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, искусственный интеллект и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

База данных учета приборов мониторинга лесных пожаров

Студент ИУ9-62Б _____ М.А.Пехова
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Д.П.Посевин
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Интеграция баз данных в сервисы и приложения, предназначенные для мониторинга природных рисков, является одной из ключевых задач современной разработки. В эпоху ускоренной цифровизации, когда своевременные данные становятся основой управленческих решений и стратегического планирования, требуется обеспечить быстрый и надёжный доступ к информации, поступающей от различных источников. Это особенно важно в системах мониторинга лесных пожаров, где необходимо обрабатывать большой поток телеметрии от датчиков температуры, влажности, содержания CO_2 и других показателей окружающей среды.

Приложения, выполняющие такие задачи, востребованы в лесном хозяйстве, экологии, страховании, туризме и государственных структурах, ответственных за охрану природных ресурсов. Они позволяют в режиме реального времени обнаруживать аномалии, формировать прогнозы пожарной опасности, оперативно предупреждать ответственных лиц и координировать действия пожарных служб. Интеграция базы данных обеспечивает централизованное хранение, валидацию и историзацию поступающих данных, упрощая последующий анализ и подготовку управленческих отчётов.

Актуальность работы обусловлена двумя основными факторами. Во-первых, объём телеметрии, генерируемой сотнями распределённых приборов, может исчисляться десятками миллионов записей в сутки; без правильно спроектированной БД такие данные невозможно обрабатывать с необходимой скоростью. Во-вторых, для совместной работы лесничеств, диспетчерских центров и мобильных групп требуется единое, согласованное приложение, поддерживающее синхронизацию и разграничение доступа.

Интеграция базы данных в такие системы повышает их стабильность и расширяемость, обеспечивая конкурентные преимущества на рынке информационных технологий для природоохранного сектора.

Таким образом, создание высокопроизводительной и надёжной базы данных в приложении мониторинга лесных пожаров является актуальной и

востребованной задачей, способствующей повышению точности прогнозов, удобства эксплуатации и конкурентоспособности конечного продукта.

Цель данной работы — разработать, внедрить и интегрировать базу данных в приложение мониторинга лесных пожаров, обеспечив эффективное логирование телеметрии, её анализ, хранение исторических срезов и поддержку совместной работы пользователей.

Для достижения цели необходимо решить следующие задачи:

1. Анализ требований к базе данных: выявить функциональные и нефункциональные требования к БД; определить основные сущности; установить требования к производительности, безопасности и надёжности.
2. Проектирование структуры базы данных: Разработать ER-диаграмму, нормализовать таблицы, задать связи «многие-ко-многим» через промежуточные отношения, выбрать подходящие типы данных и индексы.
3. Создание физической схемы базы данных: создать таблицы, первичные и внешние ключи, ограничения CHECK, триггеры для автоматического расчёта показателей, представления для аналитических выборок.
4. Выбор технологий и инструментов: определить подходящие технологии и инструменты для реализации базы данных, учитывая требования к производительности, масштабируемости и совместимости с мобильным приложением.
5. Разработка API для взаимодействия с базой данных: спроектировать REST-интерфейс для работы с объектами БД, реализовать энд-пойнты CRUD и агрегирующие запросы, обеспечить сериализацию и валидацию данных.

6. Тестирование и отладка: провести тестирование разработанной базы данных на соответствие требованиям, выявить и устранить возможные ошибки и недочеты в функциональности и производительности.
7. Подключение веб-клиента и сторонних приложений к REST-API, предоставленному Django REST Framework; обеспечить выдачу данных в JSON и возможность CRUD-операций через открытые энд-пойнты.
8. Оптимизация и масштабирование: провести оптимизацию производительности базы данных и ее масштабирование для обеспечения эффективной работы при увеличении объема данных и нагрузки.
9. Документация и поддержка: создать документацию к базе данных, описывающую ее структуру, функциональность и способы использования, а также обеспечить поддержку и сопровождение разработанной базы данных после ее внедрения.

1. Обзор предметной области

Информационная система учета приборов мониторинга лесных пожаров предназначена для сбора, хранения и анализа данных о параметрах окружающей среды, посылаемых устройствами. Основная цель данной системы – обнаружение приборов, которые передают полученные измерения о состоянии окружающей среды, определение их характеристик и географических координат. Это важно для анализа вероятности возникновения пожара в конкретной локации, соответственно и для его своевременного детектирования.

Основные сущности системы:

1. локация (Location):
    a. location name: название локации;
    b. description: описание локации.
2. экземпляры приборов (Device):
    a. inventory number: инвентарный номер;
    b. device type: тип прибора;
    c. latitude: градус широты;
    d. longitude: градус долготы;
    e. installation date: дата установки.
3. параметры окружающей среды (EnvironmentalParameters):
    a. inventory number: инвентарный номер;
    b. recorded at: время и дата записи;
    c. temperature: температура;
    d. humidity: влажность;
    e. co2 level: уровень co2;
    f. processed: флаг обработки.
4. проанализированная информация (AnalyzedInformation):

a. number of analysis: номер анализа;
b. analyzed at: дата и время анализа;
c. fire hazard: вероятность возгорания.

5. оповещения (Alarm):
    a. number of alarm: номер оповещения;
    b. status of alarm: статус;
    c. alarm level: уровень тревоги;
    d. date of alarm: дата;
    e. time of alarm: время.
6. инцидент (Incident):
    a. number of incident: номер инцидента;
    b. status of incident: статус;
    c. description: описание;
    d. time of detection: время обнаружения;
    e. time of resolve: время разрешения инцидента;
    f. location name: имя локации;
    g. time window start: начало временного окна;
    h. time window end: конец временного окна.

Взаимосвязи между сущностями:

- каждое устройство закрепляется за конкретной локацией, то есть в одной локации может находиться несколько устройств, тогда как каждое устройство связано только с одной локацией;
- каждое устройство генерирует множество записей с данными об окружающей среде, фиксируемыми в таблице параметров;
- один и тот же набор исходных данных, зафиксированный в записи параметров окружающей среды, может быть использован для проведения одного конкретного анализа;
- каждое проанализированное событие приводит к формированию максимум одного оповещения;

- каждое оповещение может приводить к созданию одного инцидента, поскольку оно сигнализирует о конкретной потенциальной угрозе, требующей реагирования; в то время как один инцидент может включать в себя несколько оповещений.

Эта модель позволяет структурировать данные, необходимые для поиска устройств, считывающих информацию о состоянии окружающей среды, и предоставляет возможности для анализа и визуализации полученных данных, что является ключевым требованием для обеспечения надежного мониторинга и детектирования лесных пожаров и впоследствии построении корректной реляционной модели базы данных.

2. Проектирование структуры базы данных

2.1. Модель «сущность-связь»

На основе сформулированных требований к базе данных была построена модель «сущность-связь» (см. рисунок 1).

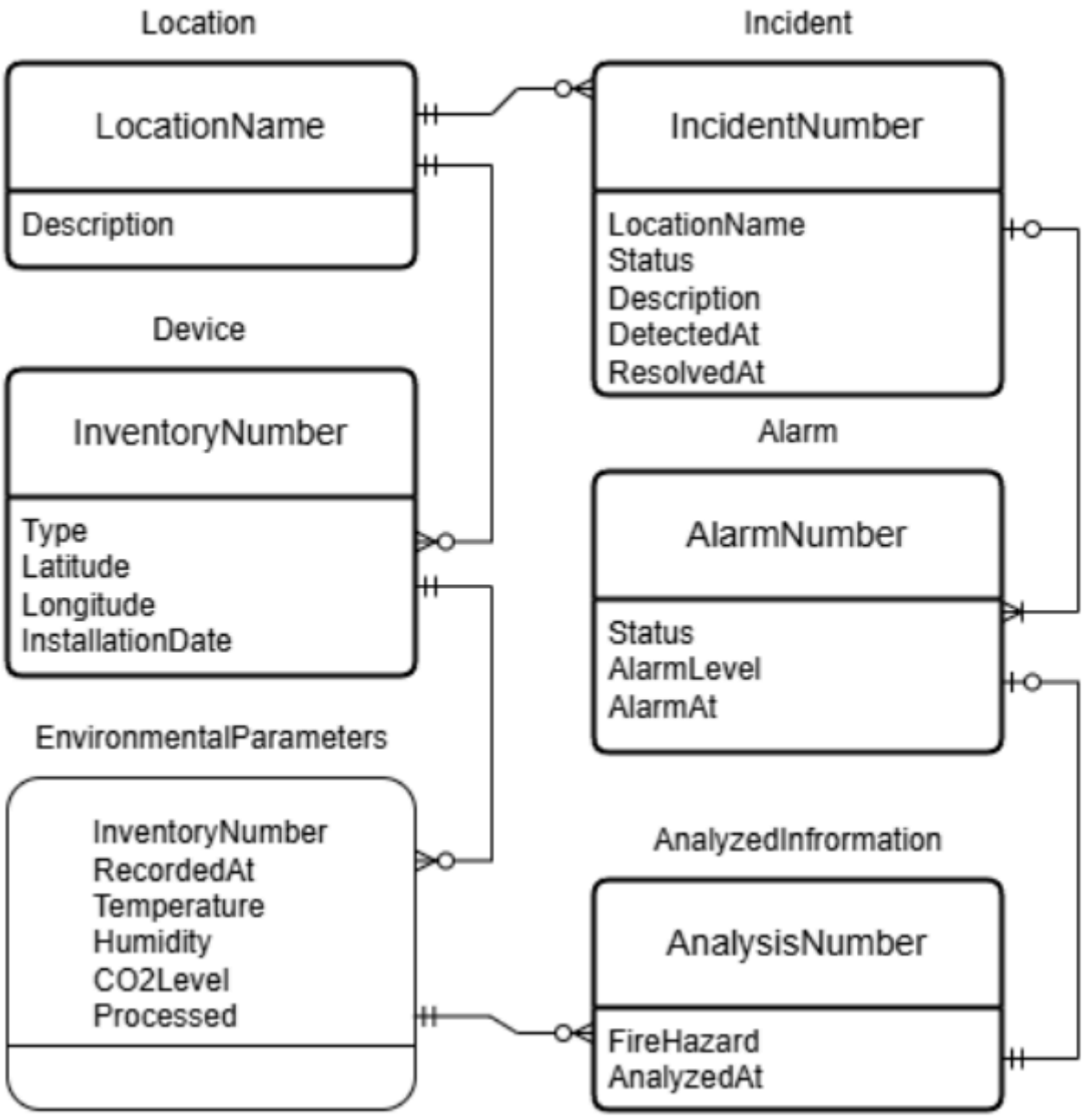

Рисунок 1 – ER-модель приложения мониторинга лесных пожаров

2.2. Модель семантических объектов

В отличие от ER-модели, которая в первую очередь фокусируется на структуре данных и их взаимосвязях, модель семантических объектов акцентирует внимание на реальных сущностях предметной области и их атрибутах, что позволяет более глубоко понять и описать бизнес-логику системы. Второй важный принцип построения модели заключается в установлении логических связей между сущностями для создания целостной структуры данных. Связи между сущностями определяются исходя из реальных взаимосвязей объектов в предметной области, например, как устройства могут использовать несколько частотных диапазонов, а частотные диапазоны могут быть связаны с различными географическими координатами. Эти связи не только обеспечивают целостность данных, но и позволяют эффективно организовать их хранение и обработку. Создание и визуализация ER-диаграммы помогает наглядно представить структуру данных, облегчая понимание и дальнейшую разработку системы. Таким образом, модель семантических объектов становится основой для разработки базы данных, которая поддерживает задачи мониторинга, контроля и исследования радиочастотного спектра.

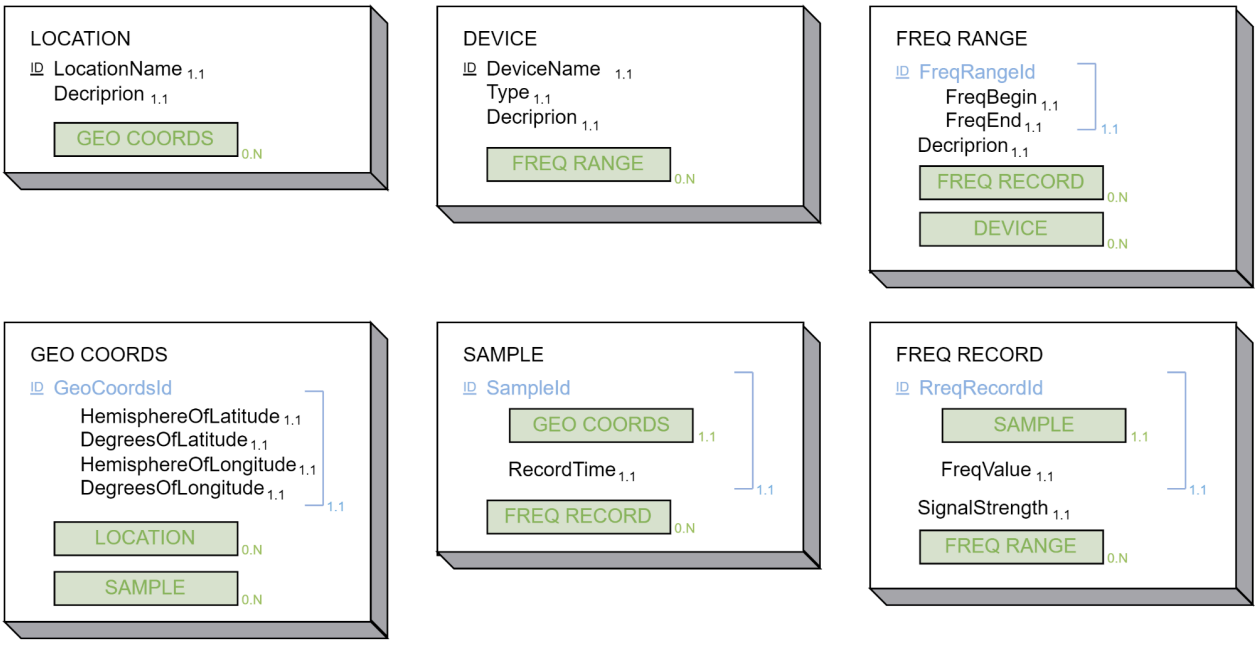

Рисунок 2 – Модель семантических объектов приложения анализа радиоэфира

2.3. Нормализация базы данных

Оптимизация структуры базы данных через нормализацию предполагает последовательное преобразование информации в соответствии с требованиями шести уровней нормальных форм (от 1НФ до 6НФ). Каждая ступень этой иерархии решает конкретные проблемы: минимизирует дублирование данных, исключает противоречия и ошибки при операциях обновления или удаления. Рассмотрим, как эти принципы применяются к системе отслеживания лесных пожаров, где точность и согласованность данных критически важны для оперативного анализа угроз.

2.3.1. Первая нормальная форма

1НФ требует, чтобы в таблицах не было повторяющихся и/или многозначных столбцов.

1. локация (location):

Таблица location уже соответствует 1НФ, так как все поля содержат атомарные данные: location_id, location_name, description — все значения уникальны и не разбиваются на составные части.

2. устройство (device):

В таблице device все поля также атомарные: device_id, location_id, inventory_number, type, date_of_installation, latitude, longitude — никаких повторяющихся групп нет.

3. параметры окружающей среды (environmental_parameters):

Все поля также соответствуют 1НФ: recorded_data_id, device_id, temperature, humidity, co2_level, recorded_at, processed — никаких групп повторяющихся данных нет.

4. проанализированная информация (analyzed_information):

Таблица analyzed_information также соответствует 1НФ, все поля атомарные: analysis_id, recorded_data_id, fire_hazard, analyzed_at.

5. оповещения (alarm):

Все поля в таблице alarm атомарные: alarm_id, analysis_id, incident_id, alarm_level, status, alarm_at.

6. инциденты (incident):

В таблице incident все поля атомарные: incident_id, location_id, status, description, detected_at, resolved_at, time_window_start, time_window_end.

В каждой таблице все поля представляют собой атомарные значения, нет массивов или вложенных структур. Следовательно, все таблицы находятся в 1НФ.

2.3.2. Вторая нормальная форма

2НФ требует, чтобы каждый неключевой столбец зависел от всего первичного ключа, а не от его части.

Во всех таблицах базы данных первичный ключ — одиночное поле id (surrogate PK), составных ключей нет, следовательно частичных зависимостей быть не может.

Все таблицы соответствуют второй нормальной форме, так как все неключевые атрибуты зависят от всего первичного ключа.

2.3.3. Третья нормальная форма

3НФ (и более строгая НФБК) требует отсутствия транзитивных зависимостей вида «PK → поле1 → поле2», когда поле2 зависит от поля1, а поле1 зависит от первичного ключа. Рассмотрев архитектуру базы данных получим, что в каждой таблице мы имеем зависимость вида primary_key → (все остальные поля), и больше никаких нет, поскольку нет лишних атрибутов, которые повторяют данные из других таблиц или дублируют их. Например, если бы в таблице device дублировали location_name (чтобы быстро смотреть название локации без джойна), то это потенциально могло бы нарушать 3НФ.

1. локация (location):

В таблице location нет транзитивных зависимостей. Поля location_name и description зависят только от location_id, и больше ничего не зависит от других полей.

2. устройство (device):

В таблице device нет транзитивных зависимостей. Все атрибуты зависят только от device_id.

3. параметры окружающей среды (environmental_parameters):

В таблице environmental_parameters нет транзитивных зависимостей. Все атрибуты зависят от recorded_data_id.

4. проанализированная информация (analyzed_information):

В таблице analyzed_information нет транзитивных зависимостей. Все атрибуты зависят от analysis_id.

5. оповещения (alarm):

В таблице alarm нет транзитивных зависимостей. Все атрибуты зависят от alarm_id.

6. инциденты (incident):

В таблице incident нет транзитивных зависимостей. Все атрибуты зависят от incident_id.

Таким образом, база данных находится в третьей нормальной форме.

2.3.4. Бойс-Кодд нормальная форма

BCNF — это усиленная версия 3НФ. Требование НФБК: для любой нетривиальной функциональной зависимости X → Y, детерминанта X должна быть кандидатом в ключи.

Во всех таблицах детерминантой является только первичный ключ (location_id, device_id и остальные), а он, естественно, кандидат в ключи.

2.3.5. Четвертая нормальная форма

4НФ требует отсутствия нетривиальных многозначных зависимостей. Чаще всего многозначные зависимости возникают, если в одной таблице хранятся «списки» значений, повторяющихся для одних и тех же ключей. Например, хрестоматийный пример выглядит так, например, «автор книги» и «иллюстратор книги» — одна и та же книга может иметь несколько авторов и нескольких иллюстраторов, и тогда таблицу приходится разносить.

В представленной схеме не просматриваются никакие атрибуты, которые могли бы одновременно принимать множество значений при одном и том же ключе. Все потенциальные «множественные» связи (например, устройство может иметь множество записей environmental_parameters) реализованы через связь один-ко-многим (device_id → множество записей в environmental_parameters) при помощи отдельной таблицы.

2.3.6. Пятая нормальная форма

5НФ ориентирована на устранение нетривиальных соединительных зависимостей между более, чем двумя таблицами, которые не следуют из ключей.

То есть, 5НФ требует, чтобы все возможные связи между элементами таблицы полностью задавались «через ключи» и не было таких «огромных» таблиц, которые можно ещё раз «распилить» на три и более таблиц (не на две) без потери данных и без появления аномалий при обновлении.

Как правило, проблемы с 5НФ, — это сложные «многие-ко-многим-ко-многим» отношения (когда одна таблица может быть результатом соединения сразу нескольких в каком-то нетривиальном виде). Например, Alarm опосредует потенциальную N:M-связь между AnalyzedInformation и Incident, сама являясь полноценной бизнес-сущностью; дополнительных скрытых зависимостей не возникает.

Таким образом, 5НФ соблюдена.

2.3.7. Шестая нормальная форма

Строгая 6 НФ (один факт — одна таблица) редко применяется в OLTP-части из-за большого числа JOIN-ов, однако при высокочастотной телеметрии и требовании версионирования отдельных атрибутов она становится разумной альтернативой “широкой” таблице. В нашей текущей реализации все показания поступают пакетом, поэтому 3 НФ/BCNF достаточно; при переходе к более мелкому шагу времени либо спарс-измерениям

потребуется рассмотреть якорное моделирование или гибридное time-series-хранилище.

2.3.8. Результат нормализации

Схема удовлетворяет 1 НФ, 2 НФ, 3 НФ, BCNF, 4 НФ и 5 НФ; 6 НФ сознательно не применяется как избыточная для оперативной системы. Такая степень нормализации минимизирует дублирование и исключает аномалии вставки, удаления и обновления, обеспечивая надёжное хранение данных для мониторинга лесных пожаров.

2.4. Преобразование модели «сущность-связь» в реляционную

Первый этап – при преобразовании ER-диаграммы в реляционную схему мы «разворачиваем» все связи «многие-ко-многим» в отдельные таблицы-связки и одновременно вводим явные первичные ключи у тех сущностей, которые логически зависят от других (т. е. «слабых» сущностей).

Второй этап – для каждой основной (не-связующей) таблицы создаём однозначный идентификатор вида <Сущность>ID (surrogate-PK), выделяем из атрибутов естественные ключи-кандидаты и оставляем их помеченными как UNIQUE, а сам PK называем просто id.

1. Ассоциативные (M:N) связи не использовались (в предметной области таких нет), а все отношения 1:N и N:1 оформлены через FK с on_delete-семантикой.
2. Суррогатные ключи: у каждой модели есть авто-поле id.
3. Кандидаты в ключи: добавлено поле unique=True на поля inventory_number в Device и location_name в Location.

После описанных итераций, включающих в себя все этапы нормализации, мы получаем верную реляционную модель (см. рисунок 3).

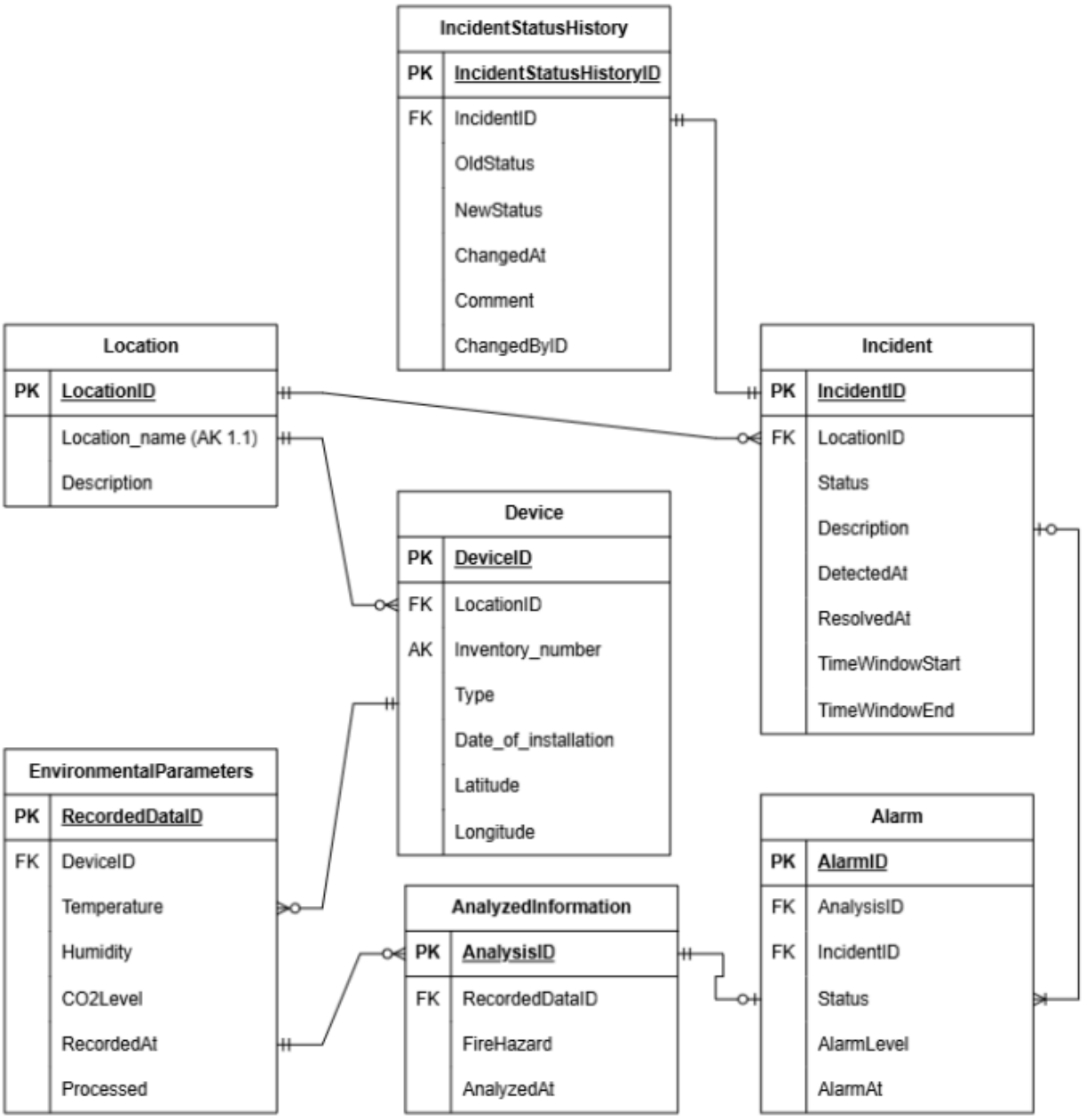

Рисунок 3 – Реляционная модель приложения мониторинга лесных пожаров

Далее для проверки была построена таблица ограничений кардинальностей связей между моделями (таблица 1).

Таблица 1 – Ограничения кардинальных чисел

Зависимости	Кардинальные числа

Родитель	Ребенок	Тип	Максимум	Минимум
LOCATION	DEVICE	Non identifying	1:N	M-O
DEVICE	ENVIRONMENTAL_PARAMETERS	Non identifying	1:N	M-O
ENVIRONMENTAL PARAMETERS	ANALYZED_INFORMATION	Non identifying	1:N	M-O
ANALYZED INFORMATION	ALARM	Non identifying	1:N	M-O
INCIDENT	ALARM	Identifying	1:N	M-O
INCIDENT	INCIDENT_STATUS_HISTORY	Identifying	1:N	M-O
ALARM	INCIDENT	Identifying	N:1	O-M

Следом рассмотрим, какие действия требуются при вставке, изменении, или удалении того или иного элемента для каждой из таблиц (таблицы 2-9).

Таблица 2 – Действия над таблицами LOCATION и DEVICE 1:N M–O

Операция	Действие над LOCATION	Действие над DEVICE
Вставка	–	подбор существующей записи-родителя
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 3 – Действия над таблицами DEVICE и ENVIRONMENTAL_PARAMETERS 1:N M–O

Операция	Действие над GEOCOORDS	Действие над ENVIRONMENTAL_PARAMETERS

Вставка	–	подбор родителя
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 4 – Действия над таблицами ENVIRONMENTAL_PARAMETERS и ANALYZED_INFORMATION 1:N M–O

Операция	Действие над ENVIRONMENTAL_PARAMETERS	Действие над ANALYZED_INFORMATION
Вставка	–	подбор родителя
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 5 – Действия над таблицами ANALYZED_INFORMATION и ALARM 1:N M–O

Операция	Действие над ANALYZED_INFORMATION	Действие над ALARM
Вставка	–	подбор родителя
Изменение	запрет	запрет
Удаление	запрет (PROTECT, необходимо предварительно удалить/переназначить alarms)	–

Таблица 6 – Действия над таблицами INCIDENT и ALARM 1:N O–M

Операция	Действие над INCIDENT	Действие над ALARM
Вставка	– (может быть NULL)	родитель указывается опционально

Изменение	запрет	запрет
Удаление	SET NULL — инцидент удаляется, поле incident_id в alarm обнуляется	–

Таблица 7 – Действия над таблицами INCIDENT и INCIDENT_STATUS_HISTORY 1:N M–O

Операция	Действие над INCIDENT	Действие над INCIDENT_STATUS_HISTORY
Вставка	–	создаётся автоматически при первом сохранении инцидента
Изменение	запрет	запрет
Удаление	каскадное удаление	–

Таблица 8 – Действия над таблицами LOCATION и INCIDENT 1:N M–O

Операция	Действие над LOCATION	Действие над INCIDENT
Вставка	–	родитель указывается опционально
Изменение	запрет	запрет
Удаление	запрет (PROTECT, если есть связанные инциденты)	–

После этого для каждого объекта были построены таблицы со следующими полями: имя атрибута, тип данных, тип ключа, статус нулевого значения, замечания. Рассмотрим каждую из них.

Таблица 9 – Атрибуты объекта LOCATION

Название	Тип	Ключ	NULL-статус	Замечания

LocationID	Int	Primary Key	NOT NULL	Surrogate Key IDENTITY (1, 1)
LocationName	Varchar (255)	Alternative Key	NOT NULL	Unique (AK1.1)
Description	Text	—	NULL	

Таблица 10 – Атрибуты объекта DEVICE

Название	Тип	Ключ	NULL-статус	Замечания
DeviceID	Int	Primary Key	NOT NULL	Surrogate Key
LocationID	Int	Foreign Key	NOT NULL	Location
InventoryNumber	Varchar(50)	Alternative Key	NOT NULL	UNIQUE (AK1.1)
Type	Varchar(50)	—	NULL	—
DateOfInstallation	Date	—	NULL	—
Latitude	Decimal(10, 6)	—	NULL	Широта
Longitude	Decimal(10, 6)	—	NULL	Долгота

Таблица 11 – Атрибуты объекта ENVIRONMENTAL_PARAMETERS

Название	Тип	Ключ	NULL-статус	Замечания
EnvParamID	Int	Primary Key	NOT NULL	Surrogate Key
DeviceID	Int	Foreign Key	NOT NULL	Device
Temperature	Decimal(5, 2)	—	NULL	CHECK −50…200 °C
Humidity	Decimal(5, 2)	—	NULL	CHECK 0…100 %
CO2_Level	Decimal(10, 2)	—	NULL	CHECK ≥ 0 ppm

RecordedAt	Datetime	—	NOT NULL	default = NOW()
Processed	Boolean	—	NOT NULL	default = FALSE

Таблица 12 – Атрибуты объекта ANALYZED_INFORMATION

Название	Тип	Ключ	NULL-статус	Замечания
AnalysisID	Int	Primary Key	NOT NULL	Surrogate Key
EnvParamID	Int	Foreign Key	NOT NULL	EnvironmentalParameters
FireHazard	Decimal(5, 2)	—	NULL	% опасности 0–100
AnalyzedAt	Datetime	—	NOT NULL	AUTO_TIMESTAMP

Таблица 13 – Атрибуты объекта INCIDENT

Название	Тип	Ключ	NULL-статус	Замечания
IncidentID	Int	Primary Key	NOT NULL	Surrogate Key
LocationID	Int	Foreign Key	NULL	Location
TimeWindowStart	Datetime	—	NOT NULL	—
TimeWindowEnd	Datetime	—	NULL	—
DetectedAt	Datetime	—	NOT NULL	AUTO_TIMESTAMP
ResolvedAt	Datetime	—	NULL	—
Status	Varchar(20)	—	NOT NULL	open / investigation / closed
Description	Text	—	NULL	—

Таблица 14 – Атрибуты объекта ALARM

Название	Тип	Ключ	NULL-статус	Замечания
AlarmID	Int	Primary Key	NOT NULL	Surrogate Key
AnalysisID	Int	Foreign Key	NOT NULL	AnalyzedInformation; ON DELETE PROTECT
IncidentID	Int	Foreign Key	NULL	Incident, ON DELETE SET NULL
AlarmLevel	Varchar(20)		NOT NULL	low / medium / high / critical
Status	Varchar(20)	—	NOT NULL	active / acknowledged / resolved (по умолчанию active)
AlarmAt	Datetime	—	NOT NULL	Время возникновения тревоги

Таблица 15 – Атрибуты объекта INCIDENT_STATUS_HISTORY

Название	Тип	Ключ	NULL-статус	Замечания
IncidentStatusHistoryID	Int	Primary Key	NOT NULL	Surrogate Key
IncidentID	Int	Foreign Key	NOT NULL	Incident
OldStatus	Varchar(20)	—	NULL	—
NewStatus	Varchar (20)	—	NOT NULL	—
ChangedAt	Datetime	—	NOT NULL	AUTO_TIMESTAMP
ChangedBy	Int		NULL	

Comment	Text	—	NULL	—

Таблица 17 – Атрибуты объекта DEVICE_FREQRANGE_INT

Название	Тип	Ключ	NULL-статус	Замечания
DeviceID	Int	Primary Key Foreign Key	NOT NULL	
FreqRangeID	Int	Primary Key Foreign Key	NOT NULL	

2.5. Преобразование модели семантических объектов в реляционную

Семантический объект LOCATION преобразован в следующую таблицу: LOCATION (LocationName, Description).

Семантическая связь между объектами LOCATION и GEOCOORDS преобразована в следующую таблицу: LOCATION_GEOCOORDS_INT (*LocationID*, *GeoCoordsID*).

Семантический объект GEOCOORDS преобразован в следующую таблицу: GEOCOORDS (HemisphereOfLatitude, DegreesOfLatitude, HemisphereOfLongitude, DegreesOfLongitude).

Семантический объект SAMPLE преобразован в следующую таблицу: SAMPLE (*GeoCoordsID*, RecordTime).

Семантический объект SAMPLE преобразован в следующую таблицу: SAMPLE (*SampleID*, FreqValue, SignalStrength).

Семантическая связь между объектами FREQRECORD и FREQRANGE преобразована в следующую таблицу: FREQRANGE_FREQRECORD_INT (*FreqRangeID*, *FreqRecordID*).

Семантический объект FREQRANGE преобразован в следующую таблицу: FREQRANGE (FreqBegin, FreqEnd, Description).

Семантическая связь между объектами FREQRANGE и DEVICE преобразована в следующую таблицу: DEVICE_FREQRANGE_INT (*DeviceID*, *FreqRangeID*).

Семантический объект DEVICE преобразован в следующую таблицу: DEVICE (DeviceName, Type, Description).

2.6. Сопоставление результатов проектирования

Изучив получившиеся две реляционные модели после преобразования из ER-модели и модели семантических объектов, нетрудно заметить, что они полностью идентичны, что с наибольшей вероятностью означает верность построения всех моделей и общего проектирования модели базы данных приложения анализа радиоэфира.

3. Выбор технологий и инструментов

В процессе разработки веб-ориентированной системы мониторинга лесных пожаров важным этапом являлся выбор подходящей системы управления базами данных. Этот выбор оказал значительное влияние на производительность, удобство разработки и масштабируемость приложения.

Среди наиболее популярных СУБД для ОС Android можно выделить следующие со своими преимуществами и недостатками:

- SQLite:
    - преимущества:
        - встроена в Android SDK, не требует дополнительных библиотек;
        - хорошо задокументирована и широко используется;
        - поддерживает большинство стандартных SQL-функций;
    - недостатки:
        - не самая быстрая при работе с большими объемами данных;
        - требует написания значительного количества SQL-запросов, что может усложнять код;
        - нет автоматической синхронизации данных с облаком;
- Room (поверх SQLite):
    - преимущества:
        - облегчает работу с SQLite, предоставляя объектно-ориентированный интерфейс;
        - проверка запросов на этапе компиляции;
        - поддержка реактивного программирования;
    - недостатки:
        - все еще использует SQLite в своей основе, что может привести к аналогичным проблемам с производительностью при больших объемах данных;
        - требует аннотаций и написания DAO (Data Access Objects), что может усложнять код;

- Realm
    - преимущества:
        - высокая производительность при вставке и чтении больших объемов данных благодаря своей архитектуре;
        - простота использования и более читаемый код благодаря объектно-ориентированному подходу;
        - поддержка реактивного программирования;
        - встроенная синхронизация данных с облаком (Realm Sync);
    - недостатки:
        - больший размер библиотеки по сравнению с SQLite;
        - лицензирование может быть дорогим для коммерческих проектов с использованием Realm Sync;
        - ограниченная поддержка сложных SQL-запросов, что может быть критично для некоторых приложений;
- ObjectBox:
    - преимущества:
        - высокая производительность благодаря уникальной структуре хранения данных;
        - простой в использовании объектно-ориентированный API;
        - поддержка реактивного программирования;
    - недостатки:
        - новая и менее зрелая технология по сравнению с SQLite и Realm;
        - меньше документации и примеров в сообществе разработчиков.

Для системы анализа радиочастот, основная задача которой заключается в обработке и анализе больших объемов данных, скорость вставки и чтения данных является критически важной согласно тестов. Realm показала себя как одна из самых производительных СУБД в этом контексте [1, 2]. Благодаря своей

архитектуре, Realm обеспечивает высокую производительность при работе с большими объемами данных, что особенно важно для системы, где ожидается интенсивная запись данных радиосигналов.

Кроме того, Realm предоставляет простой и интуитивно понятный объектно-ориентированный интерфейс, что упрощает разработку и делает код более читаемым и поддерживаемым. Поддержка реактивного программирования позволяет легко интегрировать изменения данных в пользовательский интерфейс, что упростило разработку.

Несмотря на то, что Realm имеет свои недостатки, такие как больший размер библиотеки и лицензионные ограничения при использовании Realm Sync, преимущества в виде быстродействия и простоты использования стали решающими в выборе между прочими СУБД для данной работы. Таким образом, выбор в пользу Realm был обоснован ее способностью эффективно обрабатывать большие объемы данных, что является критичным условием для успешного выполнения задач данной курсовой работы.

4. Разработка схемы базы данных

При разработке базы данных на Realm использовалась объектная модель данных, а не реляционная. В Realm данные хранятся в виде объектов, а не таблиц, как в реляционной модели. Объекты Realm соответствуют классам в коде, и связи между этими объектами также определяются как свойства этих классов. Это обеспечивает более естественную и интуитивно понятную работу с данными в объектно-ориентированном стиле.

Первым шагом в реализации стала настройка проекта в Android Studio и добавление зависимости Realm в файл `build.gradle` (см. листинг 1). После добавления зависимости необходимо было синхронизировать проект с Gradle и инициализировать Realm в классе приложения (см. листинг 2).

Следующим шагом в процессе разработки базы данных стало формирование классов, описывающих объекты базы данных (см. листинги 3-8). Эти классы включают в себя методы для получения и установки значений полей, обеспечивая удобный и интуитивно понятный доступ к данным. Для каждой сущности модели данных, такой как Location, Device, FrequencyRange, GeoCoords, Sample и FrequencyRecord, создаются соответствующие классы с аннотацией `@RealmClass` и наследованием от `RealmObject`. В этих классах определяются атрибуты, соответствующие полям базы данных, а также методы `get` и `set` для каждого атрибута. Методы тривиального взаимодействия с объектами позволяют управлять значениями полей объектов, поддерживая инкапсуляцию данных и способствуя сохранению целостности и согласованности данных при взаимодействии с базой данных [3].

5. Разработка API для взаимодействия с базой данных

Разработка API для взаимодействия с базой данных включает в себя создание функциональных возможностей для управления данными [4], таких как создание новых объектов, установление связей между существующими объектами, а также выполнение сложных запросов для извлечения данных [5]. Одним из ключевых аспектов является удобное создание новых объектов, что включает в себя инициализацию объектов с заданными атрибутами и добавление их в базу данных с помощью транзакций. В Realm это достигается через использование метода `executeTransaction`, который гарантирует атомарность, согласованность, изоляцию и долговечность (ACID) операций. Например, для создания нового объекта Location с заданными параметрами можно использовать достаточно лаконичную конструкцию, указанную в листинге 9.

Кроме создания новых объектов, важной задачей является управление связями между существующими объектами. Например, для установления связи между `Location` и `GeoCoords` достаточно добавить соответствующий объект в `RealmList` и сохранить изменения в базе данных. Это позволяет моделировать отношения "один ко многим" и "многие ко многим". Также необходимы методы для получения всех объектов класса, удовлетворяющих определенным условиям. В Realm для этого используются запросы на основе метода `where`, позволяющие фильтровать данные по различным критериям. Например, для получения всех объектов `Device` с типом `Station` можно использовать код, описанный в листинге 10.

Такие методы обеспечивают гибкость и мощные возможности для взаимодействия с базой данных, позволяя разработчикам эффективно управлять данными и выполнять сложные запросы с минимальными усилиями.

При разработке базы данных для системы анализа радиочастот в Realm особое внимание уделялось установлению и поддержанию связей между объектами. Однако, благодаря тому, что все связи "многие ко многим" имеют

минимальные кардинальные числа, равные нулю, отпадает необходимость проверки наличия связей при создании или удалении объектов. Это означает, что объект может существовать независимо от того, имеет ли он какие-либо связанные объекты, что значительно упрощает операции с базой данных. Например, устройство (Device) может быть создано или удалено без проверки на наличие связанных частотных диапазонов (FrequencyRange), так как минимальное количество таких связей равно нулю. Это снижает сложность кода, ускоряет выполнение операций и уменьшает вероятность возникновения ошибок, связанных с проверкой существования связей. Такой подход делает работу с базой данных более гибкой и удобной для разработчиков.

Для реализации каскадного удаления были созданы специальные методы, которые рекурсивно проходят по всем зависимым объектам и удаляют их. При этом используются транзакции для выполнения всех операций в рамках одной атомарной операции, что предотвращает частичное удаление данных и сохраняет целостность базы данных. Например, метод для удаления `GeoCoords` включает логику для удаления всех связанных `Sample`, он, в свою очередь, удаляет все связанные `FreqRecord`, а затем удаляются и сами объекты `Sample` и исходный объект `GeoCoords`.

6. Тестирование и отладка

Тестирование и отладка базы данных являются важными этапами в процессе разработки, обеспечивающими корректность и надежность работы системы. Процесс тестирования начался с написания тестовых сценариев, которые охватывают все основные функции базы данных, такие как создание, чтение, обновление и удаление (CRUD-операции) данных [6]. Тестирование проводилось как для отдельных методов, так и для их комбинаций, чтобы убедиться в правильности их взаимодействия. К примеру, проверялась возможность создания новых объектов, установления связей между ними и корректное сохранение этих изменений в базе данных. Особое внимание уделялось проверке соблюдения целостности данных и выполнению всех транзакций в соответствии с требованиями ACID.

Во время тестирования выявленные ошибки и недочеты были зафиксированы и проанализированы. На основе этого анализа были внесены необходимые исправления в код. Отладка проводилась с использованием инструментов логирования и дебаггинга, которые позволили отследить выполнение кода и выявить проблемные места. Логи используются для записи важных событий и состояний объектов в различные моменты времени, что помогает в диагностике и решении проблем. Повторное тестирование проводилось после каждого исправления, чтобы убедиться в устранении ошибок и проверить, что новые изменения не привели к возникновению других проблем. Этот итеративный процесс продолжался до тех пор, пока все тесты не были успешно пройдены, и база данных не стала работать стабильно и надежно.

7. Интеграция в мобильное приложение

Интеграция разработанной базы данных с приложением обработки радиоэфира была важным и тщательно спланированным этапом проекта. После создания и тестирования базы данных на основе Realm, она была встроена в существующую структуру мобильного приложения. Этот процесс включал обновление кода приложения для использования новых методов доступа к данным, определенных в Realm. Основные функциональные модули приложения, такие как модуль сбора данных радиосигналов, модуль анализа и визуализации, были адаптированы для взаимодействия с базой данных через реалм-ориентированные API. Это позволило обеспечить мгновенный доступ к необходимой информации и поддерживать актуальность данных в реальном времени [7]. В результате, приложение стало способным эффективно сохранять и обрабатывать большие объемы данных радиосигналов, улучшая производительность и расширяя функциональные возможности для конечных пользователей.

8. Документация и поддержка

Формирование документации к базе данных в коде являлось важной частью процесса разработки, направленной на обеспечение простоты понимания и поддержки кода для будущих разработчиков. Вся документация была интегрирована непосредственно в код с использованием комментариев и аннотаций JavaDoc [8]. Каждый класс, представляющий сущности базы данных, был снабжен подробным описанием его назначения и структуры. Методы также были подробно описаны, чтобы объяснить их функциональность и использование. Дополнительно, была включена информация о возможных значениях полей, ограничениях и особенностях использования. Такой подход обеспечил ясное и доступное понимание кода, облегчил процесс отладки и расширения функциональности базы данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно разработана и интегрирована база данных в мобильное приложение анализа радиоэфира на операционной системе Android (рисунки 4-6). Были проведены следующие этапы работы:

1. Анализ требований к базе данных: проведен анализ требований к базе данных, определены основные сущности и их атрибуты, а также выявлены ключевые критерии производительности и эффективности.
2. Выбор базы данных: на основе анализа был сделан выбор в пользу базы данных Realm, обеспечивающей высокую производительность и удобство использования для мобильных приложений.
3. Разработка и интеграция базы данных: была разработана структура базы данных, создана схема и реализована интеграция с мобильным приложением, обеспечивающая эффективное хранение и управление данными анализа радиоэфира.
4. Тестирование и оптимизация: проведено тестирование базы данных на соответствие требованиям, а также выполнена оптимизация производительности для обеспечения эффективной работы приложения. В качестве наиболее наглядного теста был написан генератор случайных значений в конкретном диапазоне, записывающий значения в базу данных и имитирующий поведение реальной программно-определяемой радиосистемы (см. листинг)

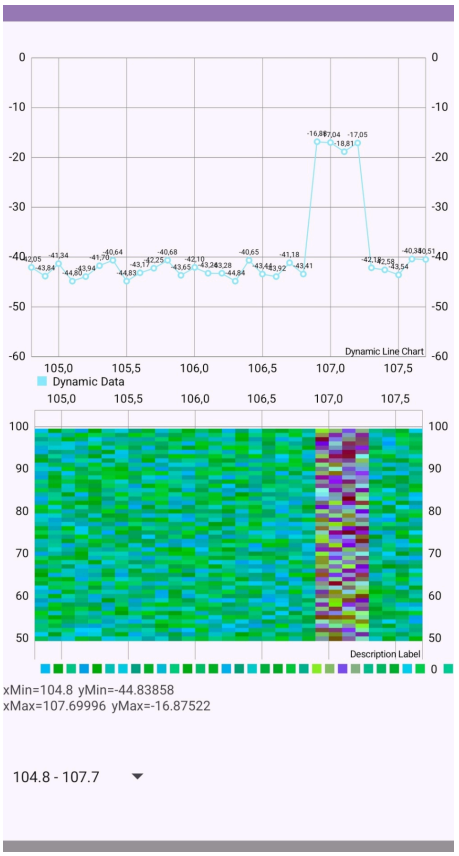

Рисунок 4 – Визуализация данных о диапазоне частот 104.8 – 107.7 FM

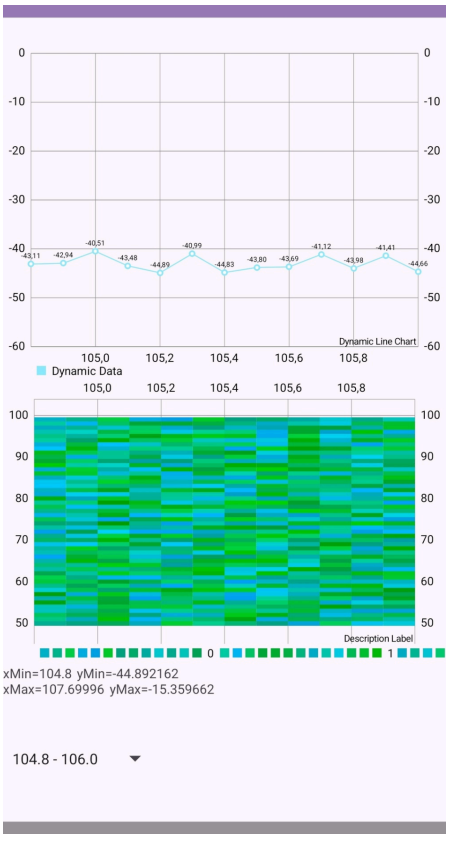

Рисунок 5 – Визуализация данных о диапазоне частот 104.8 – 106.0 FM

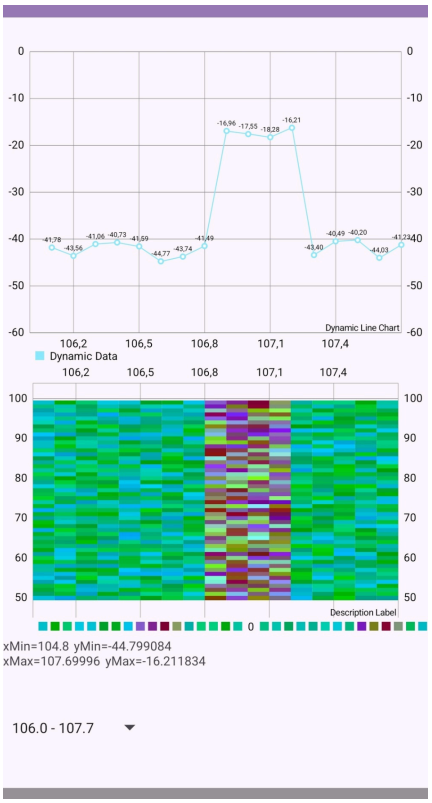

Рисунок 6 – Визуализация данных о диапазоне частот 106.0 – 107.7 FM

Разработка и интеграция базы данных в мобильное приложение анализа радиоэфира является важным шагом для обеспечения эффективного сбора, хранения и обработки данных. Выбор базы данных Realm обусловлен ее высокой производительностью и удобством использования, что позволяет считать ее оптимальным выбором для мобильных приложений на платформе Android среди прочих СУБД. Полученные результаты подтверждают возможность успешной реализации задачи интеграции базы данных в мобильное приложение анализа радиоэфира и ее значимость для современных информационных технологий.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Петров Г.В., Яблоков Е.Н. Сопоставление систем управления базами данных SQLite и Realm на платформе Android. В сборнике: Аэрокосмическое приборостроение и эксплуатационные технологии. Четвертая Международная научная конференция. Сборник докладов. В 2-х частях. Санкт-Петербург, 2023. С. 223-226.
2. Смелов О.А. Тестирование производительности систем управления базами данных для ос Android. Студенческий вестник. 2021. № 31-2 (176). С. 57-58.
3. Кучерова К.Н. Анализ масштабируемости баз данных и их приложений на этапе проектирования. В сборнике: Системный анализ в проектировании и управлении. Сборник научных трудов XXI Международной научно-практической конференции: в 2-х томах. 2017. С. 287-292.
4. Токмаков Г.П., Савкин А.Л. Функциональная связь, реквизиты и показатели как средства разработки семантически корректных схем баз данных. Автоматизация процессов управления. 2020. № 3 (61). С. 39-49.
5. Шлендер Н.С., Бугакова Т.Ю., Сирин-оол В.О. Методика разработки API для работы с базой данных электронной информационно-образовательной среды СГУГиТ. Интерэкспо Гео-Сибирь. 2021. Т. 7. № 2. С. 221-225.
6. Костычев Е.А., Омельченко В.А., Зеленов С.В. Нацеленная генерация данных для тестирования приложений над базами данных. Труды Института системного программирования РАН. 2011. Т. 20. С. 253-268.
7. Ноек И.Д., Балашов М.С. Разработка мобильного приложения с использованием локального хранилища. Символ науки: международный научный журнал. 2022. № 7-1. С. 5-8.
8. Толкачева Е.В. Система автоматизации формирования технологической документации на основе модифицированного метода поиска ассоциативных правил. диссертация на соискание ученой степени кандидата технических наук / Сиб. автомобил.-дорож. акад. (СибАДИ). Омск, 2012

ЛИСТИНГ ПРОГРАММ

Листинг 1 – Добавление зависимости Realm в файл `build.gradle`

```
dependencies {
    implementation 'io.realm:realm-android-library:10.8.0'
}
```

Листинг 2 – Синхронизация проекта с `Gradle` и инициализация базы данных Realm в классе приложения

```
import android.app.Application;
import io.realm.Realm;
import io.realm.RealmConfiguration;

public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        Realm.init(this);
                            RealmConfiguration    config    =    new
RealmConfiguration.Builder().build();
        Realm.setDefaultConfiguration(config);
    }
}
```

Листинг 3 – Класс `Location`

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
```

```
import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class Location extends RealmObject {

    @PrimaryKey
    @NotNull
    private String locationName;
    private String description;
    @NotNull
    private RealmList<GeoCoords> geoCoords;

         public  Location(String  locationName,  @Nullable  String
description) {
        this.locationName = locationName;
        this.description = description;
        geoCoords = new RealmList<>();
    }

    @NotNull
    public String getLocationName() {
        return locationName;
    }
    public void setLocationName(String locationName) {
        this.locationName = locationName;
    }

    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
```

```
    @NotNull
    public RealmList<GeoCoords> getGeoCoords() {
        return geoCoords;
    }
    public void addGeoCoords(GeoCoords geoCoords) {
        this.geoCoords.add(geoCoords);
    }
    public void removeGeoCoords(GeoCoords geoCoords) {
        this.geoCoords.remove(geoCoords);
    }

    public void destroy() {
        deleteFromRealm(this);
    }
}
```

Листинг 4 – Класс Device

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class Device extends RealmObject {

    @PrimaryKey
    @NotNull
    private String deviceName;
```

```
    private String type;
    private String description;
    private RealmList<FreqRange> freqRanges;

        public Device(String deviceName, @Nullable String type,
@Nullable String description) {
        this.deviceName = deviceName;
        this.type = type;
        this.description = description;
        freqRanges = new RealmList<>();
    }

    @NotNull
    public String getDeviceName() {
        return deviceName;
    }
    public void setDeviceName(@NotNull String deviceName) {
        this.deviceName = deviceName;
    }

    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }

    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
```

```
    public RealmList<FreqRange> getFreqRanges() {
        return freqRanges;
    }
    public void addFreqRange(FreqRange freqRange) {
        if (!freqRanges.contains(freqRange)) {
            freqRanges.add(freqRange);
            freqRange.addDevice(this);
        }
    }
    public void removeFreqRange(FreqRange freqRange) {
        if (freqRanges.contains(freqRange)) {
            freqRanges.remove(freqRange);
            freqRange.removeDevice(this);
        }
    }

    public void destroy() {
        deleteFromRealm(this);
    }
}
```

Листинг 5 – Класс FreqRange

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class FreqRange extends RealmObject {
```

```
    @PrimaryKey
    @NotNull
    private Integer freqRangeId;
    private Float freqBegin;
    private Float freqEnd;
    private String description;
    private RealmList<FreqRecord> freqRecords;
    private RealmList<Device> devices;

    private void calculateId() {
        freqRangeId = freqBegin.hashCode() ^ freqEnd.hashCode();
    }

      public FreqRange(Float freqBegin, Float freqEnd, @Nullable
String description) {
        this.freqBegin = freqBegin;
        this.freqEnd = freqEnd;
        this.description = description;
        calculateId();
        freqRecords = new RealmList<>();
        devices = new RealmList<>();
    }

    public Float getFreqBegin() {
        return freqBegin;
    }
    public void setFreqBegin(Float freqBegin) {
        this.freqBegin = freqBegin;
        calculateId();
    }

    public Float getFreqEnd() {
        return freqEnd;
```

```
}
public void setFreqEnd(Float freqEnd) {
    this.freqEnd = freqEnd;
}

public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
    calculateId();
}

public RealmList<FreqRecord> getFreqRecords() {
    return freqRecords;
}
public void addFreqRecord(FreqRecord freqRecord) {
    if (!freqRecords.contains(freqRecord)) {
        freqRecords.add(freqRecord);
        freqRecord.addFreqRange(this);
    }
}
public void removeFreqRecord(FreqRecord freqRecord) {
    if (freqRecords.contains(freqRecord)) {
        freqRecords.remove(freqRecord);
        freqRecord.removeFreqRange(this);
    }
}

public RealmList<Device> getDevices() {
    return devices;
}
public void addDevice(Device device) {
    if (!devices.contains(device)) {
```

```
            devices.add(device);
            device.addFreqRange(this);
        }
    }
    public void removeDevice(Device device) {
        if (devices.contains(device)) {
            devices.remove(device);
            device.removeFreqRange(this);
        }
    }

    public void destroy() {
        deleteFromRealm(this);
    }
}
```

Листинг 6 – Класс GeoCoords

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;

import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class GeoCoords extends RealmObject {

    @PrimaryKey
    @NotNull
    private Integer geoCoordsId;
    private Float HemisphereOfLatitude;
    private Float DegreesOfLatitude;
```

```
    private Float HemisphereOfLongitude;
    private Float DegreesOfLongitude;
    @NotNull
    private RealmList<Location> locations;
    @NotNull
    private RealmList<Sample> samples;

    private void calculateId() {
                geoCoordsId = HemisphereOfLatitude.hashCode() ^
DegreesOfLatitude.hashCode() ^ HemisphereOfLongitude.hashCode() ^
DegreesOfLongitude.hashCode();
    }

          public GeoCoords(Float HemisphereOfLatitude, Float
DegreesOfLatitude, Float HemisphereOfLongitude, Float
DegreesOfLongitude) {
        this.HemisphereOfLatitude = HemisphereOfLatitude;
        this.DegreesOfLatitude = DegreesOfLatitude;
        this.HemisphereOfLongitude = HemisphereOfLongitude;
        this.DegreesOfLongitude = DegreesOfLongitude;
        calculateId();
        locations = new RealmList<>();
        samples = new RealmList<>();
    }

                public void setHemisphereOfLatitude(Float
hemisphereOfLatitude) {
        HemisphereOfLatitude = hemisphereOfLatitude;
        calculateId();
    }
    public Float getHemisphereOfLatitude() {
        return HemisphereOfLatitude;
    }
```

```
    public void setDegreesOfLatitude(Float degreesOfLatitude) {
        DegreesOfLatitude = degreesOfLatitude;
        calculateId();
    }
    public Float getDegreesOfLatitude() {
        return DegreesOfLatitude;
    }

                public    void    setHemisphereOfLongitude(Float
hemisphereOfLongitude) {
        HemisphereOfLongitude = hemisphereOfLongitude;
        calculateId();
    }
    public Float getHemisphereOfLongitude() {
        return HemisphereOfLongitude;
    }

    public void setDegreesOfLongitude(Float degreesOfLongitude) {
        DegreesOfLongitude = degreesOfLongitude;
        calculateId();
    }
    public Float getDegreesOfLongitude() {
        return DegreesOfLongitude;
    }

    @NotNull
    public RealmList<Location> getLocations() {
        return locations;
    }
    public void addLocation(Location location) {
        if (!locations.contains(location)) {
            locations.add(location);
            location.addGeoCoords(this);
        }
```

```
    }
    public void removeLocation(Location location) {
        if (locations.contains(location)) {
            locations.remove(location);
            location.removeGeoCoords(this);
        }
    }

    @NotNull
    public RealmList<Sample> getSamples() {
        return samples;
    }
    public void addSample(Sample sample) {
        if (!samples.contains(sample)) {
            samples.add(sample);
            sample.setGeoCoords(this);
        }
    }
    public void removeSample(Sample sample) {
        if (!samples.contains(sample)) {
            samples.add(sample);
            sample.destroy();
        }
    }

    public void destroy() {
        for (Sample sample : samples) {
            sample.destroy();
        }
        deleteFromRealm(this);
    }
}
```

Листинг 7 – Класс Sample

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;

import java.util.Date;

import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class Sample extends RealmObject {

    @PrimaryKey
    @NotNull
    private Integer sampleId;
    private GeoCoords geoCoords;
    private Date recordTime;
    @NotNull
    private RealmList<FreqRecord> freqRecords;

    private void calculateId() {
        sampleId = geoCoords.hashCode() ^ recordTime.hashCode();
    }

    public Sample(GeoCoords geoCoords, Date recordTime) {
        this.geoCoords = geoCoords;
        this.recordTime = recordTime;
        calculateId();
        freqRecords = new RealmList<>();
        geoCoords.addSample(this);
    }
```

```
public GeoCoords getGeoCoords() {
    return geoCoords;
}
public void setGeoCoords(GeoCoords geoCoords) {
    this.geoCoords = geoCoords;
}

public Date getRecordTime() {
    return recordTime;
}
public void setRecordTime(Date recordTime) {
    this.recordTime = recordTime;
}

@NotNull
public RealmList<FreqRecord> getFreqRecords() {
    return freqRecords;
}
public void addFreqRecord(FreqRecord freqRecord) {
    if (!freqRecords.contains(freqRecord)) {
        freqRecords.add(freqRecord);
        freqRecord.setSample(this);
    }
}
public void removeFreqRecord(FreqRecord freqRecord) {
    if (freqRecords.contains(freqRecord)) {
        freqRecords.remove(freqRecord);
        freqRecord.destroy();
    }
}

public void destroy() {
    for (FreqRecord freqRecord : freqRecords) {
```

```
            freqRecord.destroy();
        }
        deleteFromRealm(this);
    }
}
```

Листинг 8 – Класс FrequencyRecord

```
package com.example.hackrfcommunicator.realm;

import org.jetbrains.annotations.NotNull;

import io.realm.RealmList;
import io.realm.RealmObject;
import io.realm.annotations.PrimaryKey;

public class FreqRecord extends RealmObject {

    @PrimaryKey
    @NotNull
    private Integer freqRecordId;
    private Sample sample;
    private Integer freqValue;
    @NotNull
    private Float signalStrength;
    @NotNull
    private RealmList<FreqRange> freqRanges;

    private void caluclateId() {
        freqRecordId = sample.hashCode() ^ freqValue.hashCode();
    }
```

```
    public FreqRecord(Sample sample, Integer freqValue, Float
signalStrength) {
        this.sample = sample;
        this.freqValue = freqValue;
        caluclateId();
        this.signalStrength = signalStrength;
        freqRanges = new RealmList<>();
        sample.addFreqRecord(this);
    }

    public Sample getSample() {
        return sample;
    }
    public void setSample(Sample sample) {
        this.sample = sample;
        caluclateId();
    }

    public Integer getFreqValue() {
        return freqValue;
    }
    public void setFreqValue(Integer freqValue) {
        this.freqValue = freqValue;
        caluclateId();
    }

    @NotNull
    public Float getSignalStrength() {
        return signalStrength;
    }
    public void setSignalStrength(@NotNull Float signalStrength) {
        this.signalStrength = signalStrength;
    }
```

```
    @NotNull
    public RealmList<FreqRange> getFreqRanges() {
        return freqRanges;
    }
    public void addFreqRange(FreqRange freqRange) {
        if (!freqRanges.contains(freqRange)) {
            freqRanges.add(freqRange);
            freqRange.addFreqRecord(this);
        }
    }
    public void removeFreqRange(FreqRange freqRange) {
        if (freqRanges.contains(freqRange)) {
            freqRanges.remove(freqRange);
            freqRange.removeFreqRecord(this);
        }
    }

    public void destroy() {
        deleteFromRealm(this);
    }
}
```

Листинг 9 – Конструкция для создания нового объекта Location с заданными параметрами

```
Realm realm = Realm.getDefaultInstance();
realm.executeTransaction(r -> {
        Location location = r.createObject(Location.class, "New
Location");
    location.setDescription("This is a new location");
});
realm.close();
```

Листинг 10 – Способ получения всех объектов Device с типом Station

```
RealmResults<Device> devices = realm.where(Device.class)
    .equalTo("type", "Station")
    .findAll();
```

Листинг 11 – Функция имитации поведения программно-определяемой радиосистемы, записывающая данные в базу данных

```
    private void generateData() {
        Sample sample = realm.createObject(Sample.class, -1);

sample.setGeoCoords(realm.where(GeoCoords.class).findFirst());
        sample.setRecordTime(LocalDateTime.now());

        Random r = new Random();
        float time = 107.0f;
        for (float i = 104.8f; i < 107.7f; i+=.1f) {
            Integer x = (int)(i * 1_000_000);
            Float y;
            if (Math.abs(time - i) < .1 * 2) {
                y = -20 + r.nextFloat() * 5;
            } else {
                y = -45 + r.nextFloat() * 5;
            }
                                        FreqRecord freqRecord =
realm.createObject(FreqRecord.class, -1);
            freqRecord.setSample(sample);
            freqRecord.setFreqValue(x);
            freqRecord.setSignalStrength(y);
        }
    }
```

Листинг 12 – Метод, считывающий последний обработанный программно-определяемой радиосистемой пакет, записанный в базу данных

```
    private ArrayList<BarEntry> getLastSampleFromBD() {
                                        GeoCoords geoCoords =
realm.where(GeoCoords.class).findFirst();
        Sample sample = realm
                .where(Sample.class)
                .equalTo("geoCoords", geoCoords.toString())
                .equalTo("recordTime",
                        realm
                                .where(Sample.class)
                                        .equalTo("geoCoords",
geoCoords.toString())
                                .max("recordTime")
                                .intValue()).findFirst();
        ArrayList<BarEntry> data = new ArrayList<>();

        for (FreqRecord freqRecord : sample.getFreqRecords()) {
                  data.add(new  BarEntry(freqRecord.getFreqValue(),
freqRecord.getSignalStrength()));
        }

        return data;
    }
```

Основные сущности системы:

7. локация (location):
    a. location name: название локации;
    b. description: описание локации.
8. экземпляры приборов (device):
    a. inventory number: инвентарный номер;
    b. location name: название локации;
    c. device name: название прибора;
    d. device type: тип прибора;
    e. hemisphere of latitude: полушарие широты (N или S);
    f. degrees of latitude: градусы широты;
    g. hemisphere of longitude: полушарие долготы (E или W);
    h. degrees of longitude: градусы долготы;
    i. installation date: дата установки.
9. параметры окружающей среды (environmental parameters):
    a. parameter name: название параметра;
    b. inventory number: инвентарный номер;
    c. temperature: температура;
    d. humidity: влажность;
    e. wind speed: скорость ветра;
    f. co2 level: уровень co2;
    g. time: время.
10. проанализированная информация (analyzed information):
    a. number of analysis: номер анализа;
    b. parameter name: название параметра;

c. inventory number: инвентарный номер;
d. time of analysis: время анализа;
e. fire hazard: вероятность возгорания.

11. оповещения (alarm):
    a. number of alarm: номер оповещения;
    b. number of analysis: номер анализа;
    c. status: статус;
    d. date: дата;
    e. time: время.
12. инцидент (incident):
    a. number of incident: номер инцидента;
    b. number of alarm: номер оповещения;
    c. status: статус;
    d. description: описание;
    e. time of detection: время обнаружения;
    f. time of resolve: время разрешения инцидента.