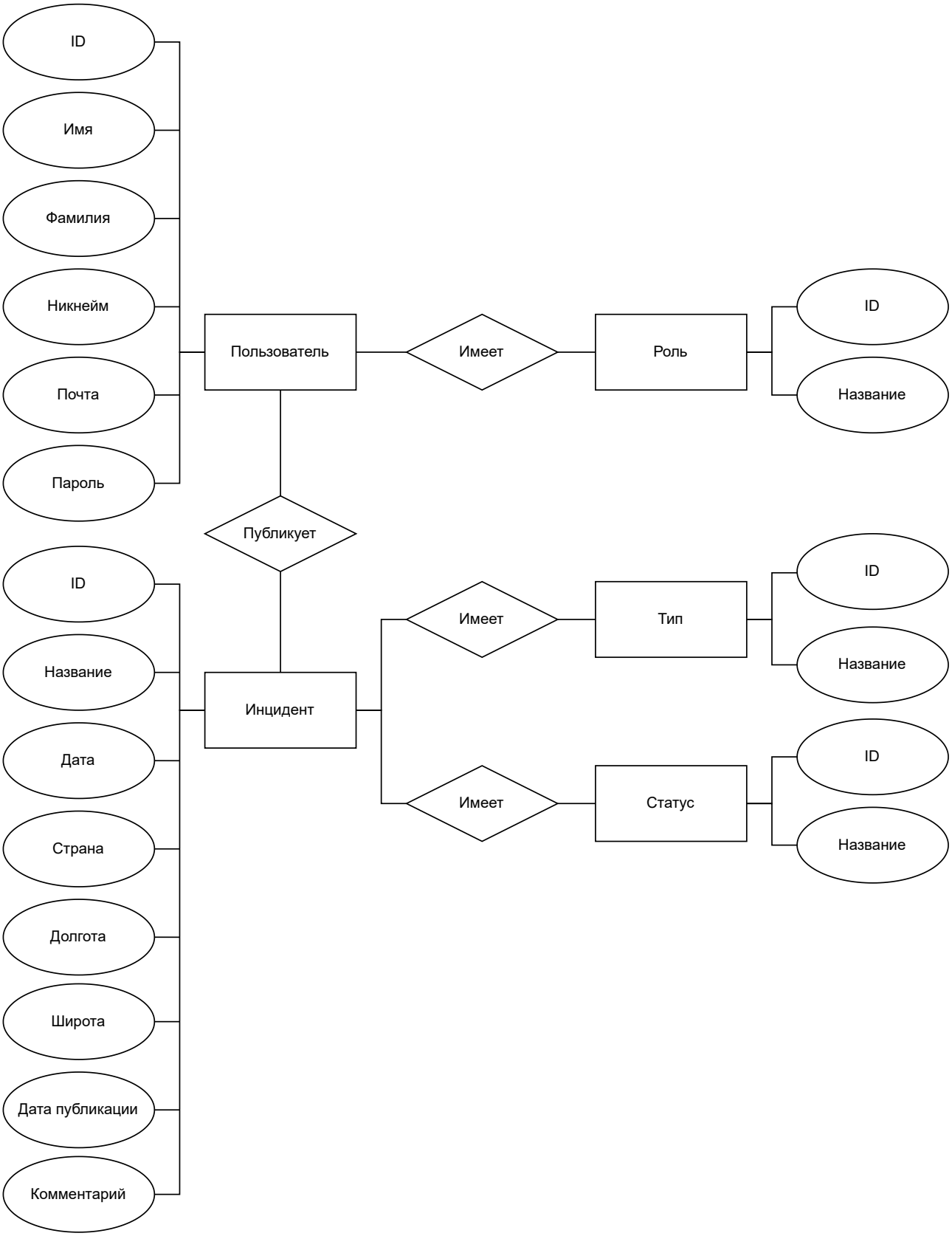


Рисунок 1.2 – ER-диаграмма сущностей в нотации Чена

Хранение данных пользователей и информации об инцидентах должно происходить в одной базе данных. Пользователи и инциденты должны иметь уникальные идентификаторы, чтобы их можно было однозначно идентифицировать.

Пользователи и инциденты связаны отношением один ко многим, информация об этих связях должна сохраняться.

Роли пользователей, типы инцидентов, статусы инцидентов должны храниться в виде пар: уникальный идентификатор – имя.

Вывод

В данном разделе:

- рассмотрены существующие решения;
- формализована решаемая задача;
- формализированы данные, используемые при решении задачи;
- выбрана модель баз данных для решения задачи.

2 Конструкторская часть

В данном разделе будет спроектирована базы данных.

2.1 Проектирование базы данных

Для основной логики приложения была спроектирована база данных, представленная в виде ER-модели в нотации Мартина.

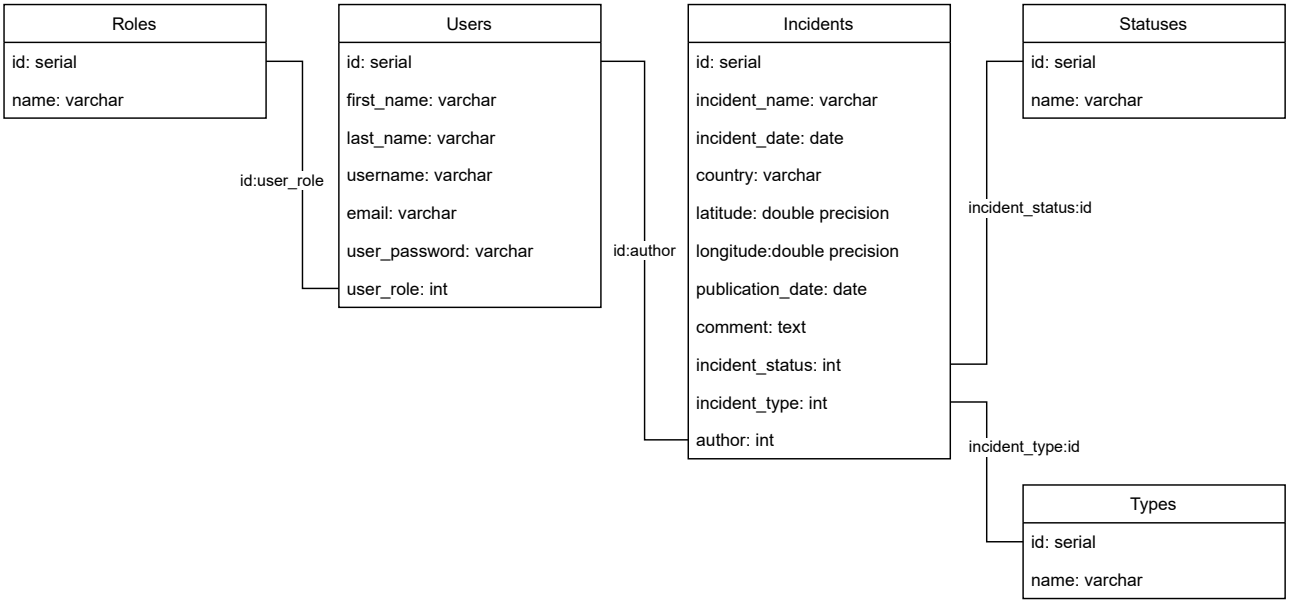


Рисунок 2.1 – ER-модель в нотации Мартина

База данных будет состоять из следующих сущностей:

1. Users – пользователи;
2. Roles – роли пользователя;
3. Incidents – экологические инциденты;
4. Statuses – статусы экологических инцидентов;
5. Types – типы экологических инцидентов.

Сущность Users

Сущность Users содержит информацию о пользователях:

Таблица 2.1 – Описание полей таблицы `Users`

Поле	**Значение**
`id`	Уникальный идентификатор
`first_name`	Имя
`last_name`	Фамилия
`username`	Псевдоним
`email`	Адресс электронной почты
`user_password`	Хешированный пароль
`user_role`	Роль

Сущность Incidents

Сущность Incidents содержит информацию об экологических инцидентах:

Таблица 2.2 – Описание полей таблицы `Incidents`

Поле	**Значение**
`id`	Уникальный идентификатор
`incident_name`	Название (краткое описание)
`incident_date`	Дата
`country`	Страна
`latitude`	Широта
`longitude`	Долгота
`publication_date`	Дата публикации
`comment`	Комментарий
`incident_status`	Статус
`incident_type`	Тип
`author`	Пользователь, опубликовавший запись об инциденте

Сущность Roles

Сущность Roles содержит информацию о существующих в системе ролях пользователй:

Таблица 2.3 – Описание полей таблицы `Roles`

Поле	**Значение**
`id`	Уникальный идентификатор.
`role_name`	Название.

Сущность Statuses

Сущность Statuses содержит информацию о существующих в системе статусах инцидентов:

Таблица 2.4 – Описание полей таблицы `Statuses`

Поле	**Значение**
`id`	Уникальный идентификатор
`role_name`	Название

Сущность Types

Сущность Types содержит информацию о существующих в системе типах инцидентов:

Таблица 2.5 – Описание полей таблицы `Types`

Поле	**Значение**
`id`	Уникальный идентификатор
`role_name`	Название

Внешние ключи

- В Таблице Users поле user_role ссылается на поле id таблицы Roles;
- В Таблице Incidents
  - поле incident_type ссылается на поле id таблицы Types;
  - поле incident_status ссылается на поле id таблицы Statuses;
  - поле author ссылается на поле id таблицы Users.

Ролевая модель

В базе данных существуют четыре роли:

1. Администратор – имеет доступ к всем таблицам, доступны все операции над ними;
2. Модератор – имеет доступ к таблицам Incidents, Users; над таблицами доступны операции: SELECT, INSERT, UPDATE, DELETE;
3. Авторизированный пользователь – доступ к таблице Incidents; над таблицами доступны операции: SELECT, INSERT;
4. Гость – доступны операция SELECT над таблицей Incidents и операция INSERT над таблицей Users.

Триггер

Для функционирования системы необходимо постоянное наличие в ней хотя бы одного модератора. Необходим механизм, который будет препятствовать нарушению этого условия.

Для решения этой задачи можно использовать триггер, который будет срабатывать при удалении пользователя или обновлении его роли. Если операция производится с единственным модератором в системе, ее выполнение будет остановлено.

Ниже представлена схема алгоритма функции, выполняемой триггером.

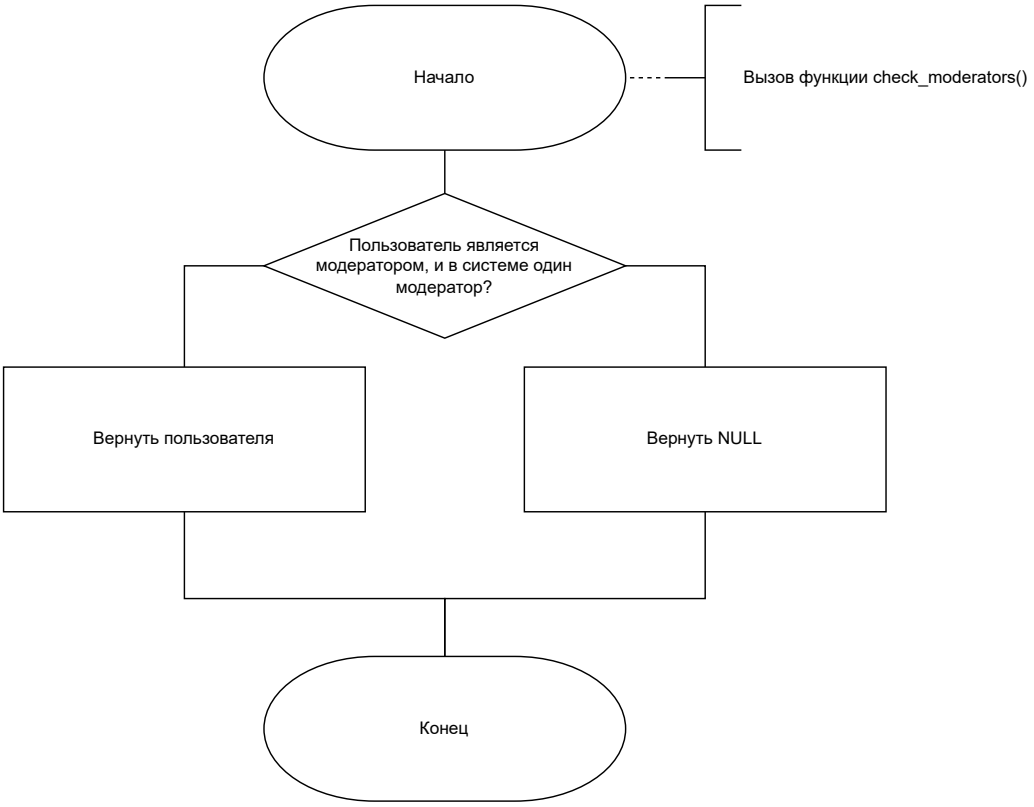


Рисунок 2.2 – Схема алгоритма функции триггера check_moderators()

Вывод

В данном разделе:

- спроектированы сущности базы данных;
- описаны связи сущностей с помощью внешних ключей;
- описана ролевая модель для разграничения доступа к базе данных;
- описан триггер для корректного функционирования системы.

3 Технологическая часть

В данном разделе будет обоснован выбор СУБД, описаны средства реализации и интерфейс ПО.

3.1 Выбор СУБД

Для хранения данных приложения выбрана свободная объектно-реляционная система управления базами данных PostgreSQL. Данная СУБД поддерживает все необходимые для работы приложения типы данных. Она отвечает требованиям ACID, а также поддерживает целостность данных за счет наличия таких средств, как первичные и внешние ключи, ограничения NOT NULL, прочие проверочные ограничения.

3.2 Выбор средств реализации

Приложение для работы с базой данных, представляет из себя Web-сервер, доступ к которому осуществляется с помощью REST API [12].

Для реализации сервера используется язык программирования Go [13]. Это компилируемый многопоточный, статически типизированный язык программирования, подходящий для создания простых, но эффективных Web-сервисов.

Для взаимодействия с базой данных используется пакет github.com/jmoiron/sqlx[14], расширяющий стандартный пакет языка database/sql[15], а также драйвер github.com/lib/pq[16] для работы с СУБД PostgreSQL.

Для реализации REST API используется Web фреймворк Gin [17].

Для реализации пользовательского Web-интерфейса использованы язык гипертекстовой разметки документов HTML[18] и язык программирования JavaScript[19]. Для отображения экологических инцидентов на интерактивной карте использован API Яндекс.Карт[20].

3.3 Детали реализации

Создание и наполнение базы данных

SQL-скрипты создания базы данных, создания таблиц, установки ограничений, создания ролевой модели, создания триггера представлены в листингах

А.1 — А.2. Наполнение базы данных происходило с помощью заготовленных .csv файлов.

Паттерн взаимодействия с данными

Для взаимодействия с данными используется паттерн "Репозиторий". Он позволяет отделить логику приложения от деталей реализации слоя доступа к данным. Структура, реализующая репозиторий для сущности Incidents базы данных, приведена в листингах Б.1 — Б.2.

REST API

`REST API` интерфейс приложения представлен в Таблице 3.1.

Таблица 3.1 – Описание реализованного `REST API`

Путь	Метод	Описание
/api/auth/sign-up	POST	Регистрация пользователя
/api/auth/sign-in	POST	Вход в систему
/api/users/	GET	Получение списка всех пользователей
/api/users/id	GET	Получение пользователя с заданным id
/api/users/id	PUT	Обновление данных пользователя с заданным id
/api/users/id	DELETE	Удаление пользователя с заданным id
/api/users/id/role	PUT	Изменение роли пользователя с заданным id
/api/incidents/	GET	Получение списка всех инцидентов
/api/incidents/	POST	Создание инцидента
/api/incidents/id	GET	Получение инцидента с заданным id
/api/incidents/id]	PUT	Обновление данных об инциденте с заданным id
/api/incidents/id	DELETE	Удаление инцидента с заданным id
/api/incidents/type/type	GET	Получение списка инцидентов с заданным типом
/api/roles/	GET	Получение списка всех ролей
/api/roles/	POST	Создание роли
/api/roles/id	GET	Получение роли с заданным id
Продолжение на следующей странице		

Таблица 3.1 – продолжение

Путь	Метод	Описание
/api/roles/id]	PUT	Обновление данных о роли с заданным id
/api/roles/id	DELETE	Удаление роли с заданным id
/api/statuses/	GET	Получение списка всех статусов инцидентов
/api/statuses/	POST	Создание статуса
/api/statuses/id	GET	Получение статуса с заданным id
/api/statuses/id]	PUT	Обновление данных о статусе с заданным id
/api/statuses/id	DELETE	Удаление статуса с заданным id
/api/types/	GET	Получение списка всех типов инцидентов
/api/types/	POST	Создание типа
/api/types/id	GET	Получение типа с заданным id
/api/types/id]	PUT	Обновление данных о типе с заданным id
/api/types/id	DELETE	Удаление типа с заданным id
		Конец таблицы

Web-интерфейс

Web-интерфейс предназначен для просмотра информации об экологических инцидентах в виде списка, а также в виде отметок на интерактивной карте. Демонстрация Web-интерфейса представлена на рисунке Д.1.

Вывод

В данном разделе:

- определена СУБД для решения задачи;
- определены средства реализации ПО: используемые языки программирования и библиотеки;
- приведены детали реализации ПО: SQL-скрипты, слой взаимодействия с данными, REST API интерфейс, Web-интерфейс.

4 Исследовательская часть

В данном разделе будет проведен эксперимент по сравнению производительности реляционной и документоориентированной СУБД.

4.1 Цель исследования

Цель исследования – сравнить время, которое требуется для выполнения операций вставки, удаления, обновления и получения данных с помощью реляционной СУБД PostgreSQL и документоориентированной СУБД MongoDB.

Для достижения цели требуется:

- создать базу данных MongoDB для хранения сущностей экологических инцидентов;
- создать слой доступа к данным (паттерн "Репозиторий") MongoDB;
- сравнить скорость выполнения операций вставки, удаления, обновления и получения данных при использовании репозиториев PostgreSQL и MongoDB.

4.2 Описание исследования

Для выполнения исследования была создана база данных MongoDB с коллекцией экологических инцидентов. Далее в приложении был реализован слой доступа к данным из базы данных MongoDB (паттерн "Репозиторий"), который представлен в листингах В.1–В.4.

В ходе эксперимента выполнялись замеры времени (мс) для следующих операции с каждой из баз данных с помощью соответствующих репозиториев:

- 1000 операций вставки;
- 1000 операций обновления;
- 1000 операций поиска по id (один результат);
- 1000 операций поиска по типу (список результатов);
- 1000 операций удаления.

Код проведения исследования представлен в листингах Г.1–Г.4.

4.3 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведенно тестирование ПО:

- Операционная система: Ubuntu 20.10 [21] Linux [22] 64-bit;
- Оперативная память: 12 GB;
- Процессор: AMD® Ryzen 7 3700u with radeon vega mobile gfx × 8 [23].

4.4 Результаты исследования

Конечный результат сформирован усредненными значениями времени (мс), полученными по результатам 100 экспериментов и представлен в виде таблицы:

Таблица 4.1 – Результаты исследования (время выполнения 1000 операций в мс)

СУБД	Вставка	Обновление	Удаление	Поиск одного	Поиск списка
PostgreSQL	731	705	649	417	1932
MongoDB	516	477	430	472	4158

Вывод

В ходе исследования в приложении была реализована поддержка документоориентированной СУБД MondoDB, создана база данных, написан слой доступа к ее данным. Были проведены замеры времени, которое требуется для выполнения операций вставки, удаления, обновления и получения данных с помощью PostgreSQL и MongoDB.

Исследование показало, что выполнение операци вставки, обновления и удаления выполняются примерно в 1.5 раза быстрее при использовании MongoDB. Поиск по ID выполняется примерно за равное время независимо от использованной СУБД (MongoDB быстрее в 1.1 раза). Поиск списка инцидентов (по типу) выполняется в 2.1 раза быстрее при использовании PostgreSQL.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана база данных для хранения данных об экологических инцидентах.

Для достижения данной цели были решены следующие задачи:

- формализована задача и сформулированы требования к разрабатываемому ПО;
- проанализированы существующие СУБД и выбрана подходящую для решения задачи систему;
- спроектирована база данных, описаны ее сущности и связи;
- реализован интерфейс доступа к базе данных;
- реализовано ПО для работы пользователей с базой данных.

В ходе выполнения исследовательской части работы было установлено, что выполнение операци вставки, обновления и удаления выполняются примерно в 1.5 раза быстрее при использовании документоориентированной СУБД MongoDB. Поиск по ID выполняется примерно за равное время независимо от использованной СУБД (MongoDB быстрее в 1.1 раза). Поиск списка инцидентов (по типу) выполняется в 2.1 раза быстрее при использовании реляционной СУБД PostgreSQL.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Разлив дизтоплива в Норильске стал самым масштабным на планете - РИА Новости [Электронный ресурс]. — Режим доступа: https://ria.ru/20201224/razliv-1590698352.html (дата обращения: 13.05.2022).
2. В Сибири горят более 25 тысяч гектаров леса - РИА Новости [Электронный ресурс]. — Режим доступа: https://ria.ru/20200429/1570719017.html (дата обращения: 13.05.2022).
3. Экологическая катастрофа на Камчатке - РИА Новости [Электронный ресурс]. — Режим доступа: https://ria.ru/20201005/kamchatka-1578207502.html (дата обращения: 13.05.2022).
4. Эксперт Greenpeace об экологических катастрофах на Камчатке и в Норильске и их последствиях - Forbes [Электронный ресурс]. — Режим доступа: https://www.forbes.ru/forbeslife/410857-my-ostaemsya-v-plenu-sovetskih-tehnologiy-i-okazhemsya-v-glubokom-ekonomicheskom (дата обращения: 13.05.2022).
5. Предотвратить экологические катастрофы можно только при участии общественности - WWF [Электронный ресурс]. — Режим доступа https://wwf.ru/resources/news/ekologicheskaya-politika/wwf-predotvratit-ekologicheskie-katastrofy-mozhno-tolko-pri-uchastii-obshchestvennosti/ (дата обращения: 13.05.2022).
6. Сильные идеи для нового времени [Электронный ресурс]. — Режим доступа: https://idea.asi.ru. (дата обращения: 13.05.2022).
7. Предотвратить экологические катастрофы можно только при участии общественности - WWF [Электронный ресурс]. — Режим доступа: https://wwf.ru/resources/news/Regulirovanie/wwf-rossii-i-brend-fairy-zapustili-obshchestvennyy-monitoring-avariynykh-ekologicheskikh-situatsiy-v/ (дата обращения: 13.05.2022).
8. Модель баз данных. — URL: https://clck.ru/323j87 (дата обр. 16.09.2022).
9. MongoDB. — URL: https://www.mongodb.com/ (дата обр. 13.05.2022).

10. Tarantool – Платформа In-memory вычислений [Электронный ресурс]. — Режим доступа: https://www.tarantool.io/ru/ (дата обращения: 07.06.2021).

11. Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker [Электронный ресурс]. — Режим доступа: https://redis.io/ (дата обращения: 07.06.2021).

12. What is a REST API? - Red Hat [Электронный ресурс]. — Режим доступа: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата обращения: 07.06.2021).

13. The Go Programming Language. — URL: https://go.dev/ (дата обр. 27.08.2022).

14. Go sqlx documentation [Электронный ресурс]. — Режим доступа: https://pkg.go.dev/github.com/jmoiron/sqlx (дата обращения: 16.05.2022).

15. Go sql documentation [Электронный ресурс]. — Режим доступа: https://pkg.go.dev/database/sql (дата обращения: 13.05.2022).

16. Go pq documentation [Электронный ресурс]. — Режим доступа: https://pkg.go.dev/github.com/lib/pq (дата обращения: 13.05.2022).

17. Go Gin documentation [Электронный ресурс]. — Режим доступа: https://pkg.go.dev/github.com/gin-gonic/gin (дата обращения: 07.06.2022).

18. HTML documentation [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTML (дата обращения: 07.06.2022).

19. JavaScript documentation [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 07.06.2022).

20. API Яндекс Карт [Электронный ресурс]. — Режим доступа: https://yandex.ru/dev/maps/?p=realty (дата обращения: 08.06.2022).

21. Ubuntu: Enterprise Open Source and Linux [Электронный ресурс]. — Дата обращения: 14.09.2021. Режим доступа: https://ubuntu.com/.

22. Linux – Getting Started [Электронный ресурс]. — Дата обращения: 14.09.2021. Режим доступа: https://linux.org.

23. Мобильный процессор AMD Ryzen™ 7 3700U с графикой Radeon™ RX Vega 10 [Электронный ресурс]. — Дата обращения: 14.09.2020. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u.

ПРИЛОЖЕНИЕ А
SQL-скрипты

В Листингах А.1-?? представлены SQL-скрипты создания базы данных, создания таблиц, установки ограничений, создания ролевой модели, создания триггера, наполнения базы данных из .csv файлов..

Листинг А.1 – Скрипт создания базы данных.

```
DROP DATABASE IF EXISTS greenalarm;
DROP ROLE IF EXISTS greenalarm;

CREATE DATABASE greenalarm;
CREATE USER greenalarm WITH
    CREATEROLE
    PASSWORD 'greenalarm';
GRANT ALL PRIVILEGES ON DATABASE greenalarm TO greenalarm;
```

Листинг А.2 – Скрипт создания таблиц базы данных. Часть 1

```
CREATE TABLE Users (
  id SERIAL PRIMARY KEY,
  first_name varchar NOT NULL,
  last_name varchar NOT NULL,
  username varchar NOT NULL UNIQUE,
  email varchar NOT NULL UNIQUE,
  user_password varchar NOT NULL,
  user_role int NOT NULL
);

CREATE TABLE Roles (
  id SERIAL PRIMARY KEY,
  role_name varchar NOT NULL
);

CREATE TABLE Incidents (
  id SERIAL PRIMARY KEY,
  incident_name varchar NOT NULL,
  incident_date date,
  country varchar NOT NULL,
  latitude double precision,
  longitude double precision,
  publication_date date NOT NULL,
  comment text,
  incident_status int NOT NULL,
  incident_type int NOT NULL,
  author int NOT NULL
);
```

Листинг А.3 – Скрипт создания таблиц базы данных. Часть 2

```
CREATE TABLE Statuses (
  id SERIAL PRIMARY KEY,
  status_name varchar NOT NULL
);

CREATE TABLE Types (
  id SERIAL PRIMARY KEY,
  type_name varchar NOT NULL
);

ALTER TABLE Users ADD FOREIGN KEY (user_role) REFERENCES Roles (
   id);
ALTER TABLE Incidents ADD FOREIGN KEY (incident_type) REFERENCES
   Types (id);
ALTER TABLE Incidents ADD FOREIGN KEY (incident_status)
   REFERENCES Statuses (id);
ALTER TABLE Incidents ADD FOREIGN KEY (author) REFERENCES Users (
   id);
```

Листинг А.4 – Скрипт создания триггера.

```
CREATE FUNCTION count_moderators() RETURNS integer AS $$
    SELECT
        COUNT (*)
    FROM
        users
    WHERE user_role = 2;
$$ LANGUAGE SQL;

CREATE FUNCTION check_moderators() RETURNS TRIGGER AS
$$
BEGIN
    IF (OLD.user_role = 2 and count_moderators() = 1)
    THEN
        RETURN NULL;
    ELSE
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE  plpgsql;

CREATE TRIGGER check_moderators BEFORE DELETE OR UPDATE of
   user_role ON users
    FOR EACH ROW EXECUTE FUNCTION check_moderators();
```

Листинг А.5 – Скрипт создания ролевой модели базы данных.

```
DROP ROLE IF EXISTS admin, moderator, authorised_user, guest;

-- Администратор
CREATE USER admin
    WITH PASSWORD 'password';

GRANT SELECT, INSERT, UPDATE, DELETE
    ON ALL TABLES IN SCHEMA public
    TO admin;

-- Модератор
CREATE USER moderator
    WITH PASSWORD 'password';

GRANT SELECT, INSERT, UPDATE, DELETE
    ON TABLE incidents, users
    TO moderator;

-- Авторизированный пользователь
CREATE USER authorised_user
    WITH PASSWORD 'password';

GRANT SELECT, INSERT
    ON TABLE incidents
    TO authorised_user;

-- Гость
CREATE USER guest
    WITH PASSWORD 'password';

GRANT SELECT
    ON TABLE incidents
    TO guest;

GRANT INSERT
    ON TABLE users
    TO guest;
```

Листинг А.6 – Скрипт установки ограничений базы данных.

```
ALTER TABLE Incidents
    ADD CONSTRAINT latitude_check CHECK (latitude >= -90 AND
       latitude <= 90),
    ADD CONSTRAINT longitude_check CHECK (longitude >= -180 AND
       longitude < 180)
```

ПРИЛОЖЕНИЕ Б
Паттерн "Репозиторий"PostgreSQL

В Листингах Б.1-Б.4 представлена реализация паттерна "Репозиторий"для доступа к данным PostgreSQL на примере сущности Incidents.

Листинг Б.1 – Слой доступа к данным PostgreSQL. Часть 1

```
package pgrepo

import (
    "context"
    "fmt"

    "github.com/ekaterinamzr/green-alarm/internal/entity"
    "github.com/ekaterinamzr/green-alarm/pkg/postgres"
)

type IncidentRepository struct {
    *postgres.Postgres
}

func NewIncidentRepository(pg *postgres.Postgres) *
   IncidentRepository {
    return &IncidentRepository{pg}
}

func (r *IncidentRepository) Create(ctx context.Context, i entity
   .Incident) (string, error) {
    var id string
    query := `  INSERT INTO
                    incidents (
                        incident_name,
                        incident_date,
                        country,
                        latitude,
                        longitude,
                        publication_date,
                        comment,
                        incident_status,
                        incident_type,
                        author)
                VALUES
                    ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)
                RETURNING
                    id`

    row := r.DB.QueryRowContext(ctx, query, i.Name, i.Date, i.
       Country, i.Latitude, i.Longitude, i.Publication_date, i.
       Comment, i.Status, i.Type, i.Author)
```

Листинг Б.2 – Слой доступа к данным PostgreSQL. Часть 2

```
    err := row.Scan(&id)
    if err != nil {
        return "", fmt.Errorf("pgrepo - incident - CreateIncident
           : %w", err)
    }
    return id, nil
}

func (r *IncidentRepository) GetAll(ctx context.Context) ([]
   entity.Incident, error) {
    var all []entity.Incident

    query := `  SELECT
                    id,
                    incident_name,
                    incident_date,
                    country,
                    latitude,
                    longitude,
                    publication_date,
                    COALESCE(comment, '') AS comment,
                    incident_status,
                    incident_type,
                    author
                FROM
                    incidents`

    if err := r.DB.SelectContext(ctx, &all, query); err != nil {
        return nil, fmt.Errorf("pgrepo - incident - GetAll: %w",
           err)
    }

    return all, nil
}

func (r *IncidentRepository) GetById(ctx context.Context, id
   string) (*entity.Incident, error) {
    var incident entity.Incident

    query := `  SELECT
                    id,
                    incident_name,
                    incident_date,
                    country,
                    latitude,
                    longitude,
                    publication_date,
                    COALESCE(comment, '') AS comment,
                    incident_status,
                    incident_type,
```

Листинг Б.3 – Слой доступа к данным PostgreSQL. Часть 3

```
                    author
                FROM
                    incidents
                WHERE
                    id = $1`

    row := r.DB.QueryRowxContext(ctx, query, id)

    if err := row.StructScan(&incident); err != nil {
        return nil, fmt.Errorf("pgrepo - incident - GetById: %w",
            err)
    }
    return &incident, nil
}

func (r *IncidentRepository) GetByType(ctx context.Context,
   requiredType int) ([]entity.Incident, error) {
    var incidents []entity.Incident

    query := `  SELECT
                    id,
                    incident_name,
                    incident_date,
                    country,
                    latitude,
                    longitude,
                    publication_date,
                    COALESCE(comment, '') AS comment,
                    incident_status,
                    incident_type,
                    author
                FROM
                    incidents
                WHERE
                    incident_type = $1`

    if err := r.DB.SelectContext(ctx, &incidents, query,
       requiredType); err != nil {
        return nil, fmt.Errorf("pgrepo - incident - GetByType: %w
           ", err)
    }

    return incidents, nil
}

func (r *IncidentRepository) Update(ctx context.Context, id
   string, updated entity.Incident) error {
    query := `  UPDATE
                    incidents
                SET
                    incident_name = $1,
```

Листинг Б.4 – Слой доступа к данным PostgreSQL. Часть 4

```
                    incident_date = $2,
                    country = $3,
                    latitude = $4,
                    longitude = $5,
                    comment = $6,
                    incident_status = $7,
                    incident_type = $8,
                    author = $9
                WHERE
                    id = $10`

    _, err := r.DB.ExecContext(ctx, query,
        updated.Name,
        updated.Date,
        updated.Country,
        updated.Latitude,
        updated.Longitude,
        updated.Comment,
        updated.Status,
        updated.Type,
        updated.Author,
        id,
    )

    if err != nil {
        return fmt.Errorf("pgrepo - incident - Update: %w", err)
    }

    return nil
}

func (r *IncidentRepository) Delete(ctx context.Context, id
   string) error {
    query := `  DELETE
                FROM
                    incidents
                WHERE
                    id = $1`

    _, err := r.DB.ExecContext(ctx, query, id)

    if err != nil {
        return fmt.Errorf("pgrepo - incident - Delete: %w", err)
    }

    return nil
}
```

ПРИЛОЖЕНИЕ В
Паттерн "Репозиторий"MongoDB

В Листингах В.1-В.4 представлена реализация паттерна "Репозиторий"для доступа к данным MongoDB для проведения исследования.

Листинг В.1 – Слой доступа к данным MongoDB. Часть 1

```
package mongorepo

import (
    "context"
    "fmt"

    "github.com/ekaterinamzr/green-alarm/internal/entity"
    "github.com/ekaterinamzr/green-alarm/pkg/mongo"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/bson/primitive"
)

type IncidentRepository struct {
    *mongo.Mongo
}

func NewIncidentRepository(mongo *mongo.Mongo) *
   IncidentRepository {
    return &IncidentRepository{mongo}
}

func (r *IncidentRepository) Create(ctx context.Context, i entity
   .Incident) (string, error) {
    collection := r.DB.Database("greenalarm").Collection("
       incidents")
    incident := bson.D{
        {Key: "incident_name", Value: i.Name},
        {Key: "incident_date", Value: i.Date},
        {Key: "country", Value: i.Country},
        {Key: "latitude", Value: i.Latitude},
        {Key: "longitude", Value: i.Longitude},
        {Key: "publication_date", Value: i.Publication_date},
        {Key: "comment", Value: i.Comment},
        {Key: "incident_status", Value: i.Status},
        {Key: "incident_type", Value: i.Type},
        {Key: "author", Value: i.Author}}
    res, err := collection.InsertOne(ctx, incident)
    if err != nil {
        return "", fmt.Errorf("mongorepo - incident - Create: %w"
           , err)
    }
```

Листинг B.2 – Слой доступа к данным MongoDB. Часть 2

```
    id := res.InsertedID.(primitive.ObjectID).Hex()
    return id, nil
}

func (r *IncidentRepository) GetAll(ctx context.Context) ([]
   entity.Incident, error) {
    var all []entity.Incident

    collection := r.DB.Database("greenalarm").Collection("
       incidents")

    filter := bson.D{}

    cursor, err := collection.Find(ctx, filter)
    if err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetAll: %w
           ", err)
    }

    if err = cursor.All(ctx, &all); err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetAll: %w
           ", err)
    }

    return all, nil
}

func (r *IncidentRepository) GetById(ctx context.Context, id
   string) (*entity.Incident, error) {
    var incident entity.Incident

    collection := r.DB.Database("greenalarm").Collection("
       incidents")

    objId, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetById: %
           w", err)
    }

    filter := bson.D{{Key: "_id", Value: objId}}

    err = collection.FindOne(ctx, filter).Decode(&incident)
    if err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetById: %
           w", err)
    }

    return &incident, nil
}
```

Листинг B.3 – Слой доступа к данным MongoDB. Часть 3

```
func (r *IncidentRepository) GetByType(ctx context.Context,
   requiredType int) ([]entity.Incident, error) {
    var incidents []entity.Incident

    collection := r.DB.Database("greenalarm").Collection("
       incidents")

    filter := bson.D{{Key: "incident_type", Value: requiredType}}

    cursor, err := collection.Find(ctx, filter)
    if err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetByType:
            %w", err)
    }

    if err = cursor.All(ctx, &incidents); err != nil {
        return nil, fmt.Errorf("mongorepo - incident - GetByType:
            %w", err)
    }

    return incidents, nil
}

func (r *IncidentRepository) Update(ctx context.Context, id
   string, updated entity.Incident) error {
    collection := r.DB.Database("greenalarm").Collection("
       incidents")

    update := bson.D{{Key: "$set", Value: bson.D{
        {Key: "incident_name", Value: updated.Name},
        {Key: "incident_date", Value: updated.Date},
        {Key: "country", Value: updated.Country},
        {Key: "latitude", Value: updated.Latitude},
        {Key: "longitude", Value: updated.Longitude},
        {Key: "comment", Value: updated.Comment},
        {Key: "incident_status", Value: updated.Status},
        {Key: "incident_type", Value: updated.Type},
        {Key: "author", Value: updated.Author},
    }}}

    objId, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return fmt.Errorf("mongorepo - incident - Update: %w",
           err)
    }
    _, err = collection.UpdateByID(ctx, objId, update)
    if err != nil {
        return fmt.Errorf("mongorepo - incident - Update: %w",
           err)
    }
    return nil
```

Листинг B.4 – Слой доступа к данным MongoDB. Часть 4

```
func (r *IncidentRepository) Delete(ctx context.Context, id
   string) error {
    collection := r.DB.Database("greenalarm").Collection("
       incidents")

    objId, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return fmt.Errorf("mongorepo - incident - Delete: %w",
           err)
    }

    filter := bson.D{{Key: "_id", Value: objId}}

    _, err = collection.DeleteOne(ctx, filter)
    if err != nil {
        return fmt.Errorf("mongorepo - incident - Delete: %w",
           err)
    }

    return nil
}
```

ПРИЛОЖЕНИЕ Г
Проведение исследования

В Листингах Г.1-Г.4 представлен код проведения исследования.

Листинг Г.1 – Проведение исследования. Часть 1

```
package main

import (
    "context"
    "fmt"
    "log"
    "math/rand"
    "time"

    "github.com/ekaterinamzr/green-alarm/config"
    "github.com/ekaterinamzr/green-alarm/internal/entity"
    "github.com/ekaterinamzr/green-alarm/internal/infrastructure/
       mongorepo"
    "github.com/ekaterinamzr/green-alarm/internal/infrastructure/
       pgrepo"
    "github.com/ekaterinamzr/green-alarm/pkg/mongo"
    "github.com/ekaterinamzr/green-alarm/pkg/postgres"
)

const (
    n    = 1000
    reps = 10
)

type repo interface {
    Create(context.Context, entity.Incident) (string, error)
    GetAll(context.Context) ([]entity.Incident, error)
    GetById(context.Context, string) (*entity.Incident, error)
    Update(context.Context, string, entity.Incident) error
    Delete(context.Context, string) error
    GetByType(context.Context, int) ([]entity.Incident, error)
}

const charset = "
   abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
   "

func randomString() string {
    c := charset[rand.Intn(len(charset))]
    return string(c)
}
```

Листинг Г.2 – Проведение исследования. Часть 2

```
func randomInt(min, max int) int {
    return rand.Intn(max-min+1) + min
}

func randomFloat(min, max float64) float64 {
    return rand.Float64()*(max-min) + min
}

func randomIncident() entity.Incident {
    return entity.Incident{
        Name:             randomString(),
        Date:             time.Now(),
        Country:          randomString(),
        Latitude:         randomFloat(-90.0, 90.0),
        Longitude:        randomFloat(-180.0, 180.0),
        Publication_date: time.Now(),
        Comment:          randomString(),
        Status:           1,
        Type:             randomInt(1, 7),
        Author:           1,
    }
}

func randomIncidents(n int) []entity.Incident {
    rand.Seed(time.Now().UnixNano())
    incidents := make([]entity.Incident, n)
    for i := range incidents {
        incidents[i] = randomIncident()
    }
    return incidents
}

func randomTypes(n int) []int {
    rand.Seed(time.Now().UnixNano())
    min := 1
    max := 7

    types := make([]int, n)
    for i := range types {
        types[i] = randomInt(min, max)
    }
    return types
}

func create(r repo, data []entity.Incident) int64 {
    startedAt := time.Now().UnixNano()
    for i := range data {
        id, _ := r.Create(context.Background(), data[i])
        data[i].Id = id
    }
    finishedAt := time.Now().UnixNano()
```

Листинг Г.3 – Проведение исследования. Часть 3

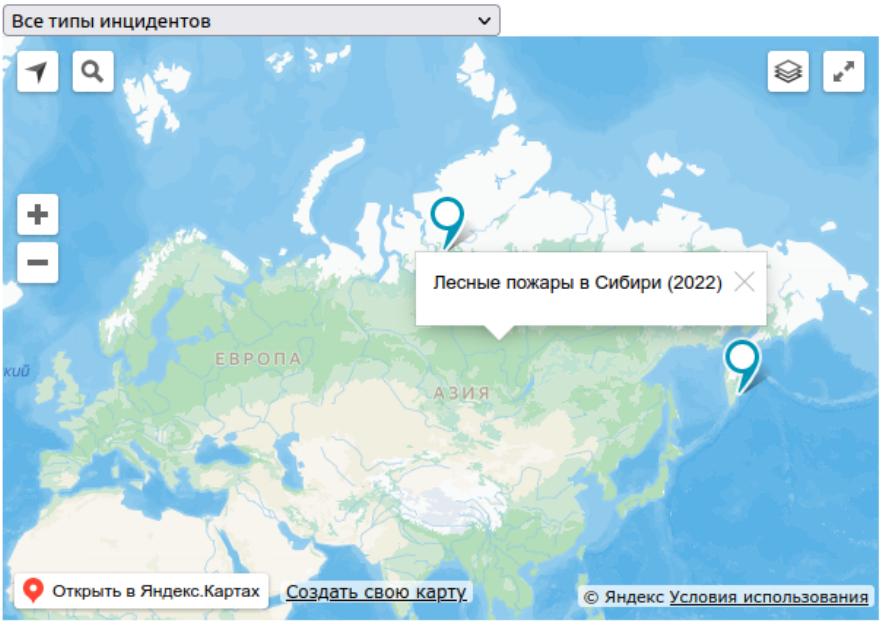
```
    return finishedAt - startedAt
}

func update(r repo, data, updated []entity.Incident) int64 {
    startedAt := time.Now().UnixNano()
    for i := range data {
        r.Update(context.Background(), data[i].Id, updated[i])
    }
    finishedAt := time.Now().UnixNano()

    return finishedAt - startedAt
}

func delete(r repo, data []entity.Incident) int64 {
    startedAt := time.Now().UnixNano()
    for i := range data {
        r.Delete(context.Background(), data[i].Id)
    }
    finishedAt := time.Now().UnixNano()

    return finishedAt - startedAt
}

func getById(r repo, data []entity.Incident) int64 {
    startedAt := time.Now().UnixNano()
    for i := range data {
        r.GetById(context.Background(), data[i].Id)
    }
    finishedAt := time.Now().UnixNano()

    return finishedAt - startedAt
}

func getByType(r repo, types []int) int64 {
    startedAt := time.Now().UnixNano()
    for i := range types {
        r.GetByType(context.Background(), types[i])
    }
    finishedAt := time.Now().UnixNano()

    return finishedAt - startedAt
}

func run(r repo, incidents, updated []entity.Incident, types []
   int) [5]int64 {
    res := [5]int64{}

    res[0] = create(r, incidents)
    res[1] = update(r, incidents, updated)
    res[3] = getById(r, incidents)
    res[4] = getByType(r, types)
```

Листинг Г.4 – Проведение исследования. Часть 4

```
    res[2] = delete(r, incidents)

    return res
}

func main() {
    cfg, err := config.Load()
    if err != nil {
        log.Fatalf("Config error: %s", err)
    }
    pgAvg := [5]int64{}
    mongoAvg := [5]int64{}
    for i := 0; i < reps; i++ {
        pg, err := postgres.New(cfg.Database.URI)
        if err != nil {
            log.Fatal(err, "benchmarking - postgres.New")
        }
        mongo, err := mongo.New(cfg.MongoDB.URI)
        if err != nil {
            log.Fatal(err, "benchmarking - mongo.New")
        }

        pgrepo := pgrepo.NewIncidentRepository(pg)
        mongorepo := mongorepo.NewIncidentRepository(mongo)

        incidents := randomIncidents(n)
        updated := randomIncidents(n)
        types := randomTypes(n)

        pgRes := run(pgrepo, incidents, updated, types)
        mongoRes := run(mongorepo, incidents, updated, types)

        for i := 0; i < 5; i++ {
            pgAvg[i] += pgRes[i]
            mongoAvg[i] += mongoRes[i]
        }

        pg.Close()
        mongo.Close()
    }
    for i := 0; i < 5; i++ {
        pgAvg[i] /= reps
        pgAvg[i] = int64(float64(pgAvg[i]) * 1e-6)

        mongoAvg[i] /= reps
        mongoAvg[i] = int64(float64(mongoAvg[i]) * 1e-6)
    }
    fmt.Println(pgAvg)
    fmt.Println(mongoAvg)
}
```

ПРИЛОЖЕНИЕ Д
Web-интерфейс

Рисунок Д.1 демонстрирует пользовательский Web-интерфейс.



Список экологических инцидентов:

Массовая гибель морских животных на Камчатке

Лесные пожары в Сибири (2022)

Разлив дизельного топлива в Норильске

Рисунок Д.1 – Web-интерфейс