МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**



Институт интеллектуальных кибернетических систем

**Кафедра кибернетики (№ 22)**

**Отчёт о работе по курсу**

**«Базы данных (теоретические основы баз данных)»**

Вариант «Яндекс.Музыка»

|  |  |
| --- | --- |
| Выполнил | Миронов А.М. |
| Группа | Б20-524 |
| Вариант | Яндекс.Музыка |
| Преподаватель | Миркасымов Р.Б. |
| Проверяющий |  |
| Оценка |  |

**Москва 2022**

**Содержание**

1. [Формулировка задания 3](#_bookmark0)
2. [Концептуальная модель базы данных 3](#_bookmark1)
   1. [Конкретизация предметной области 4](#_bookmark2)
   2. [Описание предметной области и принятых ограничений 4](#_bookmark3)
   3. [Описание атрибутов 5](#_bookmark4)
3. [Логическое проектирование 6](#_bookmark5)
4. [Физическое проектирование 7](#_bookmark6)
   1. [Создание таблиц 7](#_bookmark7)
   2. [Заполнение базы данных 14](#_bookmark8)
      1. [Подготовка данных 14](#_bookmark9)
      2. [Программа заполнения базы данных 15](#_bookmark10)
      3. [Результаты заполнения 17](#_bookmark11)
5. [Выполнение запросов 21](#_bookmark12)

# Формулировка задания

#### Спроектировать базу данных для аудио-платформы «Яндекс.Музыка». Каталог Яндекс.Музыки — это база данных, содержащая информацию о музыкальном контенте: треках, исполнителях, альбомах, жанрах и лейблах.

# Концептуальная модель базы данных

#### После проведения анализа предметной области была спроектирована следующая концептуальная модель:

#### https://sun9-10.userapi.com/impg/2xJmC_In0My1n3H-XQprhpn-mLRdPFndTYm1kg/rxPpcF1VYE4.jpg?size=1062x793&quality=96&sign=ff0ca1a77894601ca5c79a83b983e5ca&type=album

#### Рисунок 1.1 – Концептуальная модель базы данных, часть 1.

#### 

#### Рисунок 1.2 – Концептуальная модель базы данных, часть 2.

## Конкретизация предметной области

#### Необходимо было создать базу данных, хранящую данные о треках, альбомах, исполнителях, подкастах и их взаимосвязях. Также должны учитываться действия пользователей по отношению к трекам, альбомам, подкастам. Например, прослушивание, лайки, блокировка.

* 1. **Описание предметной области**

#### Система ориентирована на следующих пользователей:

#### Обычные пользователи

#### Исполнители

#### Любой пользователь может прослушать трек, лайкнуть его или заблокировать, чтобы он больше ему не попадался. Также он может создать плейлист, прослушать и лайкнуть другие плейлисты.

#### Плейлисты могут содержать различные треки. К каждому треку присваивается тэг, отражающий категорию трека. Исполнители могут обладать треками, альбомами. Треки могут входить в альбомы. У каждого трека, как и у артиста также есть жанр. Исполнитель при желании может оставить ссылки на свои соц сети/другие ссылки. Пользователи могут лайкать понравившихся исполнителей, как и альбомы.

#### Также в системе есть категория подкастов. Каждый пользователь может прослушать/лайкнуть любой подкаст. Подкасты могут образовывать альбомы, а также пользователи могут сами создавать плейлисты из существующих подкастов. Юзеры также могут лайкать альбомы и плейлисты подкастов.

#### Таким образом, были выделены следующие сущности:

#### Пользователь

#### Исполнитель

#### Трек

#### Тэг

#### Жанр

#### Ссылка

#### Плейлист

#### Альбом

#### Подкаст

#### Альбом подкастов

#### Плейлист подкастов

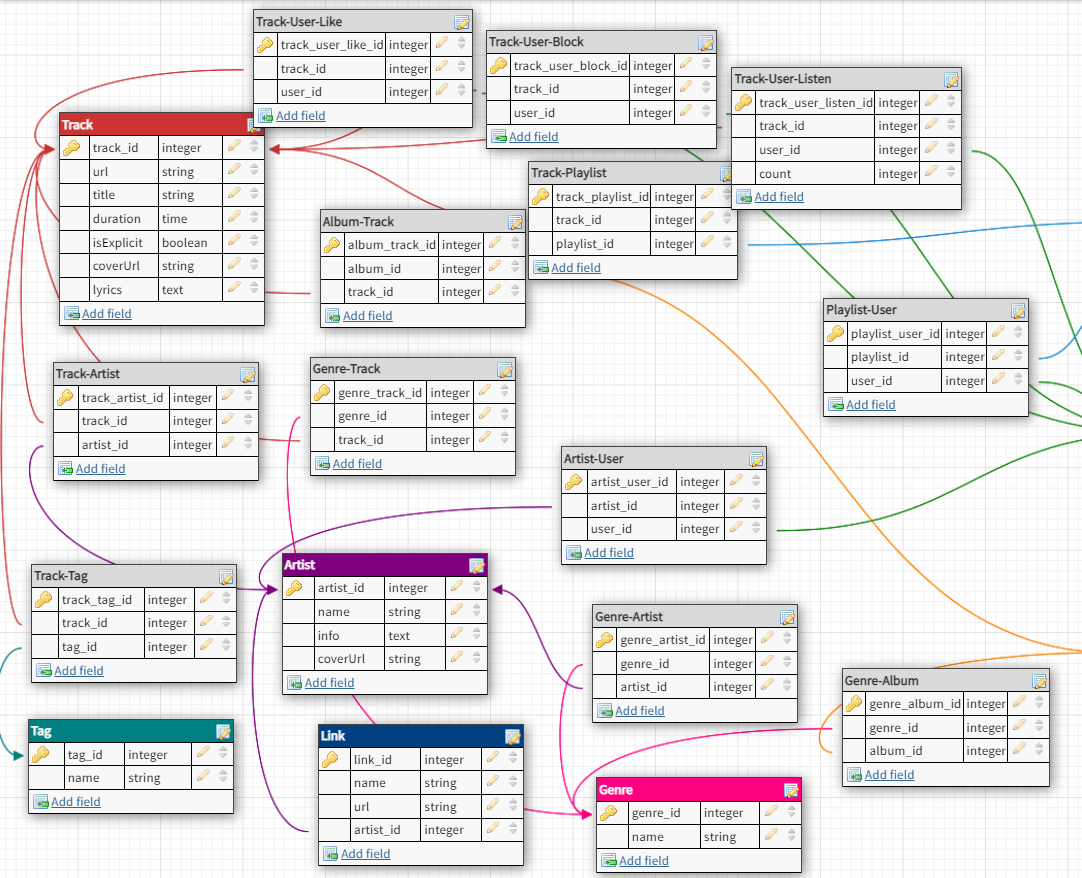
## Описание атрибутов

#### В процессе анализа были выделены следующие атрибуты, название и описание которых приведены в таблице ниже.

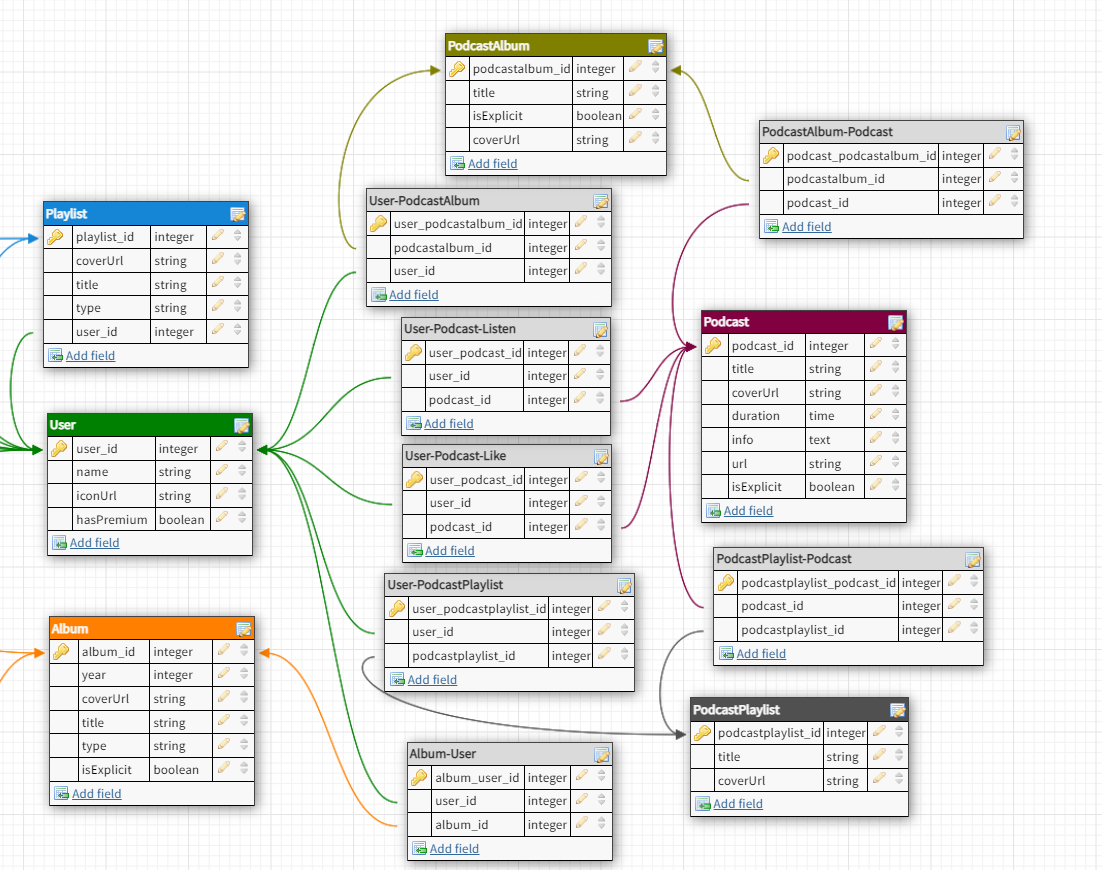
|  |  |
| --- | --- |
| Имя атрибута | Расшифровка |
| album\_id, artist\_id, track\_id, user\_id, ... | Уникальный идентификатор соответственно альбома, артиста, трека, юзера и т.д. |
| year | Год выпуска альбома |
| coverUrl | Ссылка на обложку трека/подкаста и т.п. |
| album.title, playlist.title, podcast.title, podcastalbum.title, podcastplaylist.title | Название альбома/плейлиста/подкаста/альбома подкастов/плейлиста подкастов |
| album.type, playlist.type | Тип альбома/плейлиста |
| isExplicit | Есть ли нецензурные слова |
| author.name | Имя автора |
| info | Информация об исполнителе/подкасте |
| coverUrl | Ссылка на фото/обложку |
| genre.name | Название жанра |
| link.name | Название соц сети или сайта |
| url | Ссылка на соц сеть или сайт/подкаст/трек |
| duration | Длительность подкаста/трека |
| tag.name | Имя тэга – информации для рекомендаций |
| lyrics | Текст трека |
| count | Число прослушиваний одним пользователем одного трека |
| user.name | Имя пользователя |
| hasPremium | Показывает, есть ли у пользователя премиум |
| iconUrl | Иконка пользователя |

# Логическое проектирование

#### Следующим шагом на основе КМПО была разработана логическая модель базы данных, представленная ниже:



#### Рисунок 2.1 – Логическая модель базы данных, часть 1.



#### Рисунок 2.2 – Логическая модель базы данных, часть 2.

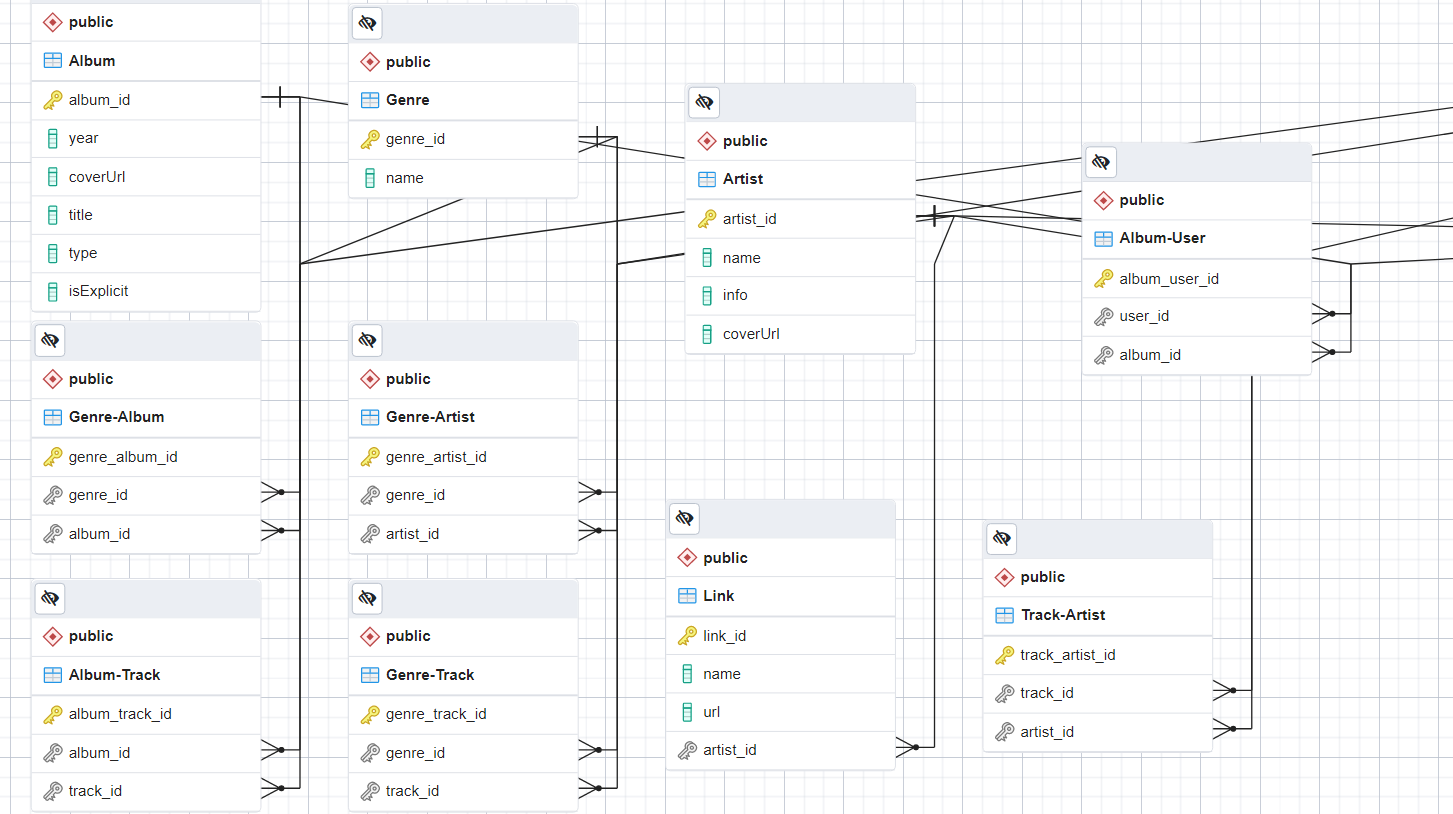
#### Все связи типа «Многие ко многим» были реализованы через отдельные таблицы.

#### Многозначные атрибуты Tag, Genre были выделены в отдельные сущности, чтобы привести базу данных к 1 нормальной форме.

#### Таким образом, после этого все таблицы базы данных находятся в 3НФ форме, т.к. все атрибуты являются простыми, отсутствуют функциональные зависимости между неключевыми атрибутами и первичными ключами, а также нет транзитивных зависимостей.

# Физическое проектирование

#### В качестве СУБД для реализации разработанной базы данных была выбрана PostgreSQL. В связи с проведённым анализом предметной области и была проработана следующая физическая схема базы данных. Она представлена на следующем рисунке:



#### Рисунок 3 – Графическое представление базы данных.

## Создание таблиц

#### Ниже приведены скриптовый код на языке SQL (диалект PostgreSQL) для создания таблиц для таблиц, описанных выше.

#### Код для создания таблицы Track

CREATE TABLE "Track" (

"track\_id" serial,

"url" VARCHAR(255),

"title" VARCHAR(255) NOT NULL,

"duration" TIME NOT NULL,

"isExplicit" BOOLEAN NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

"lyrics" TEXT NOT NULL,

CONSTRAINT "Track\_pk" PRIMARY KEY ("track\_id")

);

#### Код для создания таблицы Album

CREATE TABLE "Album" (

"album\_id" serial,

"year" integer,

"coverUrl" VARCHAR(255),

"title" VARCHAR(255) NOT NULL,

"type" VARCHAR(255) NOT NULL,

"isExplicit" BOOLEAN NOT NULL,

CONSTRAINT "Album\_pk" PRIMARY KEY ("album\_id")

);

#### Код для создания таблицы User

CREATE TABLE "User" (

"user\_id" serial,

"name" VARCHAR(255),

"iconUrl" VARCHAR(255) NOT NULL,

"hasPremium" BOOLEAN NOT NULL,

CONSTRAINT "User\_pk" PRIMARY KEY ("user\_id")

);

#### Код для создания таблицы Playlist

CREATE TABLE "Playlist" (

"playlist\_id" serial NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

"title" VARCHAR(255) NOT NULL,

"type" VARCHAR(255) NOT NULL,

"user\_id" integer NOT NULL,

CONSTRAINT "Playlist\_pk" PRIMARY KEY ("playlist\_id")

);

#### Код для создания таблицы Tag

CREATE TABLE "Tag" (

"tag\_id" serial NOT NULL,

"name" VARCHAR(255) NOT NULL,

CONSTRAINT "Tag\_pk" PRIMARY KEY ("tag\_id")

);

#### Код для создания таблицы Artist

CREATE TABLE "Artist" (

"artist\_id" serial NOT NULL,

"name" VARCHAR(255) NOT NULL,

"info" serial NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

CONSTRAINT "Artist\_pk" PRIMARY KEY ("artist\_id")

);

#### Код для создания таблицы Genre

CREATE TABLE "Genre" (

"genre\_id" serial NOT NULL,

"name" VARCHAR(255) NOT NULL,

CONSTRAINT "Genre\_pk" PRIMARY KEY ("genre\_id")

);

#### Код для создания таблицы Link

CREATE TABLE "Link" (

"link\_id" serial NOT NULL,

"name" VARCHAR(255) NOT NULL,

"url" VARCHAR(255) NOT NULL,

"artist\_id" serial NOT NULL,

CONSTRAINT "Link\_pk" PRIMARY KEY ("link\_id")

);

#### Код для создания таблицы PodcastAlbum

CREATE TABLE "PodcastAlbum" (

"podcastalbum\_id" serial NOT NULL,

"title" VARCHAR(255) NOT NULL,

"isExplicit" serial NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

CONSTRAINT "PodcastAlbum\_pk" PRIMARY KEY ("podcastalbum\_id")

);

#### Код для создания таблицы Podcast

CREATE TABLE "Podcast" (

"podcast\_id" serial NOT NULL,

"title" VARCHAR(255) NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

"duration" serial NOT NULL,

"info" serial NOT NULL,

"url" VARCHAR(255) NOT NULL,

"isExplicit" serial NOT NULL,

CONSTRAINT "Podcast\_pk" PRIMARY KEY ("podcast\_id")

);

#### Код для создания таблицы PodcastPlaylist

CREATE TABLE "PodcastPlaylist" (

"podcastplaylist\_id" serial NOT NULL,

"title" VARCHAR(255) NOT NULL,

"coverUrl" VARCHAR(255) NOT NULL,

CONSTRAINT "PodcastPlaylist\_pk" PRIMARY KEY ("podcastplaylist\_id")

);

Промежуточные таблцы для связи основных

#### Код для создания таблицы Album-Track

CREATE TABLE "Album-Track" (

"album\_track\_id" serial NOT NULL,

"album\_id" serial NOT NULL,

"track\_id" serial NOT NULL,

CONSTRAINT "Album-Track\_pk" PRIMARY KEY ("album\_track\_id")

);

#### Код для создания таблицы Album-User

CREATE TABLE "Album-User" (

"album\_user\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

"album\_id" serial NOT NULL,

CONSTRAINT "Album-User\_pk" PRIMARY KEY ("album\_user\_id")

);

#### Код для создания таблицы Genre-Track

CREATE TABLE "Genre-Track" (

"genre\_track\_id" serial NOT NULL,

"genre\_id" serial NOT NULL,

"track\_id" serial NOT NULL,

CONSTRAINT "Genre-Track\_pk" PRIMARY KEY ("genre\_track\_id")

);

#### Код для создания таблицы Genre-Album

CREATE TABLE "Genre-Album" (

"genre\_album\_id" serial NOT NULL,

"genre\_id" serial NOT NULL,

"album\_id" serial NOT NULL,

CONSTRAINT "Genre-Album\_pk" PRIMARY KEY ("genre\_album\_id")

);

#### Код для создания таблицы Track-Artist

CREATE TABLE "Track-Artist" (

"track\_artist\_id" serial NOT NULL,

"track\_id" serial NOT NULL,

"artist\_id" serial NOT NULL,

CONSTRAINT "Track-Artist\_pk" PRIMARY KEY ("track\_artist\_id")

);

#### Код для создания таблицы Track-User-Block

CREATE TABLE "Track-User-Block" (

"track\_user\_block\_id" serial NOT NULL,

"track\_id" integer NOT NULL,

"user\_id" integer NOT NULL,

CONSTRAINT "Track-User-Block\_pk" PRIMARY KEY ("track\_user\_block\_id")

);

#### Код для создания таблицы Track-Tag

CREATE TABLE "Track-Tag" (

"track\_tag\_id" serial NOT NULL,

"track\_id" serial NOT NULL,

"tag\_id" serial NOT NULL,

CONSTRAINT "Track-Tag\_pk" PRIMARY KEY ("track\_tag\_id")

);

#### Код для создания таблицы Track-User-Like

CREATE TABLE "Track-User-Like" (

"track\_user\_like\_id" serial NOT NULL,

"track\_id" integer NOT NULL,

"user\_id" integer NOT NULL,

CONSTRAINT "Track-User-Like\_pk" PRIMARY KEY ("track\_user\_like\_id")

);

#### Код для создания таблицы Track-User-Listen

CREATE TABLE "Track-User-Listen" (

"track\_user\_listen\_id" serial NOT NULL,

"track\_id" integer NOT NULL,

"user\_id" integer NOT NULL,

"count" integer NOT NULL,

CONSTRAINT "Track-User-Listen\_pk" PRIMARY KEY ("track\_user\_listen\_id")

);

#### Код для создания таблицы Track-Playlist

CREATE TABLE "Track-Playlist" (

"track\_playlist\_id" serial NOT NULL,

"track\_id" serial NOT NULL,

"playlist\_id" serial NOT NULL,

CONSTRAINT "Track-Playlist\_pk" PRIMARY KEY ("track\_playlist\_id")

);

#### Код для создания таблицы Playlist-User

CREATE TABLE "Playlist-User" (

"playlist\_user\_id" serial NOT NULL,

"playlist\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

CONSTRAINT "Playlist-User\_pk" PRIMARY KEY ("playlist\_user\_id")

);

#### Код для создания таблицы Artist-User

CREATE TABLE "Artist-User" (

"artist\_user\_id" serial NOT NULL,

"artist\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

CONSTRAINT "Artist-User\_pk" PRIMARY KEY ("artist\_user\_id")

);

#### Код для создания таблицы Genre-Artist

CREATE TABLE "Genre-Artist" (

"genre\_artist\_id" serial NOT NULL,

"genre\_id" serial NOT NULL,

"artist\_id" serial NOT NULL,

CONSTRAINT "Genre-Artist\_pk" PRIMARY KEY ("genre\_artist\_id")

);

#### Код для создания таблицы User-Podcast-Listen

CREATE TABLE "User-Podcast-Listen" (

"user\_podcast\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

"podcast\_id" serial NOT NULL,

CONSTRAINT "User-Podcast-Listen\_pk" PRIMARY KEY ("user\_podcast\_id")

);

#### Код для создания таблицы User-PodcastAlbum

CREATE TABLE "User-PodcastAlbum" (

"user\_podcastalbum\_id" serial NOT NULL,

"podcastalbum\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

CONSTRAINT "User-PodcastAlbum\_pk" PRIMARY KEY ("user\_podcastalbum\_id")

);

#### Код для создания таблицы User-PodcastPlaylist

CREATE TABLE "User-PodcastPlaylist" (

"user\_podcastplaylist\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

"podcastplaylist\_id" serial NOT NULL,

CONSTRAINT "User-PodcastPlaylist\_pk" PRIMARY KEY ("user\_podcastplaylist\_id")

);

#### Код для создания таблицы PodcastAlbum-Podcast

CREATE TABLE "PodcastAlbum-Podcast" (

"podcast\_podcastalbum\_id" serial NOT NULL,

"podcastalbum\_id" serial NOT NULL,

"podcast\_id" serial NOT NULL,

CONSTRAINT "PodcastAlbum-Podcast\_pk" PRIMARY KEY ("podcast\_podcastalbum\_id")

);

#### Код для создания таблицы PodcastPlaylist-Podcast

CREATE TABLE "PodcastPlaylist-Podcast" (

"podcastplaylist\_podcast\_id" serial NOT NULL,

"podcast\_id" serial NOT NULL,

"podcastplaylist\_id" serial NOT NULL,

CONSTRAINT "PodcastPlaylist-Podcast\_pk" PRIMARY KEY ("podcastplaylist\_podcast\_id")

);

#### Код для создания таблицы User-Podcast-Like

CREATE TABLE "User-Podcast-Like" (

"user\_podcast\_id" serial NOT NULL,

"user\_id" serial NOT NULL,

"podcast\_id" serial NOT NULL,

CONSTRAINT "User-Podcast-Like\_pk" PRIMARY KEY ("user\_podcast\_id")

);

#### Код для соединения всех таблиц:

ALTER TABLE "Playlist" ADD CONSTRAINT "Playlist\_fk0" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Link" ADD CONSTRAINT "Link\_fk0" FOREIGN KEY ("artist\_id") REFERENCES "Artist"("artist\_id");

ALTER TABLE "Album-Track" ADD CONSTRAINT "Album-Track\_fk0" FOREIGN KEY ("album\_id") REFERENCES "Album"("album\_id");

ALTER TABLE "Album-Track" ADD CONSTRAINT "Album-Track\_fk1" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Album-User" ADD CONSTRAINT "Album-User\_fk0" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Album-User" ADD CONSTRAINT "Album-User\_fk1" FOREIGN KEY ("album\_id") REFERENCES "Album"("album\_id");

ALTER TABLE "Genre-Track" ADD CONSTRAINT "Genre-Track\_fk0" FOREIGN KEY ("genre\_id") REFERENCES "Genre"("genre\_id");

ALTER TABLE "Genre-Track" ADD CONSTRAINT "Genre-Track\_fk1" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Genre-Album" ADD CONSTRAINT "Genre-Album\_fk0" FOREIGN KEY ("genre\_id") REFERENCES "Genre"("genre\_id");

ALTER TABLE "Genre-Album" ADD CONSTRAINT "Genre-Album\_fk1" FOREIGN KEY ("album\_id") REFERENCES "Album"("album\_id");

ALTER TABLE "Track-Artist" ADD CONSTRAINT "Track-Artist\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-Artist" ADD CONSTRAINT "Track-Artist\_fk1" FOREIGN KEY ("artist\_id") REFERENCES "Artist"("artist\_id");

ALTER TABLE "Track-User-Block" ADD CONSTRAINT "Track-User-Block\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-User-Block" ADD CONSTRAINT "Track-User-Block\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Track-Tag" ADD CONSTRAINT "Track-Tag\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-Tag" ADD CONSTRAINT "Track-Tag\_fk1" FOREIGN KEY ("tag\_id") REFERENCES "Tag"("tag\_id");

ALTER TABLE "Track-User-Like" ADD CONSTRAINT "Track-User-Like\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-User-Like" ADD CONSTRAINT "Track-User-Like\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Track-User-Listen" ADD CONSTRAINT "Track-User-Listen\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-User-Listen" ADD CONSTRAINT "Track-User-Listen\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Track-Playlist" ADD CONSTRAINT "Track-Playlist\_fk0" FOREIGN KEY ("track\_id") REFERENCES "Track"("track\_id");

ALTER TABLE "Track-Playlist" ADD CONSTRAINT "Track-Playlist\_fk1" FOREIGN KEY ("playlist\_id") REFERENCES "Playlist"("playlist\_id");

ALTER TABLE "Playlist-User" ADD CONSTRAINT "Playlist-User\_fk0" FOREIGN KEY ("playlist\_id") REFERENCES "Playlist"("playlist\_id");

ALTER TABLE "Playlist-User" ADD CONSTRAINT "Playlist-User\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Artist-User" ADD CONSTRAINT "Artist-User\_fk0" FOREIGN KEY ("artist\_id") REFERENCES "Artist"("artist\_id");

ALTER TABLE "Artist-User" ADD CONSTRAINT "Artist-User\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "Genre-Artist" ADD CONSTRAINT "Genre-Artist\_fk0" FOREIGN KEY ("genre\_id") REFERENCES "Genre"("genre\_id");

ALTER TABLE "Genre-Artist" ADD CONSTRAINT "Genre-Artist\_fk1" FOREIGN KEY ("artist\_id") REFERENCES "Artist"("artist\_id");

ALTER TABLE "User-Podcast-Listen" ADD CONSTRAINT "User-Podcast-Listen\_fk0" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "User-Podcast-Listen" ADD CONSTRAINT "User-Podcast-Listen\_fk1" FOREIGN KEY ("podcast\_id") REFERENCES "Podcast"("podcast\_id");

ALTER TABLE "User-PodcastAlbum" ADD CONSTRAINT "User-PodcastAlbum\_fk0" FOREIGN KEY ("podcastalbum\_id") REFERENCES "PodcastAlbum"("podcastalbum\_id");

ALTER TABLE "User-PodcastAlbum" ADD CONSTRAINT "User-PodcastAlbum\_fk1" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "User-PodcastPlaylist" ADD CONSTRAINT "User-PodcastPlaylist\_fk0" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "User-PodcastPlaylist" ADD CONSTRAINT "User-PodcastPlaylist\_fk1" FOREIGN KEY ("podcastplaylist\_id") REFERENCES "PodcastPlaylist"("podcastplaylist\_id");

ALTER TABLE "PodcastAlbum-Podcast" ADD CONSTRAINT "PodcastAlbum-Podcast\_fk0" FOREIGN KEY ("podcastalbum\_id") REFERENCES "PodcastAlbum"("podcastalbum\_id");

ALTER TABLE "PodcastAlbum-Podcast" ADD CONSTRAINT "PodcastAlbum-Podcast\_fk1" FOREIGN KEY ("podcast\_id") REFERENCES "Podcast"("podcast\_id");

ALTER TABLE "PodcastPlaylist-Podcast" ADD CONSTRAINT "PodcastPlaylist-Podcast\_fk0" FOREIGN KEY ("podcast\_id") REFERENCES "Podcast"("podcast\_id");

ALTER TABLE "PodcastPlaylist-Podcast" ADD CONSTRAINT "PodcastPlaylist-Podcast\_fk1" FOREIGN KEY ("podcastplaylist\_id") REFERENCES "PodcastPlaylist"("podcastplaylist\_id");

ALTER TABLE "User-Podcast-Like" ADD CONSTRAINT "User-Podcast-Like\_fk0" FOREIGN KEY ("user\_id") REFERENCES "User"("user\_id");

ALTER TABLE "User-Podcast-Like" ADD CONSTRAINT "User-Podcast-Like\_fk1" FOREIGN KEY ("podcast\_id") REFERENCES "Podcast"("podcast\_id");

## Заполнение базы данных

#### Заполнение базы данных производилось при помощи ЯП Python на основе данных, генерированных через библиотеку Faker. Подключение к самому postgreSQL осуществлялось с помощью библиотеки psycopg2.

### Подготовка к генерации данных

#### В сети Интернет была найдена документация к библиотеке Faker. На её основе изучены основные методы для генерации различных типов данных, которые также были проверены экспериментально. Изучена документация к библиотеке psycopg2, проверено подключение к базе данных.

### Программа заполнения базы данных

#### Выполнение запросов к БД осуществляется посредством выполнения запросов прямо из приложения PyCharm.

#### Импортирование необходимых библиотек:

import psycopg2

from psycopg2 import Error

from psycopg2.extensions import ISOLATION\_LEVEL\_AUTOCOMMIT

from faker import Faker

from faker.providers import DynamicProvider#, internet, misc, date\_time, lorem

import random

fake = Faker('ru-RU')

#### Подлючение к базе данных

*try:*

*connection = psycopg2.connect(user="postgres",*

*password="0000",*

*host="localhost",*

*port="5432",*

*database="Yandex\_music")*

*cursor = connection.cursor()*

...

except (Exception, Error) as error:

print("Ошибка при работе с PostgreSQL", error)

finally:

if connection:

cursor.close()

connection.close()

print("Соединение с PostgreSQL закрыто")

#### Функция заполнения таблицы Artist

def artist():

for \_ in range(1000):

name = fake.name()[:50]

info = fake.text(255)

coverurl = fake.domain\_name()[:255]

cursor.execute(f'''INSERT INTO "Artist" VALUES (DEFAULT, '{name}', '{info}', '{coverurl}')''')

#### Функция заполнения таблицы Album

def album():

for \_ in range(1000):

year = fake.year()

coverurl = fake.domain\_name()[:255]

title = fake.text(50)

type = fake.text(50)

isexpl = fake.boolean()

cursor.execute(f'''INSERT INTO "Album" VALUES (DEFAULT, '{year}', '{coverurl}', '{title}', '{type}', {isexpl})''')

#### Как видно, происходит обыкновенный INSERT запрос, который содержит случайно сгенерированные данные. После вызова каждой функции также выполняется проверка правильности заполнения данных:

cursor.execute('''SELECT \* FROM "Album"

LIMIT 3''')

print(cursor.fetchall())

#### Полный текст программы для заполнения таблиц приложены к отчёту — main.py

### Результаты заполнения

#### Далее представлены результаты работы программы на примере таблиц, соответствующих функциям, приведенным выше.

#### Таблица Album

#### 

#### Таблица Artist

#### 

#### Остальные таблицы:

#### Таблица Genre

#### 

#### Таблица Link

#### 

#### Таблица Playlist

#### 

#### Таблица Podcast

#### 

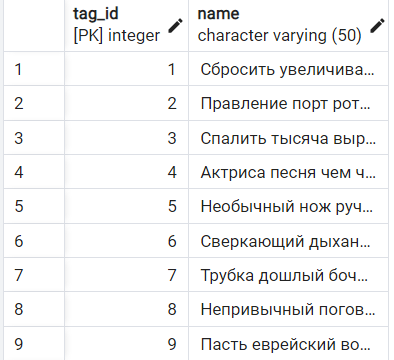
#### Таблица PodcastAlbum

#### 

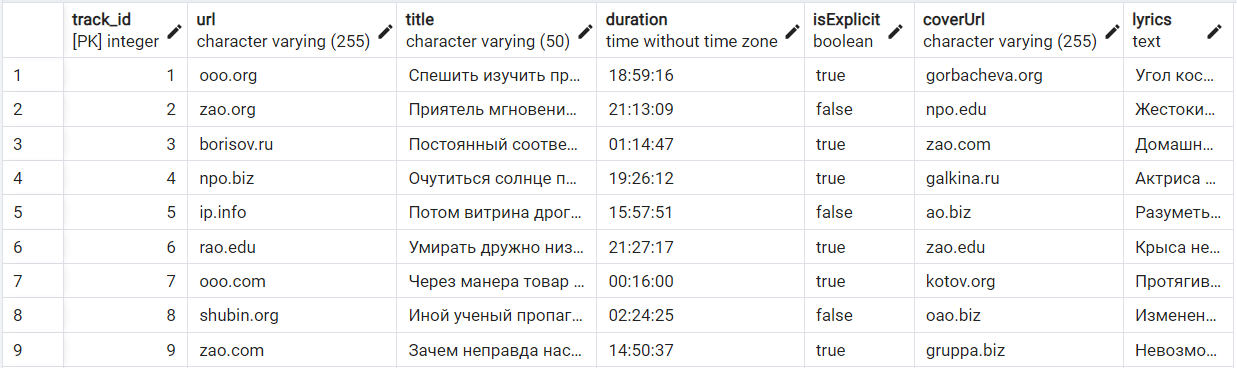
#### Таблица PodcastPlaylist



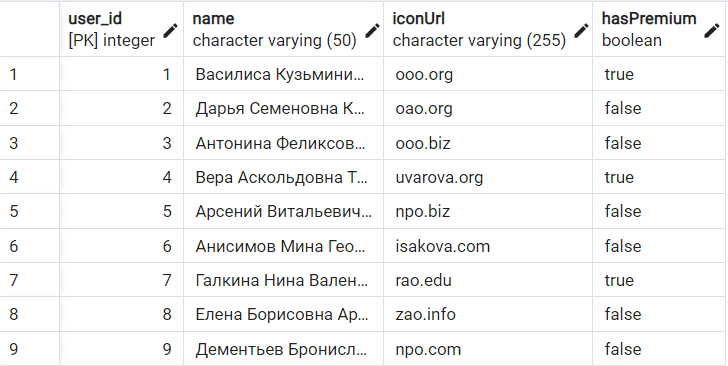
#### Таблица Tag



#### Таблица Track



#### Таблица User



# Выполнение запросов

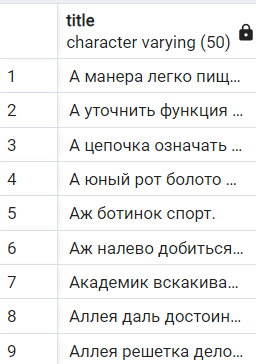
#### В этом разделе приведены различные запросы к реализованной базе данных — их краткие описания, непосредственно запрос на SQL языке и результат выполнения.

#### Вывести названия альбомов подкастов без пометки e ("isExplicit" = false) в отсортированном по алфавиту порядке.

SELECT title FROM "PodcastAlbum"

WHERE "isExplicit" = false

ORDER BY title



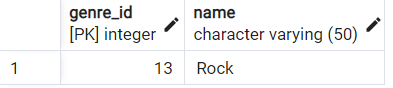
#### Изменить название 13-го жанра на “Rock” и вывести изменившуюся строчку.

UPDATE "Genre"

SET name = 'Rock'

WHERE genre\_id = 13

RETURNING \*



#### Вывести пользователей, их прослушанные треки и количество прослушиваний в порядке возрастания количества для 10 пользователей с наибольшим числом прослушиваний. Учитывать треки длительностью до часа.

SELECT \* FROM (

SELECT name, title, count

FROM "User"

INNER JOIN "Track-User-Listen" ON "Track-User-Listen".user\_id = "User".user\_id

INNER JOIN "Track" ON "Track".track\_id = "Track-User-Listen".track\_id

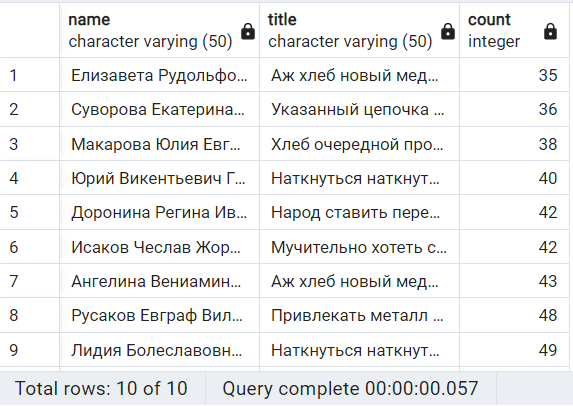
WHERE "Track".duration < '01:00:00'

ORDER BY count DESC

LIMIT 10

) new\_table

ORDER BY count



#### Вывести первые 10 исполнителей и число их треков, осортированных в порядке убывания количества треков.

SELECT name, COUNT(track\_id) num\_tracks FROM "Track"

INNER JOIN "Track-Artist" USING (track\_id)

INNER JOIN "Artist" USING (artist\_id)

GROUP BY artist\_id, name

ORDER BY num\_tracks DESC

LIMIT 10

#### 

#### Вывести 10 артистов, выступающих в жанре ‘Rock’ и осортированных по имени, и информацию о них.

SELECT "Artist".name artist FROM "Genre-Artist"

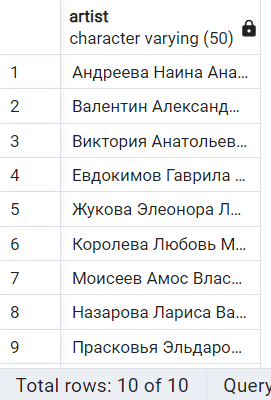
INNER JOIN "Genre" USING (genre\_id)

INNER JOIN "Artist" USING (artist\_id)

WHERE "Genre".name = 'Rock'

ORDER BY "Artist"."name"

LIMIT 10



#### Вывести 10 записей c именем артиста и количеством его ссылок, переименованных в num\_links, упорядоченных по числу ссылок по убыванию.

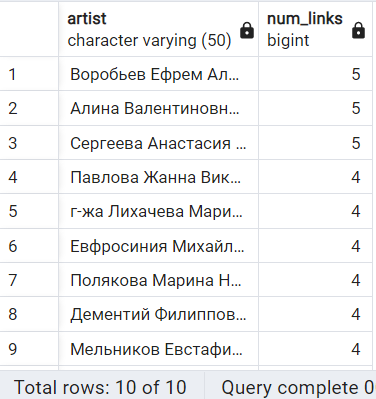
SELECT "Artist".name artist, COUNT(link\_id) num\_links FROM "Link"

INNER JOIN "Artist" USING (Artist\_id)

GROUP BY "Artist".name

ORDER BY num\_links DESC

LIMIT 10



#### Найти топ-3 пользователей по количеству плейлистов подкастов. Вывести их имена, а также количество плейлистов.

SELECT name, COUNT("PodcastPlaylist".podcastplaylist\_id) num\_podcasts FROM "User-PodcastPlaylist"

INNER JOIN "User" USING (user\_id)

INNER JOIN "PodcastPlaylist" USING (podcastplaylist\_id)

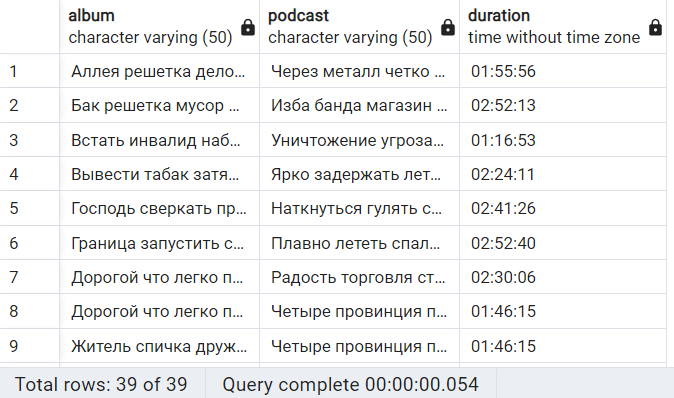
GROUP BY name

ORDER BY num\_podcasts DESC

LIMIT 3

#### 

#### Вывести все альбомы подкастов, сами подкасты, их длительность, где нет марировки e, а длительность составляет от часу до 3 часов. Результаты отсортировать по названию альбома и по названию подкаста.



SELECT "PodcastAlbum".title Album, "Podcast".title Podcast, duration FROM "PodcastAlbum-Podcast"

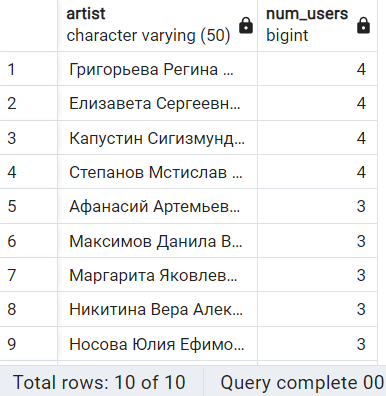
INNER JOIN "PodcastAlbum" USING (podcastalbum\_id)

INNER JOIN "Podcast" USING (podcast\_id)

WHERE "PodcastAlbum"."isExplicit" = False AND duration BETWEEN '01:00:00' AND '03:00:00'

ORDER BY "PodcastAlbum".title, "Podcast".title

1. Найти топ-10 артистов по количеству лайков от премиум-пользователей. Вывести их ФИО, а также число лайков, отсортированные сначала по числу лайков по убыванию, потом по имени по алфавиту.



SELECT "Artist".name artist, COUNT("User"."name") num\_users FROM "Artist-User"

INNER JOIN "Artist" ON "Artist".artist\_id = "Artist-User".artist\_id

INNER JOIN

(SELECT \* FROM "User"

WHERE "User"."hasPremium" = True) "User"

ON "User".user\_id = "Artist-User".user\_id

GROUP BY "Artist".artist\_id

ORDER BY num\_users DESC, artist

LIMIT 10

#### Вывести ФИО пользователей, кто больше всего суммарно прослушал треков (таких может быть несколько). Учесть, что названия треков и ФИО пользователей могут совпадать. Отсортировать по алфавиту.

SELECT name FROM "User"

WHERE user\_id IN (SELECT user\_id FROM "Track-User-Listen"

INNER JOIN "Track" USING (track\_id)

INNER JOIN "User" USING (user\_id)

GROUP BY user\_id

HAVING SUM(count) = (SELECT SUM(count) FROM "Track-User-Listen"

INNER JOIN "Track" USING (track\_id)

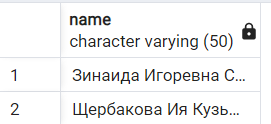
INNER JOIN "User" USING (user\_id)

GROUP BY user\_id

ORDER BY sum DESC

LIMIT 1))

ORDER BY name



#### Найти треки, которые заблокировало наибольшее число пользователей. Учесть, что названия треков, ФИО пользователей могут сопадать. Отсортировать по алфавиту.

SELECT title FROM "Track"

WHERE track\_id IN (SELECT track\_id FROM "Track-User-Block"

INNER JOIN "Track" USING (track\_id)

INNER JOIN "User" USING (user\_id)

GROUP BY track\_id

HAVING COUNT(user\_id) = (SELECT COUNT(user\_id) FROM "Track-User-Block"

INNER JOIN "Track" USING (track\_id)

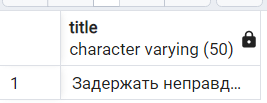
INNER JOIN "User" USING (user\_id)

GROUP BY track\_id

ORDER BY count DESC

LIMIT 1))

ORDER BY title



#### Найти топ-3 самых популярных жанра у треков с пометкой e (isExplicit). Суммарная продолжительность треков одного жанра должна быть более 30 часов. Вывести название жанров, количество их треков.

SELECT name, COUNT(track\_id) FROM "Genre-Track"

INNER JOIN (SELECT \* FROM "Track" WHERE "Track"."isExplicit" = true) "Track" USING (track\_id)

INNER JOIN "Genre" USING (genre\_id)

GROUP BY name

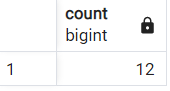
HAVING SUM(duration) > '30:00:00'

ORDER BY count DESC

LIMIT 3

#### 

1. Найти подкасты, которые прослушало наименьшее число премиум пользователей. Учитывать только подкасты, длящиеся менее часа. Также учесть, что названия подкастов и ФИО пользователей могут повторяться. Вывести единственную строчку – их количество.



SELECT COUNT(title) FROM "Podcast"

WHERE podcast\_id IN (SELECT podcast\_id FROM "User-Podcast-Listen"

INNER JOIN (SELECT \* FROM "User" WHERE "User"."hasPremium" = TRUE) "User" USING (user\_id)

INNER JOIN (SELECT \* FROM "Podcast" WHERE "Podcast"."duration" < '01:00:00') "Podcast" USING (podcast\_id)

GROUP BY podcast\_id

HAVING COUNT(user\_id) = (SELECT COUNT(user\_id) FROM "User-Podcast-Listen"

INNER JOIN (SELECT \* FROM "User" WHERE "User"."hasPremium" = TRUE) "User" USING (user\_id)

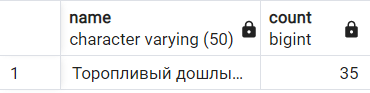
INNER JOIN (SELECT \* FROM "Podcast" WHERE "Podcast"."duration" < '01:00:00') "Podcast" USING (podcast\_id)

GROUP BY podcast\_id

ORDER BY count

LIMIT 1))

#### Вывести любимый жанр у пользователя с id = 3, который суммарно прослушал треки этого жанра менее чем на 10 часов. Также вывести количество его треков в этом жанре.



SELECT "Genre".name, COUNT(track\_id) FROM "Track-User-Like"

INNER JOIN "Track" USING (track\_id)

INNER JOIN (SELECT \* FROM "User" WHERE user\_id = 3) "User" USING (user\_id)

INNER JOIN "Genre-Track" USING (track\_id)

INNER JOIN "Genre" USING (genre\_id)

GROUP BY user\_id, "Genre".name

HAVING SUM("Track".duration) < '10:00:00'

ORDER BY count DESC

LIMIT 1

#### Вывести не премиум пользователя, который лайкнул более 3 подкастов на наибольшее суммарное время. Учесть, что названия подкастов могут совпадать. Вывести пользователя и это время.

SELECT name, sum FROM "User"

RIGHT JOIN (SELECT user\_id, SUM(duration) FROM "User-Podcast-Like"

INNER JOIN (SELECT \* FROM "User" WHERE "User"."hasPremium" = FALSE) "User" USING (user\_id)

INNER JOIN "Podcast" USING (podcast\_id)

GROUP BY user\_id

HAVING COUNT(podcast\_id) > 3

ORDER BY sum DESC

LIMIT 1) t1 USING (user\_id)