|  |  |
| --- | --- |
|  | **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](#_Toc107074789)

[1. Обзор предметной области 4](#_Toc107074790)

[1.1. Формирование требований к приложению 4](#_Toc107074791)

[1.2. ER-модель 4](#_Toc107074792)

[1.2.1. Описание сущностей 6](#_Toc107074793)

[1.2.2. Обоснование выбора кардинальных связей 8](#_Toc107074794)

[2. Проектирование клиент-серверного приложения 10](#_Toc107074795)

[2.1. Реляционная модель 10](#_Toc107074796)

[2.1.1. Свойства отношений 12](#_Toc107074797)

[2.1.2. Ограничения минимальной кардинальности 15](#_Toc107074798)

[CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc107074799)

[ПРИЛОЖЕНИЯ 18](#_Toc107074800)

[ПРИЛОЖЕНИЕ А 18](#_Toc107074801)

# 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.

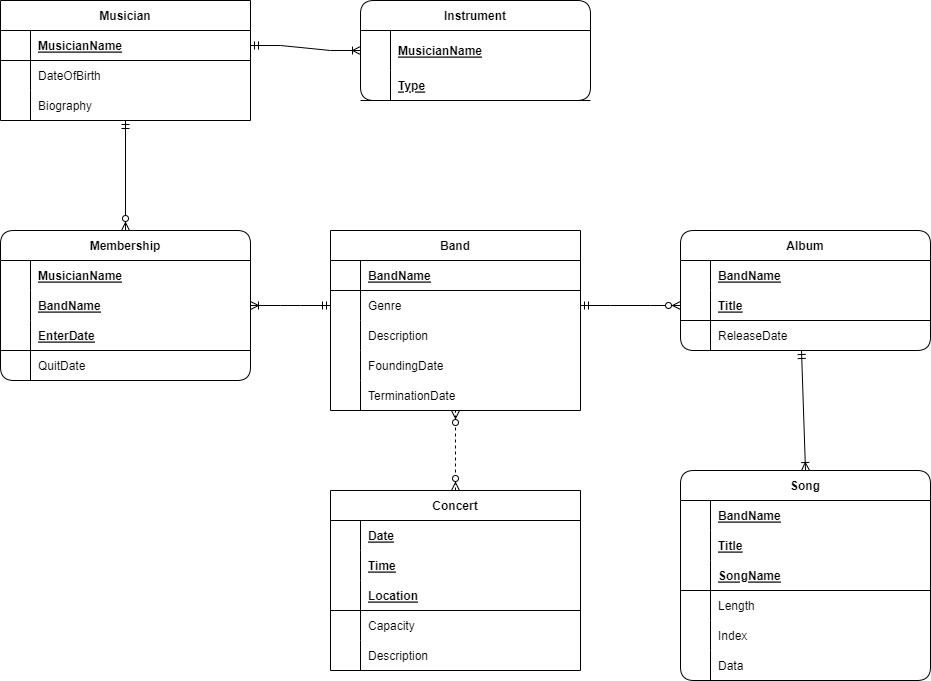


Рисунок 1.1 - ER-модель

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.

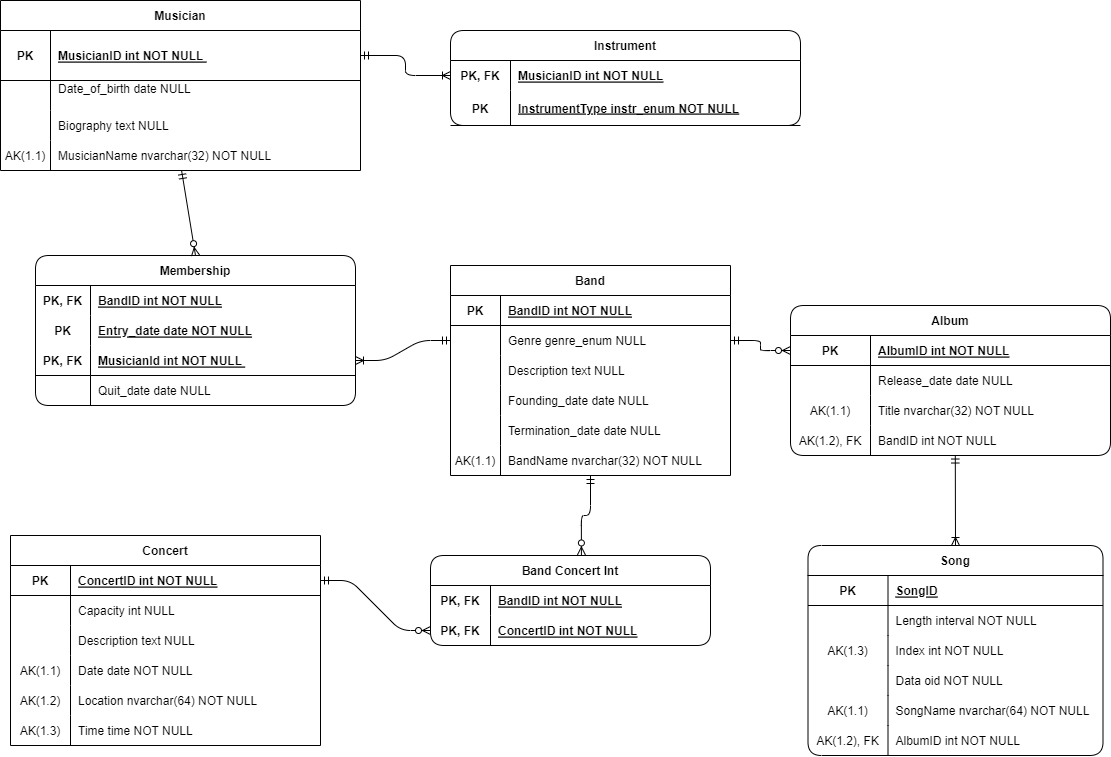
После преобразования, помимо таблиц, соответствующих сущностям ER-модели, добавилась еще одна промежуточная таблица для связи «многие-ко-многим»: Band Concert Int, связывающая группу и концерт. Также в угоду производительности, где возможно, у отношений в качестве первичного ключа или его части (при 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 обоснованно.

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 | - |

CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* + 1. Керриск М. Linux API. Исчерпывающее руководство. Питер, 2010.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Поддерживаемые коды управляющих последовательностей с их видом и описанием

#define ERROR 0

#define ERASE\_VISIBLE\_SCREEN 1 //CSI 2 J - erase whole buffer, CSI is ESC [

#define MOVE\_CURSOR\_HOME 2 //CSI H - move cursor home

#define MOVE\_CURSOR\_LINE\_COLUMN 3 //CSI <line> <column> H - move cursor

#define MOVE\_CURSOR\_LINE 4 //CSI <lines\_num> A, CSI <lines\_num> B - cursor UP/DOWN on amount <lines\_num>

#define MOVE\_CURSOR\_COLUMN 5 //CSI <cols\_num> C, CSI <cols\_num> D - cursor RIGHT/LEFT on amount <cols\_num>

#define MOVE\_CURSOR\_BEGIN\_NEXT\_LINE 6 //CSI <lines\_num> E - move <lines\_num> lines down and to the beginning of the line

#define MOVE\_CURSOR\_BEGIN\_PREV\_LINE 7 //CSI <lines\_num> F - move <lines\_num> lines up and to the beginning of the line

#define MOVE\_CURSOR\_POS\_COL 8 //CSI <col\_pos> G - move cursor to <col\_pos> column

#define REQ\_CURSOR 9 //CSI 6 n - request cursor

#define SAVE\_CURSOR 10 //CSI s - save cursor on stack

#define RESTORE\_CURSOR 11 //CSI u - pop from stack and set cursor position

#define ERASE\_CUR\_TO\_END 12 //CSI J - erase screen from current cursor position to the end

#define ERASE\_START\_TO\_CURSOR 13 //CSI 1 J - erase screen from start position to cursor

#define ERASE\_ENTIRE\_BUFFER 14 //CSI 3 J - erase scrollback area

#define CLEAR\_CURRENT\_LINE 15 //CSI 2 K - clear current line

#define CLEAR\_CUR\_TO\_END\_OF\_LINE 16 //CSI K - clear elements from cursor position to the end of line

#define CLEAR\_START\_TO\_CURSOR\_LINE 17 //CSI 1 K - clear elements from line start to cursor position

#define RESET\_STYLE 18 // CSI 0 m, CSI m - reset current SGR state

#define STYLE 19 //CSI <params> m - style structure is filled, <params> ::= <n> | <n> ; <params>

#define SET\_ICON\_WINDOW\_NAME 20 //OSC 0 ; txt \x07 - set txt as window and icon name of terminal emulator, OSC is ESC ]

#define SET\_ICON\_NAME 21 //OSC 1 ; txt \x07 - set txt as icon name

#define SET\_WINDOW\_NAME 22 //OSC 2 ; txt \x07 - set txt as window name

#define ERASE\_N\_CHARS\_FROM\_CURSOR 23 //CSI <n> X - erase <n> chars from cursor

#define ALT\_BUF\_ON 24 //CSI ? <47 | 1047 | 1049> h - set alt buf with specific params

#define ALT\_BUF\_OFF 25 //CSI ? <47 | 1047 | 1049> l - set main buf with specific params

#define HIDE\_CUR 26 //CSI ? 25 l - cursor is not visible

#define SHOW\_CUR 27 //CSI ? 25 h - cursor is visible

#define AUTO\_WRAP\_ON 28 //CSI ? 7 h - set auto wrap mode

#define AUTO\_WRAP\_OFF 29 //CSI ? 7 l - reset auto wrap mode

#define MOVE\_CUR\_ABS 30 //CSI <n> d - move cursor to absolute position in row <n>

#define DELETE\_N\_CHARS\_RIGHT\_FROM\_CURSOR\_WITH\_SHIFT 31 //CSI <n> P - delete <n> chars right from cursor with shift

#define ROW\_MARGINS 32 //CSI <top> ; <bottom> r - vertical margins set scroll area

#define INDEX 33 //CSI D - invoke index

#define REVERSE\_INDEX 34 //CSI M - invoke reversed index

#define SCROLL\_UP 35 //CSI <n> S - perform scroll up within scroll area on <n> rows

#define SCROLL\_DOWN 36 //CSI <n> T - perform scroll down on <n> rows

#define INSERT\_LINE 37 //CSI <n> L - insert <n> lines on current cursor row position with shift

#define DELETE\_LINE 38 //CSI <n> M - delete <n> lines from current cursor row position with shift

#define NOT\_SUPPORTED 100