|  |  |
| --- | --- |
|  | **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 Теоретическая информатика и технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ**

**ПО КУРСУ БАЗЫ ДАННЫХ**

**НА ТЕМУ:**

«Проектирование базы данных музыкальной платформы»

Студент

*подпись, дата фамилия, и.о.*

Научный руководитель

*подпись, дата фамилия, и.о.*

*2022 г*

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc107093999)

[1. Обзор предметной области 4](#_Toc107094000)

[1.1. Формирование требований к приложению 4](#_Toc107094001)

[1.2. ER-модель 4](#_Toc107094002)

[1.2.1. Описание сущностей 6](#_Toc107094003)

[1.2.2. Обоснование выбора кардинальных связей 8](#_Toc107094004)

[2. Проектирование клиент-серверного приложения 10](#_Toc107094005)

[2.1. Реляционная модель 10](#_Toc107094006)

[2.1.1. Свойства отношений 12](#_Toc107094007)

[2.1.2. Ограничения минимальной кардинальности 16](#_Toc107094008)

[2.2. Клиент-серверное взаимодействие 18](#_Toc107094009)

[2.3. Протокол клиент-серверного приложения 20](#_Toc107094010)

[3. Реализация 22](#_Toc107094011)

[CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc107094012)

[ПРИЛОЖЕНИЯ 24](#_Toc107094013)

[ПРИЛОЖЕНИЕ А 24](#_Toc107094014)

# ВВЕДЕНИЕ

С развитием Интернета и компьютерных технологий становилось возможным упрощать жизнь их пользователей. Например, раньше, чтобы узнать о какой-либо интересной человеку музыкальной группе, приходилось искать и покупать журналы, в которых возможно и не было нужной информации. Чтобы послушать музыку, приходилось покупать кассеты и винил, храня их у себя дома. При этом они могли занимать много места у особых меломанов.

В начале 2000-х все чаще музыка стала распространяться в цифровом формате. Были разные варианты: скачивание после покупки в онлайн-магазине или через торрент. Теперь музыкальные композиции можно было хранить на компьютере, что было довольно удобно, так как экономилось место в доме и можно было прослушивать музыку без Интернета. Однако возникала проблема с синхронизацией песен, если предполагалось прослушивание с разных устройств. Развитие процессоров и технологий Интернета сделали возможным так называемую потоковую передачу (стриминг) аудио.

На сегодняшний день стриминговые сервисы обладают наибольшей популярностью среди разных вариантов прослушивания песен. Заходя в приложение с разных устройств, пользователь получает доступ к одной и той же библиотеке песен с возможными рекомендациями. Зачастую в таких сервисах также присутствует информация о группах, музыкантах, концертах и т.д. По причине хранения всей нужной информации в одном месте они и обрели популярность.

Целью данной курсовой работы является разработка базы данных для музыкальной платформы, предполагающей потоковую передачу аудио и хранение информации о группах, музыкантах и концертах.

# Обзор предметной области

## Формирование требований к приложению

В качестве предметной области была выбрана музыкальная индустрия. База данных должна хранить информацию о музыкантах, историю участия музыкантов в группе и учитывать то, что музыкант мог играть в ней несколько раз. Модель данных должна содержать информацию о существующих и уже распавшихся группах. Помимо этого, для каждой группы должно быть возможным посмотреть прошедшие и запланированные концерты. Альбомы должны содержать информацию о песнях в них. При просмотре альбома песни должны быть отсортированы в исходном, заданном создателями порядке.

Предполагается, что пользователь из приложения сможет просматривать различную информацию о группах, альбомах, песнях и музыкантах. Также параллельно пользователь сможет прослушивать музыку. Помимо этого представляется возможным поиск музыкантов, групп, концертов, песен по имени/названию.

Добавлять в базу данных различные сущности должен администратор, для этого нужно реализовать отдельное приложение. В основном, оно будет предназначено для добавления песен и альбомов, так как просто SQL-запросами делать это неудобно (нужно добавлять все песни альбома в базу данных, проще это автоматизировать). Помимо этого, для удобства данная утилита не должна зависеть от определенного формата добавляемых аудио-файлов.

## ER-модель

На основе сформированных требований была создана модель «сущность-связь», представленная на рисунке 1.1, в нотации «Crow’s Foot» Дж. Мартина.

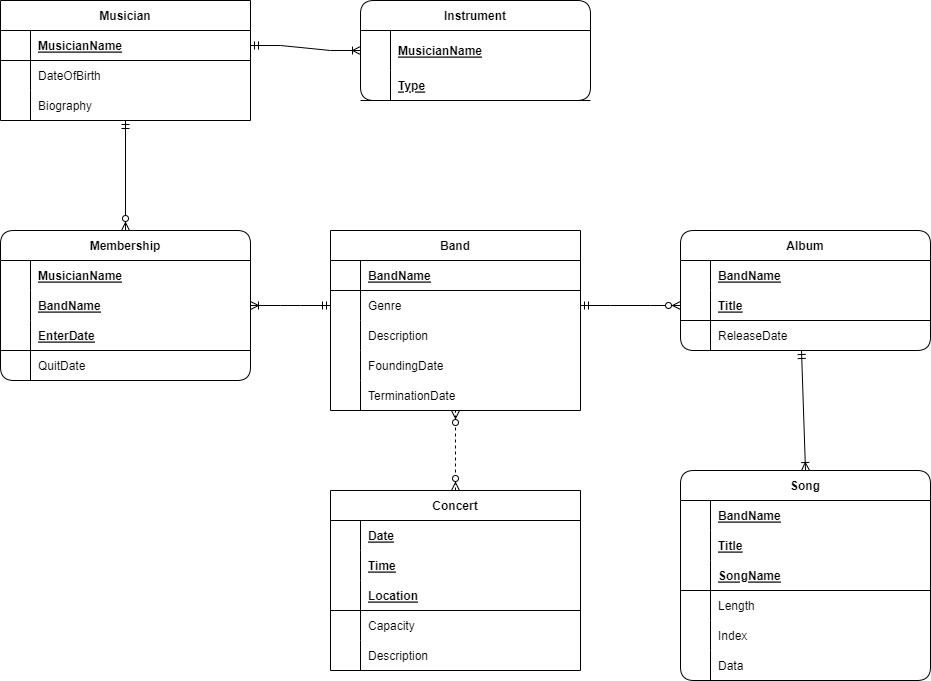


Рисунок 1.1 - ER-модель

В результате получилось 7 сущностей, 3 сильные и 4 слабые. К сильным сущностям относятся:

1. Musician (музыкант)
2. Band (группа)
3. Concert (концерт)

К слабым относятся:

1. Instrument (инструмент)
2. Membership (членство в группе)
3. Album (альбом)
4. Song (песня в альбоме)

## Описание сущностей

Далее описывается каждая из сущностей ER-модели.

1. Musician – сущность, описывающая отдельного музыканта.

Идентификатор:

MusicianName: имя музыканта

Атрибуты:

DateOfBirth – день рождения

Biography – биография

1. Instrument – многозначный атрибут у Musician, обозначающий инструменты, на которых играет музыкант.

Составной идентификатор:

MusicianName – имя соответствующего музыканта

Type – тип инструмента (гитара, вокал и т.д.)

1. Membership – сущность, хранящая членство музыканта в группе в определенный период времени. Используется шаблон «сопряжение».

Составной идентификатор:

MusicianName – имя музыканта

BandName – имя группы

EnterDate – дата вступления в группу

Атрибуты:

QuitDate – дата, когда музыкант покинул группу

Стоит отметить, что наличие в составном идентификаторе атрибута EnterDate обеспечивает случай, когда музыкант присоединялся к группе несколько раз.

1. Band – сущность, обозначающая группу.

Идентификатор:

BandName – имя группы

Атрибуты:

Genre – преобладающий жанр музыки у группы

Description – описание группы

FoundingDate – дата основания группы

TerminationDate – дата распада группы

1. Concert – сущность, обозначающая концерт.

Составной идентификатор:

Date – дата проведения

Time – время проведения

Location – место проведения

Атрибуты:

Capacity – вместимость

Description – описание концерта

В данном случае место проведения концерта является и его названием, так как зачастую они совпадают.

1. Album – сущность, обозначающая альбом, является многозначным атрибутом у группы.

Составной идентификатор:

BandName – имя группы, выпустившей альбом

Title – название альбома

Атрибуты:

ReleaseDate – дата выпуска альбома

1. Song – сущность, обозначающая песню, является многозначным атрибутом у альбома.

Составной идентификатор:

BandName – название группы

Title – название альбома

SongName – название песни

Атрибуты:

Length – длительность песни

Index – номер песни в альбоме, начиная с 1

Data – блок бинарных данных песни

## Обоснование выбора кардинальных связей

Далее рассматриваются кардинальные связи между сущностями ER-модели.

1. Связь 1:N, M-M, идентифицирующая Musician – Instrument

Musician: музыкант может владеть несколькими инструментами, но при этому хотя бы одним, иначе он не музыкант.

Instrument: у инструмента музыкант ровно один по шаблону многозначный атрибут.

1. Связь 1:N, M-O, идентифицирующая Musician – Membership

Musician: музыкант мог участвовать во многих группах или не участвовать ни в одной.

Membership: по шаблону «сопряжение» сущность связана ровно с одной родительской сущностью.

1. Связь 1:N, M-M, идентифицирующая Band – Membership

Band: в группе должен быть хотя бы один музыкант, при этом их может быть несколько.

Membership: аналогично предыдущему случаю, по шаблону «сопряжение» ровно одна родительская сущность.

1. Связь N:M, O-O, неидентифицирующая Band-Concert

Band: группа могла выступать (будет выступать) как нескольких концертах, так и ни на одном.

Concert: концерт может быть запланирован, но не иметь расписания, поэтому на нем может не играть ни одной группы в текущий момент. Также понятно, что на концерте может играть больше одной группы.

1. Связь 1:N, M-O, идентифицирующая Band-Album

Band: группа может иметь как ноль альбомов, так и несколько.

Album: у альбома по шаблону «многозначный атрибут» ровно одна группа.

1. Связь 1:N, M-M, идентифицирующая Album-Song

Album: в альбоме может быть много песен, но как минимум одна.

Song: песня принадлежит какому-либо альбому, поэтому он должен быть ровно один.

# Проектирование клиент-серверного приложения

Основной задачей данного этапа является разработка архитектуры клиент-серверного приложения, реализующего просмотр клиентом информации о разных сущностях базы данных и потоковую передачу аудио-файлов. Все приложение можно разделить на 4 части:

1. Клиент, реализующий обмен данными с сервером
2. Сервер, взаимодействующий с клиентом и выполняющий запросы в базу данных
3. Приложение для администратора, позволяющее добавлять альбомы в базу данных, а также некоторые другие сущности
4. База данных, представляющая собой совокупность таблиц, представлений и триггеров

Прежде всего, нужно определить, как сущности будут храниться в базе данных. Для этого необходимо создать реляционную модель базы данных путем преобразования ER-модели.

## Реляционная модель

Реляционная модель, получившаяся из ER-модели, изображена на рисунке 2.1.

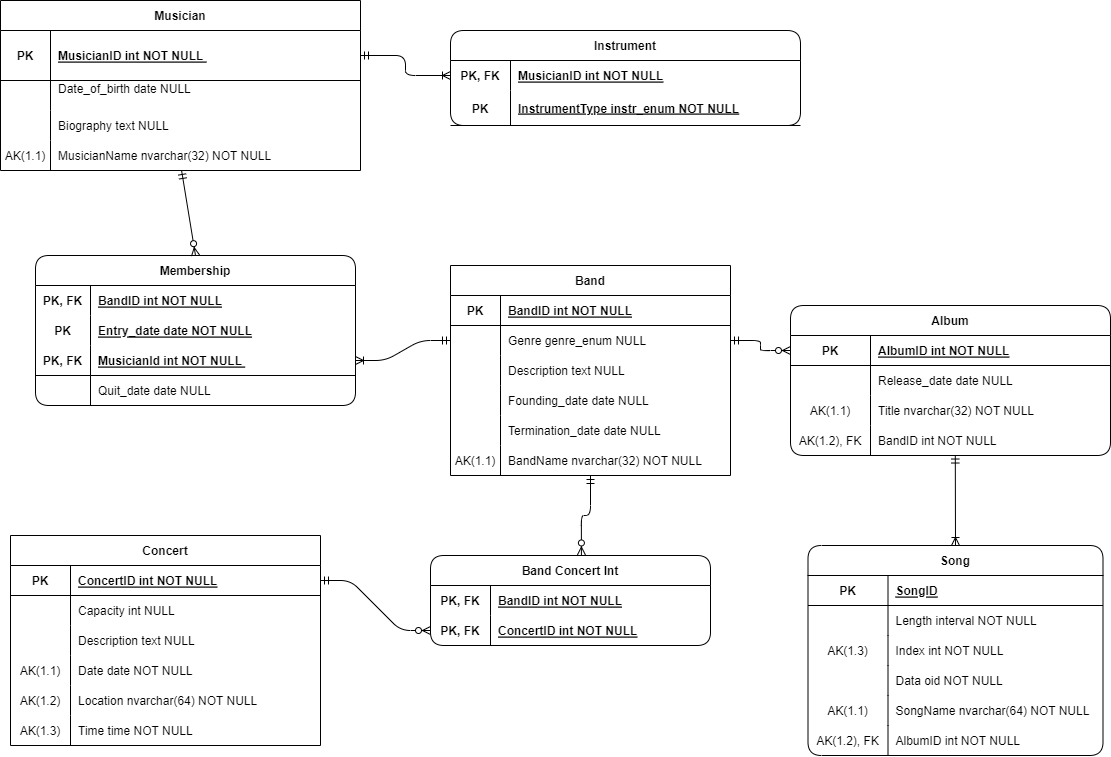
После преобразования, помимо таблиц, соответствующих сущностям ER-модели, добавилась еще одна промежуточная таблица для связи «многие-ко-многим»: Band Concert Int, связывающая группу и концерт. Также в угоду производительности, где возможно, у отношений в качестве первичного ключа или его части (при составном ключе) выступает целочисленное значение. Аналогично и с внешними ключами. Выигрыш будет при осуществлении поиска записей. Существенный прирост производительности будет, например, в случае, если нужно узнать на каких концертах играют группы. Для этого необходимо сделать двойное присоединение таблиц Band, Band Concert Int и Concert. При этом идентификатор сущности преобразовался в альтернативный ключ в соответствующих таблицах.

Рисунок 2.1 - Реляционная модель

Для полей, значение которых может принадлежать только определенному множеству значений, задан тип Enum (перечисление) для поддержания целостности базы данных. К ним относятся поля жанра музыки у группы и тип инструмента музыканта.

Поскольку подразумевается потоковая передача аудио-файлов, для их хранения нужно выбрать тип данных, позволяющий считывать блок данных из определенного места. То есть должны поддерживаться аналоги функций read, seek, tell стандартной библиотеки языка Си для чтения файлов. В разных СУБД такие типы данных называются по-разному, поэтому на данном этапе под типом данных поля Data отношения Song понимается некий абстрактный тип BLOB бинарных данных.

При получении запроса аудио-файла от клиента сервер начинает с определенной периодичностью (между другими запросами) посылать в ответ блоки данных одинакового размера. Перед отправкой сервер каждый раз запрашивает новый блок из базы данных с определенной позиции, а не хранит целый аудио-файл и не отправляет его частями. Таким образом, это решение является оптимальным по памяти, а значит использование описанного типа данных BLOB обосновано.

Также возможен вариант хранения файлов на диске, а не в базе данных непосредственно. При этом в базе данных хранятся названия файлов, соответствующих записям. Плюсом данного подхода является меньший размер базы данных и, соответственно, более быстрое резервное копирование. Однако из этого вытекают минусы – отсутствие атомарности операций над базой данных и целостности. Также более трудно переносить базу данных с одной машины на другую. Помимо этого файлы уязвимы к действиям администратора. Исходя из этих минусов, был выбран первый вариант хранения аудио-файлов непосредственно в БД.

### Свойства отношений

В таблицах 2.1 – 2.8 представлена информация об отношениях в базе данных: типы полей, типы ключей, допустимость неопределенных значений и дополнительная информация.

Таблица 2.1 – Отношение Musician

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| MusicianID | int | PK | Нет | Суррогатный ключ |
| DateOfBirth | date | Нет | Да | Дата без времени |
| Biography | text | Нет | Да |  |
| MusicianName | nvarchar(32) | AK | Нет | Уникальный |

Таблица 2.2 – Отношение Instrument

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| MusicianID | int | PK, FK | Нет | Суррогатный ключ |
| InstrumentType | instr\_enum | PK | Нет |  |

Таблица 2.3 – Отношение Membership

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| MusicianID | int | PK, FK | Нет |  |
| BandID | int | PK, FK | Нет |  |
| EntryDate | date | PK | Нет | Дата без времени |
| QuitDate | date | Нет | Да | Дата без времени |

Таблица 2.4 – Отношение Band

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| BandID | int | PK | Нет | Суррогатный ключ |
| Genre | genre\_enum | Нет | Да | Тип «жанр» |
| Description | text | Нет | Да |  |
| FoundingDate | date | Нет | Да | Дата без времени |
| TerminationDate | date | Нет | Да | Дата без времени |
| BandName | nvarchar(32) | AK | Нет | Уникальный |

Таблица 2.5 – Отношение Band Concert Int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| BandID | int | PK, FK | Нет |  |
| ConcertID | int | PK, FK | Нет |  |

Таблица 2.6 – Отношение Concert

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| ConcertID | int | PK | Нет | Суррогатный ключ |
| Capacity | int | Нет | Да |  |
| Description | text | Нет | Да |  |
| Date | date | AK(1.1) | Нет | Уникальный (составной ключ), дата без времени |
| Location | nvarchar(64) | AK(1.2) | Нет | Уникальный, (составной ключ) |
| Time | time | AK(1.3) | Нет | Уникальный, (составной ключ), время |

Таблица 2.7 – Отношение Album

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| AlbumID | int | PK | Нет | Суррогатный ключ |
| BandID | int | FK, AK(1.1) | Нет | Уникальный (составной ключ) |
| ReleaseDate | date | Нет | Да | Дата без времени |
| Title | nvarchar(32) | AK(1.2) | Нет | Уникальный (составной ключ) |

Таблица 2.8 – Отношение Song

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название столбца | Тип | Ключ | Неопр. значение | Дополнительно |
| SongID | int | PK | Нет | Суррогатный ключ |
| Length | interval | Нет | Нет | Интервал времени в секундах |
| Data | blob | Нет | Нет | Бинарные данные песни |
| SongName | nvarchar(64) | AK(1.1) | Нет | Уникальный (составной ключ) |
| AlbumID | int | FK, AK(1.2) | Нет | Уникальный (составной ключ) |
| Index | int | AK(1.3) | Нет | Уникальный (составной ключ) |

### Ограничения минимальной кардинальности

В таблицах 2.9 – 2.15 представлены действия, нужные для выполнения ограничений минимальной кардинальности связей между отношениями.

Таблица 2.9 - Связь Musician-Instrument

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Musician | Действие над Instrument |
| Вставка | Одновременная вставка хотя бы одной записи в дочернюю таблицу | Ограничение ссылочной целостности + MusicianID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | Если единственный инструмент, то запрет |

Таблица 2.10 - Связь Musician-Membership

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Musician | Действие над Membership |
| Вставка | - | Ограничение ссылочной целостности + MusicianID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | - |

Таблица 2.11 - Связь Band-Membership

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Band | Действие над Membership |
| Вставка | Одновременная вставка хотя бы одной записи в дочернюю таблицу | Ограничение ссылочной целостности + BandID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | Если единственный Membership, то запрет |

Таблица 2.12 - Связь Band-Album

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Band | Действие над Album |
| Вставка | - | Ограничение ссылочной целостности + BandID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | - |

Таблица 2.13 – Связь Album-Song

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Album | Действие над Song |
| Вставка | Одновременная вставка хотя бы одной записи в дочернюю таблицу | Ограничение ссылочной целостности + AlbumID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | Запрет |

Здесь стоит отметить, что удаление записи в таблице Song возможно только при удалении всего альбома, так как множество песен на альбоме неразрывно связано с ним.

Таблица 2.14 – Связь Concert - BandConcertInt

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Concert | Действие над BandConcertInt |
| Вставка | - | Ограничение ссылочной целостности + ConcertID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | - |

Таблица 2.15 – Связь Band - BandConcertInt

|  |  |  |
| --- | --- | --- |
| Операция | Действие над Band | Действие над BandConcertInt |
| Вставка | - | Ограничение ссылочной целостности + BandID NOT NULL (внешний ключ) |
| Изменение ключа (первичного/внешнего) | Запрет | Запрет |
| Удаление | Каскадное удаление | - |

## Клиент-серверное взаимодействие

Клиент и сервер взаимодействуют между собой посредством сети. Как было указано в требованиях к приложению в главе 1, клиент должен иметь возможность одновременно прослушивать музыку и осуществлять навигацию по приложению. Это требование накладывает ограничение на запросы к серверу – они должны быть асинхронными. В ином случае пришлось бы ждать выполнение текущего запроса для выполнения следующего. Если это запрос на передачу поточных данных, то ожидание увеличивается в разы по сравнению, например, с простым запросом таблицы музыкантов. Даже вынесение сетевого взаимодействия в отдельный поток не помогло бы решить проблему. Таким образом, приложение нужно проектировать исходя из того, что оно должно реализовывать асинхронные запросы к серверу.

Реализовать его можно путем создания идентификатора запроса, который передается на сервер, и включается в его ответ. В таком случае на стороне клиента нужно хранить отображение идентификатора в некоторые данные, которые необходимы для обработки ответа.

Также предполагается, что у клиента будет возможность просмотреть список записей некоторых сущностей базы данных в виде таблицы и сделать по ним поиск. Например, запросить список песен. Понятно, что загружать все песни из базы данных за один запрос вряд ли нужно. Таким образом, в протоколе взаимодействия нужно предусмотреть запросы ограниченного числа записей. В предположении, что записи отсортированы, это можно реализовать путем добавления в запрос полей offset (смещение), size (максимальное число записей в ответе), filter (строка, по которой осуществляется поиск). Такому набору параметров будет соответствовать запрос: «select \* from table where col ilike <filter> order by col limit <size> offset <offset>».

Для снижения нагрузки с сервера нужно предусмотреть отмену поточной передачи данных. Это можно сделать, послав на сервер запрос отмены с нужным идентификатором. Такой случай возможен, когда клиент, не дослушав текущую песню, захочет запросить другую.

Так как клиент ожидает быстрого отклика и отсутствия задержки при проигрывании песни, передача аудио-файла должна быть как можно более эффективной. Это можно реализовать комбинацией двух приемов: использование в качестве нижележащего протокола TCP для реализации собственного протокола уровня приложения стека TCP/IP [1] и сжатием исходных файлов. Таким образом, в базе данных должны храниться песни в сжатом формате. На стороне клиента песня должна декодироваться в «сырой формат» PCM [2], состоящий из сэмплов звука определенной частоты дискретизации и который воспринимается аудиокартой. Также нужно сделать декодирование по запросу, например, когда звука для передачи в аудиокарту осталось менее чем на 100 мс. Это позволит избежать накопления большого буфера данных.

Также использование сжатых данных уменьшит размер базы данных.

## Протокол клиент-серверного приложения

Как было сказано ранее, для реализации взаимодействия между клиентом и сервером стоит использовать протокол TCP. Так как протокол будет бинарным для улучшения производительности, необходимо придумать способ сериализации запросов для записи данных в TCP соединение. Одним из способов является использование протокола Google Protobuf (Protocol Buffers).

Protobuf представляет собой описание структур, по которому генерируется код для их сериализации и десериализации [3]. Этот протокол является независимым от платформы и языка программирования. Это позволяет реализовать сервер и клиент на разных ЯП. Также Protobuf отличается быстрым разбором пакета и оптимизацией его размера (поле не занимает места, если для него нет данных).

Недостатком данного протокола является необходимость знать точный размер сообщения для его декодирования. Решить эту проблему можно путем добавления 4 байт (размер unsigned int) перед началом пакета – неотрицательное число в little-endian, которое будет обозначать размер бинарных данных в кодировке Protobuf. Таким образом, парсер сообщений сети у клиента и сервера должен работать в двух состояниях: первое – состояние считывания размера пакета, второе – считывание самого пакета и его десериализация.

Для создания сообщения необходимо определить имя следующего вида: «message <имя типа>», и внутри тела описать поля. Также можно сделать тип перечисление с помощью следующей команды: «enum <имя типа>».

Протокол приложения будет иметь 3 типа сообщений: запрос таблицы, запрос потоковой передачи аудио-файла и простой запрос какой-либо сущности. Первый запрос, как исходит из названия, выполняется, когда пользователь захочет увидеть список записей одной таблицы, например, список всех песен в базе данных. Второй – при запросе песни. Третий – при запросе какой-либо страницы, например, страницы группы. Аналогично, определено три типа ответа – на каждый тип запроса.

У всех запросов и ответов есть общие поля – номер запроса и факт отмены запроса (логическое значение). Protobuf позволяет добавлять вложенные сообщения, поэтому можно определить структуры Request и Answer с этими двумя полями и хранить в них уже конкретный запрос. Кроме этого, ключевое слово «oneof» сообщает компилятору Protobuf, что в поле хранится только одно из перечисленных значений. Под этот случай как раз подходит описанные ранее 3 типа сообщений.

Пример сообщения Request изображен на листинге 1.

message Request {

uint64 reqId = 1;

bool cancel = 3; //send again with cancel = true to cancel request

oneof msg {

TableReq tableReq = 4;

SimpleReq simpleReq = 5;

StreamReq streamReq = 6;

}

}

Листинг 1 – сообщение Request в Protobuf

Полный список сообщений протокола находится в приложении А.

# Реализация

CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* + 1. <https://www.ibm.com/docs/ru/aix/7.2?topic=protocol-tcpip-protocols>
    2. <https://www.sony.ru/electronics/audio-vysokogo-razresheniya-ot-a-do-ja-lyubiteli-audiotekhnologij>
    3. <https://developers.google.com/protocol-buffers/docs/overview>

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Протокол клиент-серверного приложения в кодировке Protobuf

syntax = "proto3";

option go\_package = "./proto";

enum EntityType {

BAND = 0;

ALBUM = 1;

SONG = 2;

CONCERT = 3;

MUSICIAN = 4;

MEMBERSHIP = 5;

}

message Request {

uint64 reqId = 1;

bool cancel = 3; //send again with cancel = true to cancel request

oneof msg {

TableReq tableReq = 4;

SimpleReq simpleReq = 5;

StreamReq streamReq = 6;

}

}

message Answer {

uint64 reqId = 1;

bool cancel = 2; //failed req on server side

oneof msg {

TableAns tableAns = 3;

SimpleAns simpleAns = 4;

StreamAns streamAns = 5;

}

}

message TableReq {

uint32 first = 1;

uint32 last = 2;

string filter = 3;

EntityType type = 4;

}

message SimpleReq {

string reqString = 1;

EntityType type = 4;

}

message StreamReq {

string reqString = 1;

int32 suggestedSize = 2;

EntityType type = 3;

}

message TableAns {

repeated Band bands = 1;

repeated Album albums = 2;

repeated Concert concerts = 3;

repeated Song songs = 4;

EntityType type = 5;

}

message SimpleAns {

oneof msg {

Band band = 1;

Album album = 2;

Song song = 3;

Concert concert = 4;

Musician musician = 5;

Membership membership = 6;

}

}

message StreamAns {

bytes data = 1;

bool isFinal = 2;

}

enum Genre {

ROCK = 0;

ALTERNATIVE = 1;

INDIE = 2;

BLUES = 3;

METAL = 4;

}

message Membership {

string musName = 1;

string bandName = 2;

int64 unixEntryDate = 3;

int64 unixQuitDate = 4;

}

message Concert {

int32 capacity = 1;

string description = 2;

int64 unixDateTime = 3;

string location = 4;

repeated string bandNames = 5;

}

message Band {

string bandName = 1;

Genre genre = 2;

int64 unixFoundDate = 3;

int64 unixTermDate = 4;

string description = 5;

repeated string albumNames = 6;

repeated Membership participants = 7;

repeated Concert concerts = 8;

}

message Song {

string songName = 1;

int32 lengthSec = 2;

string albumName = 3;

string bandName = 4;

}

message Album {

string title = 1;

int64 unixReleaseDate = 2;

repeated Song songs = 3;

string bandName = 4;

}

message Musician {

string musName = 1;

int64 unixDateOfBirth = 2;

string bio = 3;

repeated Membership memberships = 4;

}