Санкт-Петербургский политехнический университет Петра Великого Институт компьютерных наук и технологий Кафедра компьютерных систем и программных технологий

Отчёт по лабораторным работам Дисциплина:

Базы данных

Выполнил студент гр.3530901/70201 Белов Е. А.

Преподаватель Мяснов А. В.

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург 2020

Оглавление

[1. Лабораторная работа №1 4](#_Toc41745921)

[1.1 Цель работы 4](#_Toc41745922)

[1.2 Программа работы 4](#_Toc41745923)

[1.3 Выполнение работы 4](#_Toc41745924)

[1.3.1 Ссылка на созданный на GitLab проект: 4](#_Toc41745925)

[1.3.2 Предметная область: 4](#_Toc41745926)

[1.3.3 Описание таблиц: 4](#_Toc41745927)

[1.3.4 Схема БД 6](#_Toc41745928)

[1.3.5 Скрипт создания БД 7](#_Toc41745929)

[1.3.6 Изменение БД 8](#_Toc41745930)

[1.3.7 Окончательная схема БД 9](#_Toc41745931)

[1.3.8. Скрипт изменения 9](#_Toc41745932)

[1.4 Вывод 10](#_Toc41745933)

[2 Лабораторная работа № 2 12](#_Toc41745934)

[2.1 Цель работы 12](#_Toc41745935)

[2.2 Программа работы 12](#_Toc41745936)

[2.3 Выполнение работы 12](#_Toc41745937)

[2.4 Вывод 17](#_Toc41745938)

[3 Лабораторная работа №3 18](#_Toc41745939)

[3.1 Цель работы 18](#_Toc41745940)

[3.2 Программа работы 18](#_Toc41745941)

[3.3 Выполнение работы 18](#_Toc41745942)

[3.3.1 Стандартные запросы 18](#_Toc41745943)

[3.3.2 Индивидуальные задания 24](#_Toc41745944)

[3.4 Вывод 26](#_Toc41745945)

[4 Лабораторная работа №4 26](#_Toc41745946)

[4.1 Цель работы 26](#_Toc41745947)

[4.2 Программа работы 26](#_Toc41745948)

[4.3 Выполнение работы 26](#_Toc41745949)

[4.3.1 Типовые запросы пользователей 26](#_Toc41745950)

[4.3.2 Моделирование нагрузки на БД и снятие показателей 27](#_Toc41745951)

[4.3.3 Оптимизация 30](#_Toc41745952)

[4.4 Сравнительный анализ результатов 32](#_Toc41745953)

[4.4.1 Анализ графических результатов 32](#_Toc41745954)

[4.5 Вывод 33](#_Toc41745955)

# 1. Лабораторная работа №1

## 1.1 Цель работы

Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД, а также изучить язык SQL.

## 1.2 Программа работы

• Создание проекта для работы в GitLab

• Выбор задания (предметной области), описание набора данных

• Формирование в свободном формате схемы БД, соответствующей заданию

• Создание скрипта, генерирующего БД согласно схеме

• Изменение скрипта создания БД согласно требованиям преподавателя

• Окончательный вид схемы БД.

## 1.3 Выполнение работы

### 1.3.1 Ссылка на созданный на GitLab проект:

<http://gitlab.icc.spbstu.ru/Egorius/db_2020>

### 1.3.2 Предметная область:

Создание базы данных музыкального сервиса

### 1.3.3 Описание таблиц:

* Group – таблица группы
  + Name – имя группы
  + Country – страна группы
  + Icon\_path – путь к файлу с иконкой
  + Desc – описание группы
* Artist – таблица артиста
  + Name – имя артиста
  + Country – страна артиста
  + Icon\_path – путь к файлу с иконкой
  + Desc – описание группы
  + Group – внешний ключ на группу, принадлежность группе
* Groups\_table – таблица связи между группой и песней
  + Creator – внешний ключ на группу
  + Song – внешний ключ на песню
* artists\_table – таблица связи между артистом и песней
  + Creator – внешний ключ на группу
  + Song – внешний ключ на песню
* Song – таблица с информацией о песне
  + Id
  + Name
  + Path – путь к файлу
  + Album – внешний ключ на альбом
* Album – таблица с информацией об альбоме
  + Id
  + Name
  + Year – год выпуска
  + Duration – продолжительность альбома
  + Quantity – количество песен в альбоме
  + Icon\_path – путь к иконке альбома
* Playlist – таблица с информацией о плейлисте
  + Id
  + Name
  + Duration – продолжительность альбома
  + Quantity – количество песен в альбоме
  + Icon\_path – путь к иконке альбома
  + Author – создатель плейлиста
* Playlists\_table – таблица связи песен и плейлистов
  + Playlist – внешний ключ на плейлист
  + Song - внешний ключ на песню
* User – таблица с информацией о пользователе
  + Email
  + Name
  + Icon\_path
  + Password – пароль
* Like\_song – таблица связи песен, которых лайкнул пользователь
  + User
  + Song
* Like\_album – таблица связи альбомов, которых лайкнул пользователь
  + User
  + Album
* Like\_group – таблица связи групп, которых лайкнул пользователь
  + User
  + Group
* Like\_playlist – таблица связи плэйлстов, которых лайкнул пользователь
  + User
  + Playlist

### 1.3.4 Схема БД

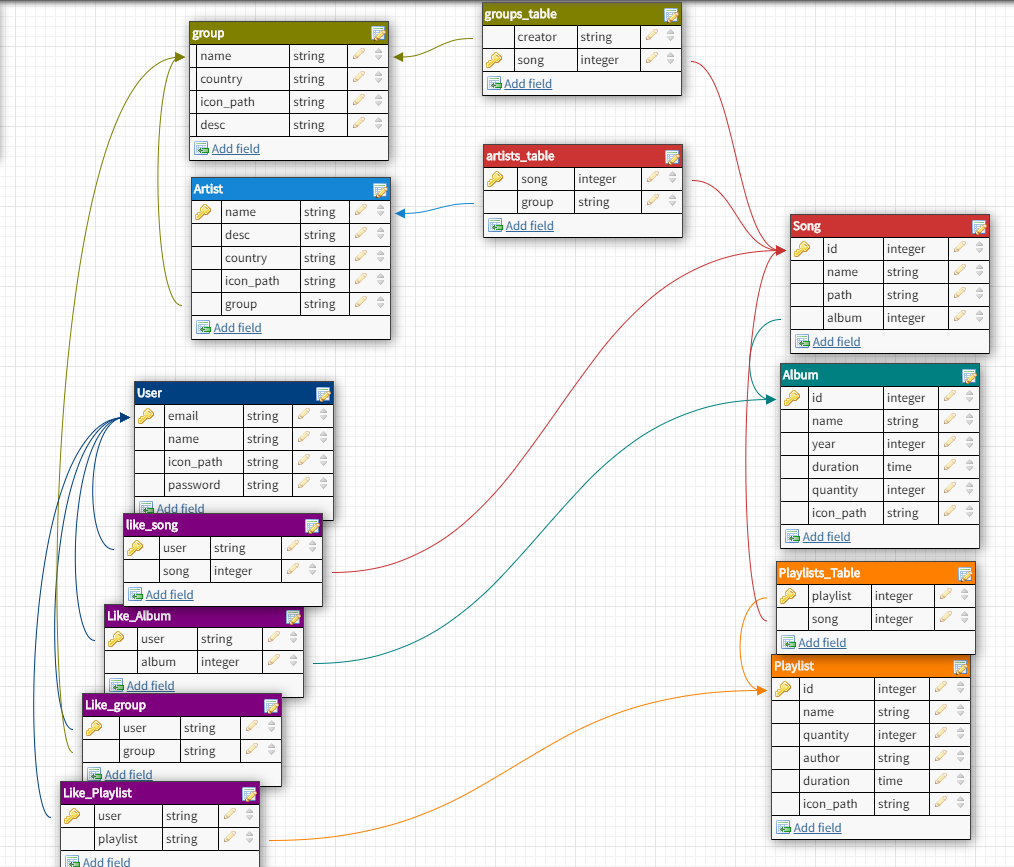


Рис. 1. Схема БД

### 1.3.5 Скрипт создания БД

Листинг. 1.

|  |
| --- |
| start transaction;  drop table if exists "group", artist, groups\_table, like\_group, artists\_table, history\_artist\_table; create table "group" (  name varchar(255) primary key,  "desc" varchar(255),  country varchar(255),  icon\_path varchar(255) );  drop table if exists artist; create table artist (  name varchar(255) primary key,  "desc" varchar(255),  country varchar(255),  icon\_path varchar(255),  "group" varchar(255) references "group" (name) );  drop table if exists album, like\_album, song, songs\_table, like\_song, playlists\_table, prizes\_albums\_table, prizes\_songs\_table; create table album (  id integer primary key,  name varchar(255) not null,  year integer not null,  duration time not null,  quantity integer not null,  icon\_path varchar(255) not null );  drop table if exists song; create table song (  id integer primary key,  name varchar(255) not null,  path varchar(255) not null,  album integer references album (id) );  drop table if exists groups\_table; create table groups\_table (  song integer references song (id),  creator varchar(255) references "group" (name),  primary key (song, creator) );  drop table if exists artists\_table; create table artists\_table (  song integer references song (id),  creator varchar(255) references artist (name),  primary key (song, creator)  );   drop table if exists playlist, like\_playlist, playlists\_table; create table playlist (  id integer primary key,  name varchar(255) not null,  quantity integer not null,  author varchar(255) not null,  duration time not null,  icon\_path varchar(255) );  drop table if exists playlists\_table; create table playlists\_table (  playlist integer references playlist (id) not null,  song integer references song (id) not null );  drop table if exists "user", like\_song, like\_playlist, like\_album; create table "user" (  email varchar(255) primary key not null,  name varchar(255) not null,  icon\_path varchar(255),  password varchar(255) not null  );  drop table if exists like\_song; create table like\_song (  "user" varchar(255) references "user" (email) not null,  song integer references song (id) not null );  drop table if exists like\_album; create table like\_album (  "user" varchar(255) references "user" (email) not null,  album integer references album (id) not null );  drop table if exists like\_group; create table like\_group (  "user" varchar(255) references "user" (email) not null,  "group" varchar(255) references "group" (name) not null );  drop table if exists like\_playlist; create table like\_playlist (  "user" varchar(255) references "user" (email) not null,  playlist integer references playlist (id) not null );  commit; |

### 1.3.6 Изменение БД

Был добавлен тип альбома для таблицы album: сборник, концертный диск или обычный альбом. Добавлена информация о наградах для песен и альбомов. Добавлена историчность для песни: год создания песни. Добавлены историчность для артистов в группе. В общей совокупности были добавлены таблицы:

* History\_artist\_table – информация об историчности артиста в группе
  + Artist – внешний ключ на артиста
  + Group – внешний ключ на группу
  + Start\_date – год вступления артиста в группу
  + End\_date – год выхода артиста из группы
* Prize – информация о награде
  + Id
  + Name
  + Year
  + Description
* Prizes\_songs\_table – таблица связи наград и песен
  + Song
  + Prize
* Prizes\_albums\_table – таблица связи наград и альбомов
  + Song
  + Album

### 1.3.7 Окончательная схема БД

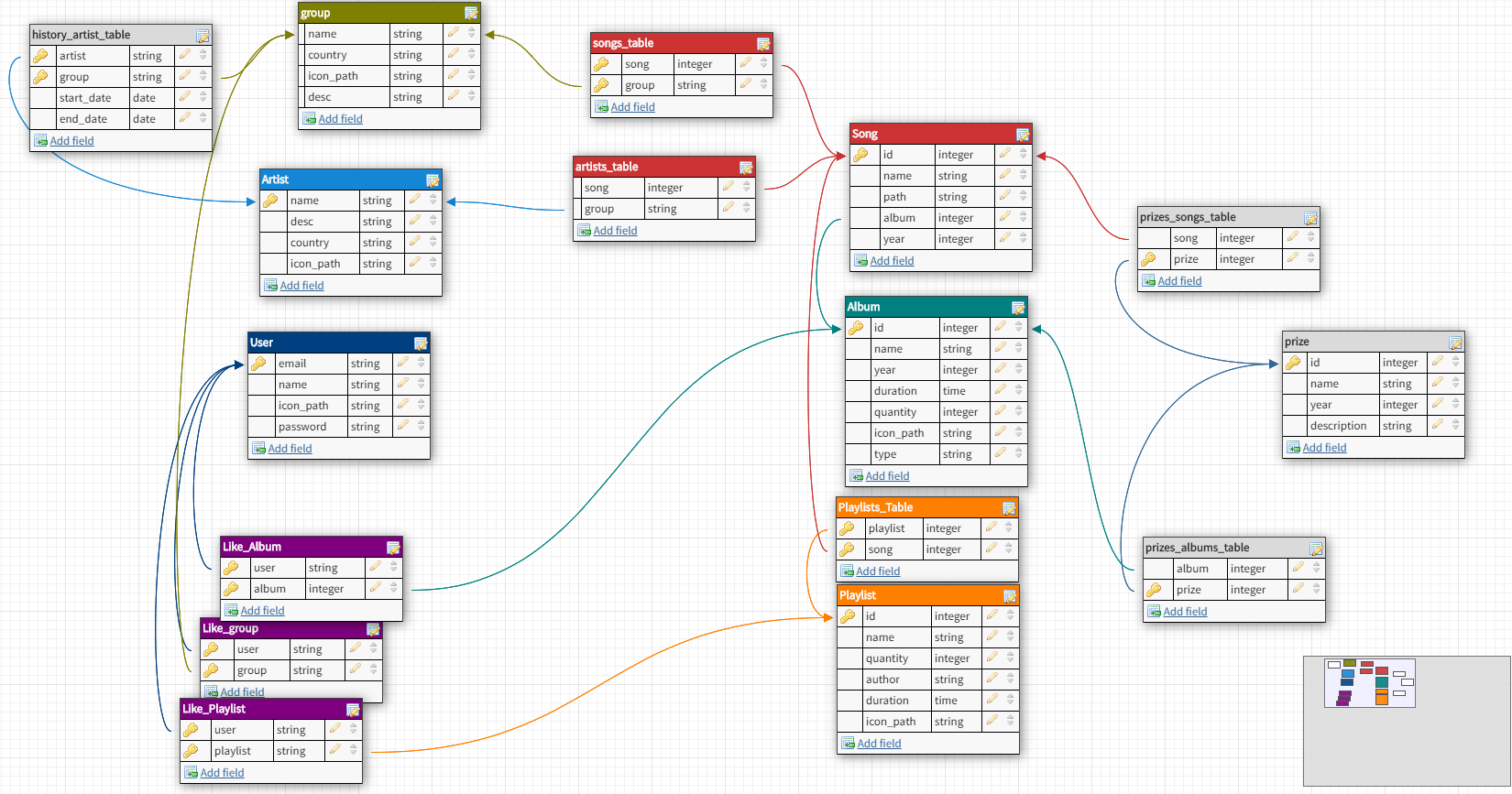


Рис. 2. Измененная схема БД.

### 1.3.8. Скрипт изменения

Листинг 2.

|  |
| --- |
| start transaction;  drop table if exists prize, prizes\_albums\_table, prizes\_songs\_table; create table prize (  id integer primary key,  name varchar(255) not null,  year integer not null,  description varchar(255) );  drop table if exists prizes\_songs\_table; create table prizes\_songs\_table (  song integer references song (id),  prize integer primary key references prize (id) );  drop table if exists prizes\_albums\_table; create table prizes\_albums\_table (  album integer references album (id),  prize integer primary key references prize (id) );    alter table song  drop column if exists year; alter table song  add column year integer;   alter table artist  drop column if exists "group";  drop table if exists history\_artist\_table; create table history\_artist\_table (  artist varchar(255) references artist (name) not null,  "group" varchar(255) references "group" (name) not null ,  start\_date integer not null,  end\_date integer not null,  primary key (artist, "group") );  alter table album  drop column if exists "type"; drop type if exists album\_type; create type album\_type as enum ('album', 'compilation', 'concert'); alter table album  add column "type" album\_type not null default 'album';  commit; |

## 1.4 Вывод

В данной лабораторной работе был создан проект на GitLab, выбрана предметная область базы данных, описан набор данных, сформирована схема БД в графическом формате, а также был изучен язык SQL, с помощью которого были разработаны скрипты для создания таблиц согласно схеме, а также скрипт изменения таблиц согласно заданию преподавателя. Сложно было разобраться как составить связи между таблицами и спроектировать схему.

# 2 Лабораторная работа № 2

## 2.1 Цель работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

## 2.2 Программа работы

• Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.

• Количество записей в справочных таблицах должно соответствовать ограничениям предметной области

• Количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации

• Значения для внешних ключей необходимо брать из связанных таблиц

## 2.3 Выполнение работы

Для выполнения лабораторной работы было создано CLI приложение на языке Python. Работа с базы данных осуществляется с помощью библиотеки psycopg2, для работы с командной строкой – click. Для генерации осмысленных данных используется библиотека mimesis. Приложение поддерживает генерацию данных в каждую отдельную таблицу, во всю БД сразу и в отдельные области базы данных (в данной БД обозначено 4 области: songs\_area, groups\_area, users\_area, prizes\_area). Таже самая функциональность поддерживается для удаления записей из таблиц. Пользователь может сам задавать сколько записей сгенерируется в таблицах.

Команды приложения и их использование:

Листинг 3.

|  |
| --- |
| Usage: Main.py [OPTIONS] COMMAND [ARGS]...  Options:  --help Show this message and exit.  Commands:  add  delete |
| Usage: Main.py add [OPTIONS]  Options:  -table TEXT Name of the table  -n INTEGER The number of records  --help Show this message and exit. |
| Usage: Main.py delete [OPTIONS]  Options:  -table TEXT Delete all records in table  --help Show this message and exit |

Один из главных компонентов программы – класс BaseGenerator. От него наследуются все остальные генераторы, которые используются для генерации данных в основные таблицы БД. В нем хранятся переменные библиотеки mimesis для генерации данных. В классе определенно две абстракные функции get\_primary\_key\_data(self, num) и get\_params(self). Первый служит для генерации данных для первичных ключей, второй – для генерации остальных полей таблиц.

Листинг 4. BaseGenerator

|  |
| --- |
| from mimesis import Address from mimesis import Generic from mimesis import Person from mimesis import Text   class BaseGenerator(object):  person = Person('en')  text = Text('en')  generic = Generic('en')  address = Address('en')  data = set()   def get\_primary\_key\_data(self, num):  raise NotImplementedError("Необходимо переопределить метод")   def get\_params(self):  raise NotImplementedError("Необходимо переопределить метод")   def clear\_data(self):  if len(self.data) != 0:  self.data.clear() |

Пример реализации BaseGenerator UserGenerator, который генерирует данные для таблицы user:

Листинг 5. UserGenerator

|  |
| --- |
| from BaseGenerator import BaseGenerator   class UserGenerator(BaseGenerator):   def get\_params(self):  user = {  'name': self.person.name(),  'icon\_path': self.person.age(),  'password': self.person.password()  }  return user   def get\_primary\_key\_data(self, num):  self.clear\_data()  while num != 0:  email = self.person.email()   if not (email in self.data):  self.data.add(self.person.email())  num = num - 1 |

В файле Adder.py находятся функции для добавления данных в таблицы. С помощью объектов классов генератора функции берут данные и выполняют запросы к БД с добавлением данных.

Листинг 6. Функция добавления данных в таблицу

|  |
| --- |
| def add\_users(n):  cursor, conn = open\_connection()  gen\_users = UserGenerator()  gen\_users.get\_primary\_key\_data(n)  for u in gen\_users.data:  params = gen\_users.get\_params()  print(u, params, sep=’: ‘)  cursor.execute(‘insert into “user” (email, name, icon\_path, password) values(%s, %s, %s, %s);commit;’,  (u, params.get(‘name’), params.get(‘icon\_path’), params.get(‘password’)))  print(n, “Records added”, sep=’ ‘)  close\_connection(cursor, conn) |

В файле Deleter.py находятся функции с запросами на удаление данных из областей БД.

В файле Main.py происходит обработка команд, который вводит пользователь.

Листинг 7. Main.py

|  |
| --- |
| import click from Adder import \* from Utils import \* from Deleter import \*   @click.group() def cli():  pass   @click.command() @click.option('-table', default="group", help='Name of the table') @click.option('-n', default=10\_000, help='The number of records') def add(table, n):  if table == "user":  add\_users(n)  elif table == "group":  add\_groups(n)  elif table == "artist":  add\_artists(n)  elif table == "album":  add\_albums(n)  elif table == "song":  add\_songs(n)  elif table == "playlist":  add\_playlists(n)  elif table == "songs\_area":  add\_songs\_area(n)  elif table == "groups\_area":  add\_groups\_area(n)  elif table == "users\_area":  add\_users\_area(n)  elif table == "prizes\_area":  add\_prizers\_area(n)  elif table == "all":  add\_all(n)  else:  print('table does not exist')   @click.command() @click.option('-table', default="group", help='Delete all records in table') def delete(table):  cursor, conn = open\_connection()  if table == "album":  cursor.execute('delete from album;')  elif table == "artist":  cursor.execute('delete from artist;')  elif table == "user":  cursor.execute('delete from "user";')  elif table == "group":  cursor.execute('delete from "group";')  elif table == "song":  cursor.execute('delete from song;')  elif table == "playlist":  cursor.execute('delete from playlist;')  elif table == "playlists\_table":  cursor.execute('delete from playlists\_table;')  elif table == "songs\_area":  delete\_songs\_area(cursor)  elif table == "groups\_area":  delete\_groups\_area(cursor)  elif table == "users\_area":  delete\_users\_area(cursor)  elif table == "prizes\_area":  delete\_prizers\_area(cursor)  elif table == "all":  delete\_prizers\_area(cursor)  delete\_users\_area(cursor)  delete\_groups\_area(cursor)  delete\_songs\_area(cursor)   cursor.execute("commit;")  print("All records were deleted in ", table)  close\_connection(cursor, conn)   cli.add\_command(delete) cli.add\_command(add)  if \_\_name\_\_ == '\_\_main\_\_':  cli() |

Примеры сгенерированных данных с помощью приложения:

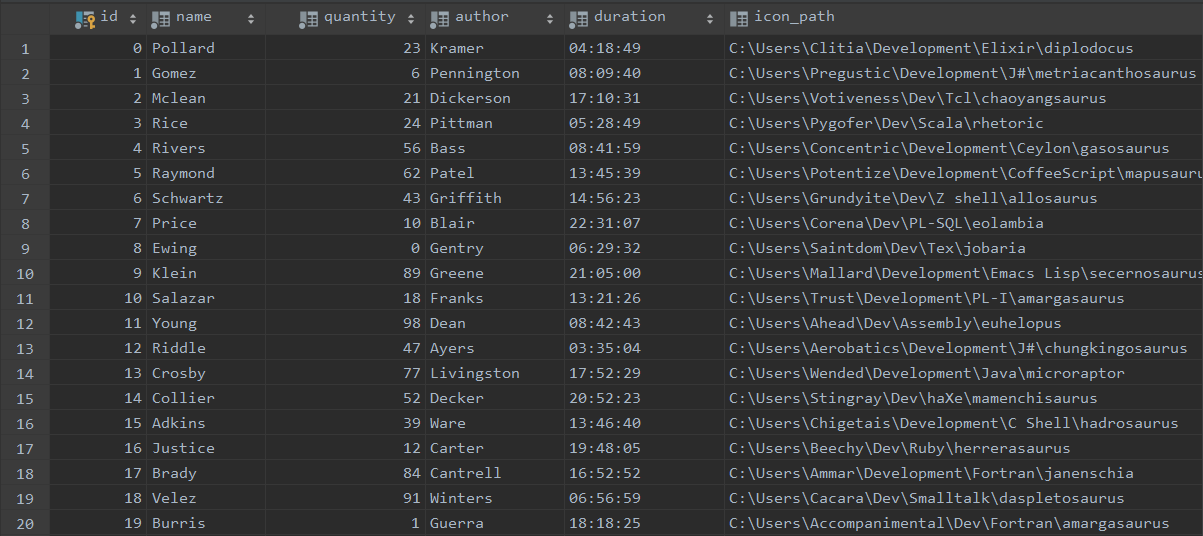


Рис. 3. Сгенерированные данные в таблице playlist.

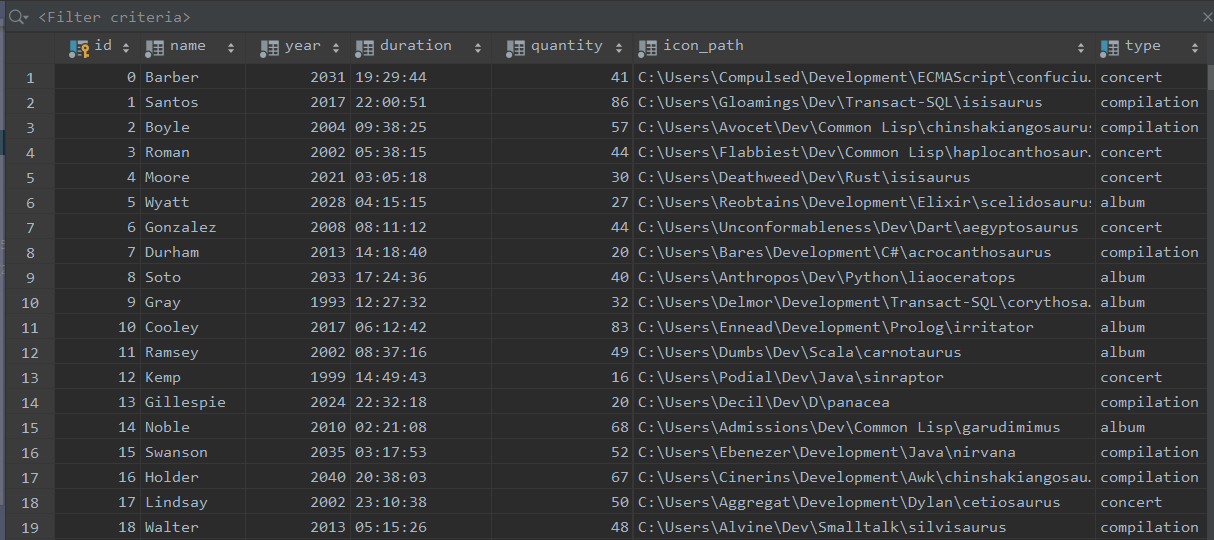


Рис. 4. Сгенерированные данные в таблице album.

Полный код программы доступен в репозитории GitLab:

<http://gitlab.icc.spbstu.ru/Egorius/db_2020/tree/master/lab2/DataGenerator>

## 2.4 Вывод

В данной лабораторной работе мной был написан генератор тестовых данных для базы данных. В ходе работы пришлось столкнуться с несовершенством библиотеки mimesis, которая использовалась для генерации данных. После генерации ~5000 записей, библиотека могла выдавать повторяющиеся значения для определенных типов данных, например генерация email для таблицы user. Это был критический момент так как email является первичным ключом в таблице user. Проблема была решена сменой типом данных или промежуточным хранением генерируемых данных в множестве (set) в самой программе, что не могло не сказаться на производительности программы. Также документация к библиотеке иногда не соответствовала действительности, некоторые функции в ней не работали.

# 3 Лабораторная работа №3

## 3.1 Цель работы

Познакомиться с языком создания запросов управления данными SQL-DML.

3.2 Программа работы

• Изучение SQL-DML.

• Выполнение всех запросов из списка стандартных запросов.

• Получение у преподавателя и реализация SQL-запросов в соответствии с индивидуальным заданием.

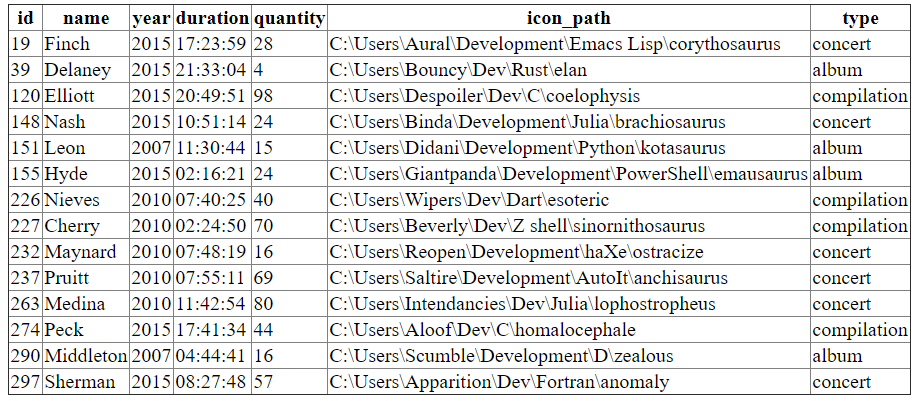
## 3.3 Выполнение работы

### 3.3.1 Стандартные запросы

Листинг 8. Стандартные запросы

|  |
| --- |
| -- 1. Сделайте выборку всех данных из каждой таблицы select \* from album; select \* from artist; select \* from artists\_table; select \* from "group"; select \* from groups\_table; select \* from history\_artist\_table; select \* from like\_album; select \* from like\_group; select \* from like\_playlist; select \* from like\_song; select \* from playlist; select \* from playlists\_table; select \* from prize; select \* from prizes\_albums\_table; select \* from prizes\_songs\_table; select \* from song; select \* from "user";  -- 2. Сделайте выборку данных из одной таблицы при нескольких условиях, с использованием логических операций, LIKE, BETWEEN, IN select email, name from "user" where email like '%yahoo%';  select id, name, year from song where year between 1990 and 2010;  select \* from album where year in (2007, 2010, 2015);  -- 3. Создайте в запросе вычисляемое поле select id, name 'средняя длительность песни', (duration / quantity) from playlist;  -- 4. Сделайте выборку всех данных с сортировкой по нескольким полям select \* from playlist order by quantity, duration;  -- 5. Создайте запрос, вычисляющий несколько совокупных характеристик таблиц select name, *count*(\*) as num\_of\_names from "user" where email like '%yandex%' group by name order by name;  -- 6. Сделайте выборку данных из связанных таблиц (не менее двух примеров) select "group".name, country, groups\_table.song, hat.artist from "group"  join groups\_table on "group".name = groups\_table.creator  join history\_artist\_table hat on "group".name = hat."group";  select pst.song, prize.id as prize\_id from prize  right join prizes\_songs\_table pst on prize.id = pst.prize;  -- 7. Создайте запрос, рассчитывающий совокупную характеристику с использованием группировки, наложите ограничение на результат группировки select hat.start\_date, *count*(\*) as num\_of\_start\_date from artist  join history\_artist\_table hat on artist.name = hat.artist group by hat.start\_date having *count*(\*) > 1;   -- 8. Придумайте и реализуйте пример использования вложенного запроса select id, name from song where album in  (select id  from album  where type = 'compilation');  -- 9. С помощью оператора INSERT добавьте в каждую таблицу по одной записи insert into song (id, name, path, album, year) values ((select *count*(\*) from song) + 1, 'My Love', 'D/songs/mylove.mp3',  (select id from album order by *random*() limit 1), 2019);  insert into album (id, name, year, duration, quantity, icon\_path, type) values ((select *count*(\*) from album) + 1, 'album', 2017, '01:37', 25, 'D:\store\album\1.png', 'album');  insert into groups\_table (song, creator) values ((select id from song order by *random*() limit 1), (select name from "group" order by *random*() limit 1));  insert into artists\_table (song, creator) values ((select id from song order by *random*() limit 1), (select name from artist order by *random*() limit 1));  insert into playlist (id, name, quantity, author, duration, icon\_path) values ((select *count*(\*) from song) + 1, 'df', 10, 'ff', '11:33', 'D:\store\album\1.png');  insert into playlists\_table (playlist, song) values ((select id from playlist order by *random*() limit 1), (select id from song order by *random*() limit 1));  -- 10. С помощью оператора UPDATE измените значения нескольких полей у всех записей, отвечающих заданному условию update album set year = 2020,  quantity = 10 where type = 'concert';  -- 11. С помощью оператора DELETE удалите запись, имеющую максимальное (минимальное) значение некоторой совокупной характеристики delete from song where year = (select *max*(year) from song);   -- 12. С помощью оператора DELETE удалите записи в главной таблице, -- на которые не ссылается подчиненная таблица (используя вложенный запрос) delete from prize where id not in (select prize from prizes\_songs\_table); |

Результаты выполнения запросов:



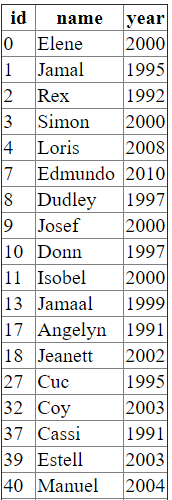
 

Рис. 5. Результаты выполнения запросов 2 пункта.

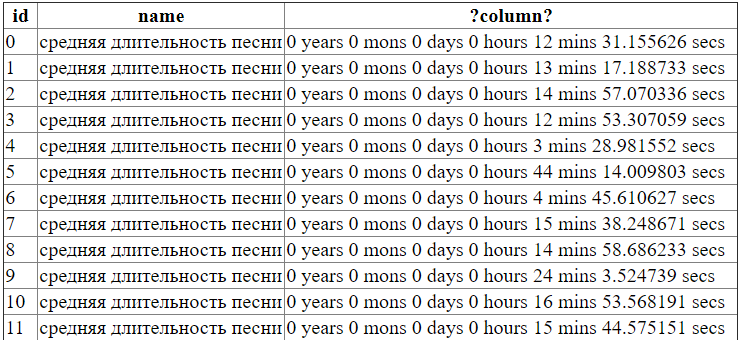


Рис. 6. Результаты выполнения запросов 3 пункта.

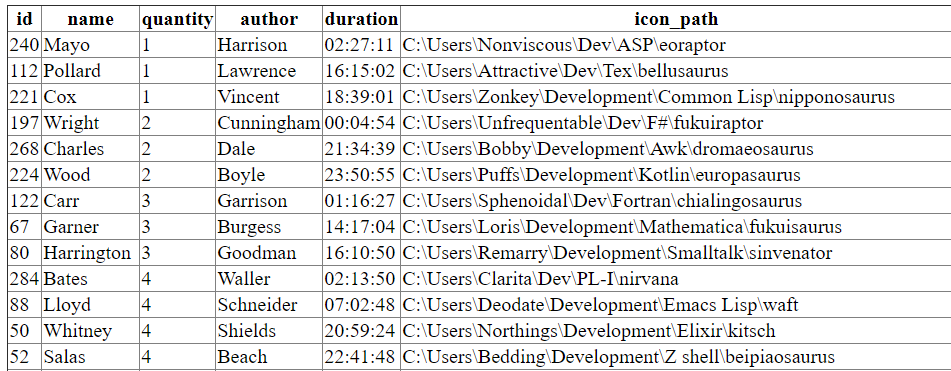


Рис. 7. Результаты выполнения запросов 4 пункта.

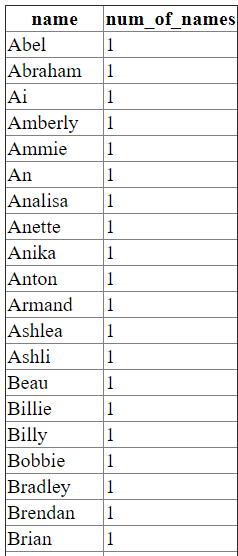


Рис. 8. Результаты выполнения запросов 5 пункта.

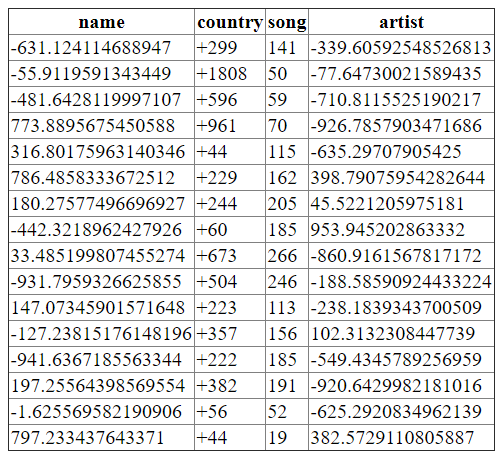
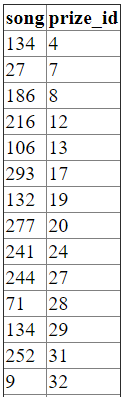
 

Рис. 9. Результаты выполнения запросов 6 пункта.

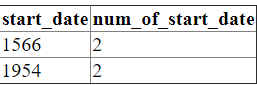


Рис. 10. Результаты выполнения запросов 7 пункта.



Рис. 11. Результаты выполнения запросов 8 пункта.

### 3.3.2 Индивидуальные задания

1. Вывести 5 исполнителей, с наибольшим отношением количества композиций, в которых они участвовали без групп, к количеству, где были в составе групп.

Листинг 9. Индивидуальное задание 1.

|  |
| --- |
| -- Вывести 5 исполнителей, с наибольшим отношением количества композиций, -- в которых они участвовали без групп, к количеству, где были в составе групп. select a.creator,  num\_of\_songs,  num\_of\_song\_with\_group,  num\_of\_songs - num\_of\_song\_with\_group as num\_of\_song\_without\_group,  *cast*((num\_of\_songs - num\_of\_song\_with\_group) as real) / *cast*((num\_of\_song\_with\_group) AS REAL) as res from (select art.creator, *count*(a.name) as num\_of\_song\_with\_group  from artist a  join artists\_table art on a.name = art.creator  join song s on art.song = s.id  join history\_artist\_table ht on a.name = ht.artist  join "group" g on ht."group" = g.name  where s.year between ht.start\_date and ht.end\_date  group by art.creator  ) a  join  -- все песни артиста  (select creator  , *count*(artists\_table.song) as num\_of\_songs  from artists\_table  group by creator) b  on a.creator = b.creator where num\_of\_song\_with\_group > 0 order by res desc limit 5; |

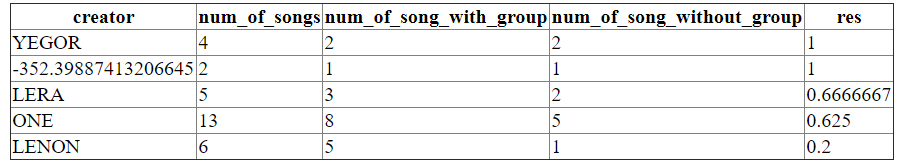


Рис. 12. Результат выполнения индивидуальное задания 1.

2. Вывести группы, в которых все участники индивидуально получили меньше наград, чем в составе групп.

Листинг 10. Индивидуальное задание 2.

|  |
| --- |
| -- Вывести группы, в которых все участники индивидуально получили меньше наград, чем в составе групп. select sum\_gr.name\_gr as group\_name,  sum\_gr.num\_gr\_prize as num\_of\_songs\_with\_prize\_in\_group,  sum\_art.num\_prize\_art as num\_of\_songs\_with\_prize\_artist from (select name\_gr, *count*(prize\_art) as num\_gr\_prize  from (select g.name as name\_gr, song  From artist a  join history\_artist\_table hat on a.name = hat.artist  join artists\_table art on a.name = art.creator  Left join song s on art.song = s.id  join "group" g on hat."group" = g.name  Where year between start\_date and end\_date) d  Left join  -- количество наград у песни  (Select song  , *Count*(prizes\_songs\_table.prize) as prize\_art  From prizes\_songs\_table  group by song) e  On d.song = e.song  Group by name\_gr) sum\_gr  join  (  Select name\_gr, *count*(prize\_art) as num\_prize\_art  From -- — считаем все награды у исполнителя вне группы (select g.name as name\_gr, song  From artist a  join history\_artist\_table hat on a.name = hat.artist  join artists\_table art on a.name = art.creator  Left join song s on art.song = s.id  join "group" g on hat."group" = g.name  Where year not between start\_date and end\_date) c  Left join -- — количество наград у песни  (Select song, *Count*(prizes\_songs\_table.prize) as prize\_art  From prizes\_songs\_table  group by song) b On c.song = b.song  Group by name\_gr) sum\_art  on sum\_gr.name\_gr = sum\_art.name\_gr Where num\_gr\_prize > num\_prize\_art; |



Рис. 12. Результат выполнения индивидуальное задания 2.

## 3.4 Вывод

В данной лабораторной работе мною были изучены возможности создания запросов, и получения различных данных из таблицы. Были сложности с выполнением индивидуальных заданий, а именно со связыванием таблиц большого количества таблиц.

# 4 Лабораторная работа №4

## 4.1 Цель работы

Знакомство с проблемами, возникающими при высокой нагрузке на базу данных, и методами их решения, путем оптимизации запросов.

## 4.2 Программа работы

• Написание параметризированных типовых запросов пользователей

• Моделирование нагрузки базы данных

• Снятие показателей работы сервиса и построение соответствующих графиков

• Применение возможных оптимизаций запросов и повторное снятие показателей

• Сравнительный анализ результатов

## 4.3 Выполнение работы

### 4.3.1 Типовые запросы пользователей

Листинг 11. Типовые запросы пользователей

|  |
| --- |
| query\_1 = "select email, " \  "name," \  "ls.song as like\_song\_id," \  "la.album as like\_album\_id," \  "lg.\"group\" as like\_group\_name," \  "lp.playlist as like\_playlit\_id " \  "from \"user\"" \  "join like\_song ls on \"user\".email = ls.\"user\"" \  "join like\_album la on \"user\".email = la.\"user\" " \  "join like\_group lg on \"user\".email = lg.\"user\" " \  "join like\_playlist lp on \"user\".email = lp.\"user\";"  query\_2 = "select \* " \  "from playlists\_table " \  "left join playlist p on playlists\_table.playlist = p.id " \  "join song s on playlists\_table.song = s.id " \  "where playlists\_table.playlist = 1813;"  query\_3 = "select public.album.id, name, year, prize " \  "from album " \  "right join prizes\_albums\_table pat on album.id = pat.album " \  "where public.album.type = 'concert';"  query\_4 = "select id, name, year, a.creator, gt.creator " \  "from song " \  "right join artists\_table a on song.id = a.song " \  "right join groups\_table gt on song.id = gt.song " \  "where song.year between 1970 and 2000;"  query\_5 = "select name, count(\*) as num\_of\_songs " \  "from artist " \  "right join artists\_table a on artist.name = a.creator " \  "group by name " \  "order by num\_of\_songs desc;"  query\_6 = "select email, name, count(\*) as num\_of\_like\_songs " \  "from \"user\" " \  "right join like\_song ls on \"user\".email = ls.\"user\"" \  " group by email, name " \  "having count(\*) > 1" \  " order by num\_of\_like\_songs desc;" |

### 4.3.2 Моделирование нагрузки на БД и снятие показателей

|  |
| --- |
| import random import threading import time  import matplotlib.pyplot as plt  from ConstantsQeuries import query\_1, query\_2, query\_3, query\_4, query\_5, query\_6, query\_list from Utils import open\_connection, drop\_index  connection, cursor = open\_connection()  curr\_res\_constant\_threads = [[]] curr\_res\_dynamic\_threads = [] cursor.execute("select \"user\" from like\_song;") like\_song\_email = cursor.fetchall() cursor.execute("select \"user\" from like\_album;") like\_album\_email = cursor.fetchall() cursor.execute("select \"user\" from like\_group;") like\_group\_email = cursor.fetchall() cursor.execute("select \"user\" from like\_playlist;") like\_playlist\_email = cursor.fetchall() cursor.execute("select song from playlists\_table;") playlist\_table\_songs = cursor.fetchall() cursor.execute("select album from prizes\_albums\_table;") prizes\_albums = cursor.fetchall() cursor.execute("select id from song;") songs\_ids = cursor.fetchall() cursor.execute("select name from \"group\";") group\_names = cursor.fetchall() cursor.execute("select song from artists\_table;") art\_songs = cursor.fetchall() cursor.execute("select song from groups\_table;") grp\_songs = cursor.fetchall() cursor.execute("select \"group\" from history\_artist\_table;") h\_group = cursor.fetchall() cursor.execute("select creator from artists\_table;") at\_creator = cursor.fetchall()  before = True   def main():  drop\_index()  check\_explain\_analyze()  # constant\_threads(1)  # constant\_threads(2)  # dinamic\_threads(200)  optimize()  check\_explain\_analyze()  # constant\_threads(1)  # constant\_threads(2)  dinamic\_threads(200)   def constant\_threads(num\_threads):  curr\_res\_constant\_threads.clear()  for t in range(num\_threads):  dbt = DBThread\_constant\_threads(t)  dbt.start()  while threading.activeCount() > 1:  time.sleep(1)  plot\_x = [k for k in range(401, 10000, 400)]  plot\_y = []  for i in range(len(curr\_res\_constant\_threads[0])):  curr\_sum = 0  for j in range(num\_threads):  curr\_sum += curr\_res\_constant\_threads[j][i]  plot\_y.append(curr\_sum / num\_threads)  plt.plot(plot\_x, plot\_y, linewidth=2.0)  plt.xlabel('Запросов в секунду')  plt.ylabel('Время ответа на один запрос, мс')  if before:  plt.title("Before. Num threads: {}".format(num\_threads))  else:  plt.title("After. Num threads: {}".format(num\_threads))  plt.show()   def dinamic\_threads(num\_querys):  plot\_x = [k for k in range(1, 31)]  plot\_y = []  for num\_threads in range(1, 31):  curr\_res\_dynamic\_threads.clear()  for t in range(num\_threads):  dbt = DBThread\_dynamic\_threads(num\_querys)  dbt.start()   while threading.activeCount() > 1:  time.sleep(1)   plot\_y.append(sum(curr\_res\_dynamic\_threads) / len(curr\_res\_dynamic\_threads))   plt.plot(plot\_x, plot\_y, linewidth=2.0)  plt.xlabel('Количество потоков')  plt.ylabel('Время ответа на один запрос, мс')  if before:  plt.title("Before. Dynamic thread num")  else:  plt.title("After. Dynamic thread num")  plt.show()   def exec\_query\_before(query, thread\_cursor):  if query == query\_1:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_1)  elif query == query\_2:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_2)  elif query == query\_3:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_3)  elif query == query\_4:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_4)  elif query == query\_5:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_5)  elif query == query\_6:  thread\_cursor.execute("EXPLAIN ANALYZE " + query\_6)  else:  print("Wrong query!")  fetch\_res = thread\_cursor.fetchall()  try:  return float(fetch\_res[-1][0].split(" ")[2]) + float(fetch\_res[-2][0].split(" ")[2])  except ValueError:  return float(fetch\_res[-1][0].split(" ")[2]) + float(fetch\_res[-2][0].split(" ")[4].split("=")[1]) + float(  fetch\_res[-3][0].split(" ")[2])   def check\_explain\_analyze():  cursor.execute("EXPLAIN ANALYZE " + query\_1)  print(cursor.fetchall())   class DBThread\_constant\_threads(threading.Thread):  def \_\_init\_\_(self, curr\_thread):   self.curr\_thread = curr\_thread  threading.Thread.\_\_init\_\_(self)  self.conn, self.cur = open\_connection()    def run(self):  for i in range(401, 10000, 400):  results = []  for j in range(0, i):  random\_query = random.choice(query\_list)  if before:  results.append(exec\_query\_before(random\_query, self.cur))  else:  results.append(exec\_query\_after(random\_query, self.cur))  if len(curr\_res\_constant\_threads) < self.curr\_thread + 1:  curr\_res\_constant\_threads.append([])  curr\_res\_constant\_threads[self.curr\_thread].append(sum(results) / len(results))  print(i)  self.conn.commit()   class DBThread\_dynamic\_threads(threading.Thread):  def \_\_init\_\_(self, num\_querys):   self.num\_querys = num\_querys  threading.Thread.\_\_init\_\_(self)  self.conn, self.cur = open\_connection()   def run(self):  results = []  for j in range(0, self.num\_querys + 1):  print(j)  random\_query = random.choice(query\_list)  if before:  results.append(exec\_query\_before(random\_query, self.cur))  else:  results.append(exec\_query\_after(random\_query, self.cur))  curr\_res\_dynamic\_threads.append(sum(results) / len(results))  self.conn.commit()   if \_\_name\_\_ == "\_\_main\_\_":  main() |

Функция constant\_threads и класс DBThread\_constant\_threads работают с постоянным количеством потоков. В constant\_threads создаётся нужное количество потоков, которые вызывают переменное количество раз (в моём случае от 401 до 10000 с шагом 200) случайные пользовательские запросы в функции класса DBThread\_constant\_threads. Аналогично функция dinamic\_threads и класс DBThread\_dinamic\_threads работают для изменяющегося количества потоков, только в этом случае работает постоянное количество вызовов запросов(в моём случае 2000). Также в функциях constant\_threads и dinamic\_threads строятся графики для отображения результатов. В первом случае на них показывается зависимость времени, потраченного на 1 запрос относительно количества запросов в секунду. Во втором случае – относительно числа потоков, так как число запросов постоянное.

### 4.3.3 Оптимизация

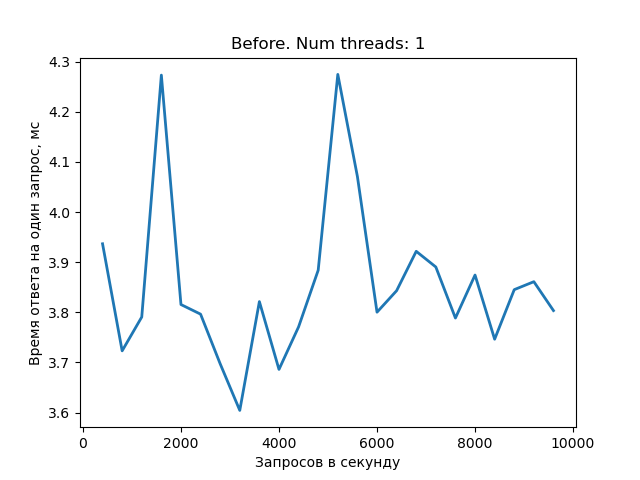
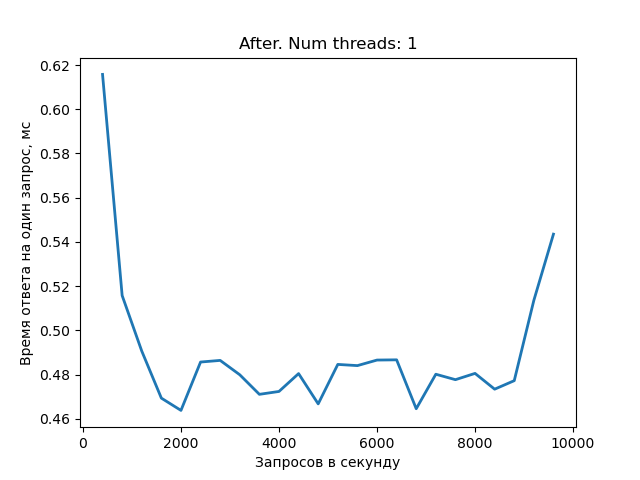
Для оптимизации запросов были выбраны 2 метода: CREATE INDEX ON И PREPARE. CREATE INDEX ON работает сразу для всех клиентов, поэтому этот вид оптимизации можно выделить в отдельную функцию, в то время как PREPARE нужно создавать отдельный для каждого клиента, поэтому он реализуется в классе потока.

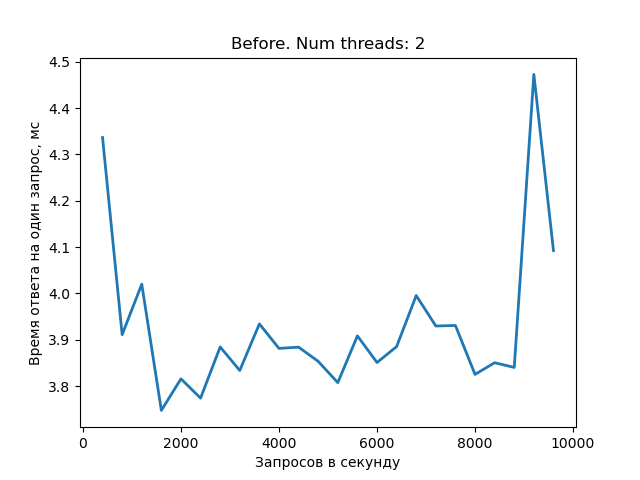
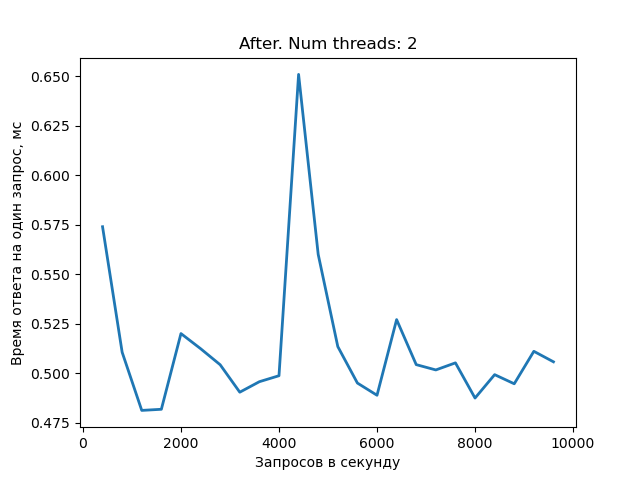
Листинг 12. Оптимизация запросов.

|  |
| --- |
| def optimize():  cursor.execute("create index us\_ind\_1 on like\_song(\"user\");")  cursor.execute("create index us\_ind\_2 on like\_album(\"user\");")  cursor.execute("create index us\_ind\_3 on like\_group(\"user\");")  cursor.execute("create index pl\_id on playlists\_table(playlist) where playlist=1813;")  cursor.execute("create index album\_type\_ind on album(type) where type = 'concert';")  cursor.execute("create index song\_id\_ind on song(year) where year between 1970 and 2000;commit;")  cursor.execute("create index user\_email\_ind on like\_song(\"user\") ;commit ;")  cursor.execute("create index artists\_id on artists\_table(creator);commit;")  self.cur.execute(  "prepare query1 (varchar) as select email, name,ls.song as like\_song\_id,la.album as "  "like\_album\_id,lg.\"group\" as like\_group\_name,lp.playlist as like\_playlit\_id from \"user\" join "  "like\_song ls on \"user\".email = $1 join like\_album la on \"user\".email = $2 join like\_group lg on "  "\"user\".email = $3 join like\_playlist lp on \"user\".email = $4;") self.cur.execute(  "prepare query2 as select \* from playlists\_table left join playlist p on playlists\_table.playlist = "  "p.id join song s on playlists\_table.song = $2 where playlists\_table.playlist = $1;") self.cur.execute(  "prepare query3 (int) as select public.album.id, name, year, prize from album right join "  "prizes\_albums\_table pat on album.id = $1 where public.album.type = 'concert';") self.cur.execute(  "prepare query4 (int) as select id, name, year, a.creator, gt.creator from song right join "  "artists\_table a on song.id = $1 right join groups\_table gt on song.id = gt.song where song.year "  "between 1970 and 2000;") self.cur.execute(  "prepare query5(varchar) as select name, count(\*) as num\_of\_songs from artist right join "  "artists\_table a on artist.name = $1 group by name order by num\_of\_songs desc;") self.cur.execute(  "prepare query6 (varchar) as select email, name, count(\*) as num\_of\_like\_songs from \"user\" right "  "join like\_song ls on \"user\".email = $1 group by email, name having count(\*) > 1 order by "  "num\_of\_like\_songs desc;") |

## 4.4 Сравнительный анализ результатов

### 4.4.1 Анализ графических результатов

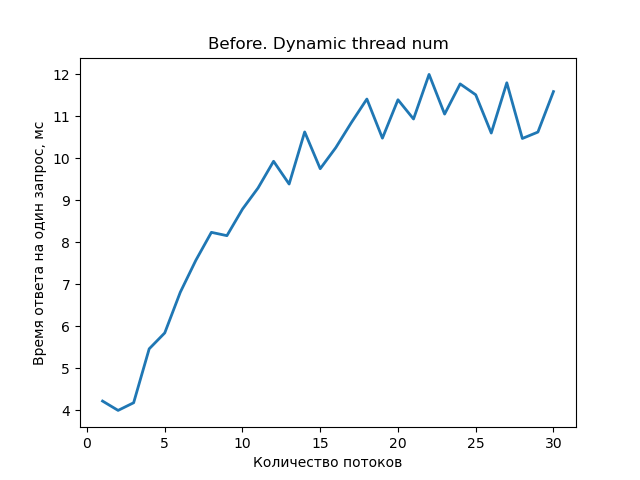
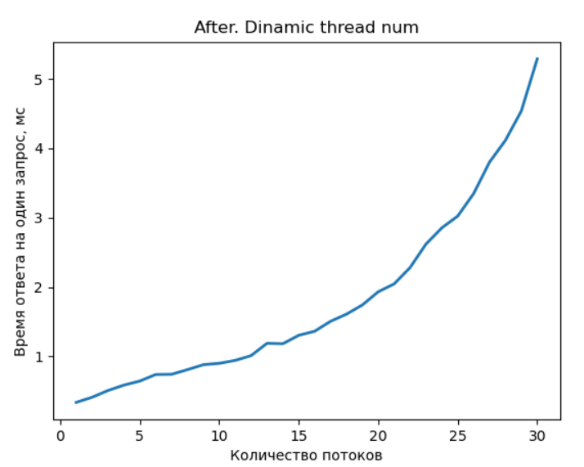
 

Рис. 13. Результаты запросов до и после оптимизации.

При работе 1 потока и 2 потоков заметно, что результат сильно улучшился при добавлении оптимизации. То же самое видно и при динамическом увеличении количества потоков.

## 4.5 Вывод

В данной лабораторной работе я научился оптимизировать запросы. Самым сложным было анализировать результаты explain. Лучшим видом оптимизации оказался index, так как при его применении результат виден сильнее, и он применим к конкретным столбцам таблицы, что помогает ускорить поиск по ней. Так же анализ графических результатов показал, что при использовании изменяющего количества потоков для оценки оптимизации результат виден сильнее.