Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Базы данных

Отчет по лабораторным работам

Работу выполнила:
Карпушкина О.А.
Группа:
3530901/70202
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2021

Содержание

1. Лабораторная работа №1. Проектирование БД.

1.1. Разработка структуры БД

Цель работы

Познакомиться с основами проектирования схемы БД, способами организации данных в SQL-БД.

В качестве задания была выбрана тема "Магазин музыки". Основной задачей при выборе такой предметной области является хранение информации, связанной с исполнителями, альбомами, музыкальными композициями (треками), звукозаписывающими лейблами и некоторой дополнительной информации (жанры, страны).

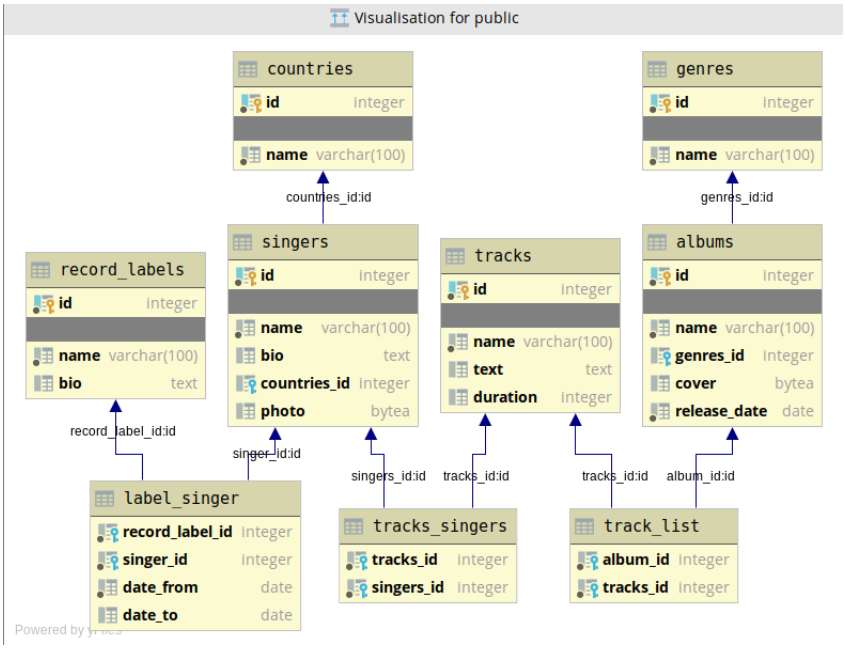

Рис. 1.1. Графическая схема базы данных по теме "Магазин музыки"

Графическая схема приведена на Рис.1.1. Полное описание всех таблиц и атрибутов приведено в разделе Wiki проекта `http://gitlab.icc.spbstu.ru/aoame/db_course_2020`

1.2. Язык SQL-DDL

1.2.1. Цель работы

Познакомиться с основами проектирования схемы БД, языком описания сущностей и ограничений БД SQL-DDL.

1.2.2. Программа работы

1. Самостоятельное изучение SQL-DDL.
2. Создание скрипта БД в соответствии с согласованной схемой. Должны присутствовать первичные и внешние ключи, ограничения на диапазоны значений. Демонстрация скрипта преподавателю.
3. Создание скрипта, заполняющего все таблицы БД данными.

4. Выполнение SQL-запросов, изменяющих схему созданной БД по заданию преподавателя. Демонстрация их работы преподавателю.

Полный скрипт для создания БД находится в репозитории с основным проектом SQL scripts/creation.sql

1.2.3. Ход работы

Выполнение лабораторной работы продолжается изучением языка SQL, а если более конкретно – SQL-DDL.

Создание базы данный начинается с создания таблиц. В качестве примера рассмотрим создание таблицы альбомов. В этой таблице должна храниться необходимая информация об альбомах – название, жанр, обложка, дата релиза.

```
1 create table albums
2 (
3     id           serial primary key,
4     name         varchar(100) not null,
5     genres_id    int,
6     cover        bytea,
7     release_date date         not null
8
9 );
```

Чтобы была возможность ссылаться на записи в этой таблице зададим обязательный целочисленный автоинкрементный уникальный идентификатор – id. Для обеспечения перечисленных свойств очень хорошо походит тип serial, а чтобы обеспечить уникальность и обязательность добавим ключевое слово primary key.

Следующим обязательным атрибутом будет являться название альбома – name. Необходимость заполнение указываем с помощью ключевого слова not null, а тип – varchar(n), который обеспечит хранение строки ограниченной переменной длины. Ещё одним обязательным атрибутом является дата релиза – release_date с типом date и ключевым словом not null.

Необязательными атрибутами для данной таблицы будут являться обложка альбома и идентификатор жанра. Идентификатора жанра – genre_id – будет ссылаться на одну запись в справочной ташблице, содержащей список жанров. По этой причине тип данного атрибута – int. Атрибут cover может использоваться для хранения обложки, тип – bytea

Также при создании таблиц можно указывать проверки, чтобы не допускать записи некорректных данных в БД. Такая проверка была использована при создании таблицы треков. Чтобы ошибочно не произошло записи отрицательного числа стоит проверка на $duration > 0$

```
1 create table tracks
2 (
3     id       serial primary key,
4     name     varchar(100) not null,
5     text     text,
6     duration int check ( duration > 0 ) not null
7
8 );
```

Теперь чтобы такие поля как *genres_id* ссылались на записи в других таблицах нужно организовать связь с помощью внешнего ключа. Это можно делать несколькими способами.

Можно явно указать связь при создании таблицы с помощью *references*, как это показано на листинге ниже. После ключевого слова указывается таблица, на которую ссылается внешний ключ, а в скобках – имя атрибута.

```
create table label_singer
(
    record_label_id int  not null
        references record_labels (id) on delete restrict,
    singer_id       int  not null
        references singers (id) on delete restrict,
    date_from       date not null,
    date_to         date

);
```

Если при создании таблицы ограничение не было указано, можно при помощи альтернативной таблицы его добавить, как на примере ниже. Также в конце можно указать действия при удалении записи, на которую ссылается внешний ключ. В приведенном примере удаление записей из таблиц *albums* и *tracks* будет запрещено, если в таблице *track_list* будет находиться запись, содержащая ссылки на какие-либо записи в этих таблицах

```
alter table track_list
    add constraint albums_id_fk foreign key (album_id) references albums (id)
        on delete restrict,
    add constraint tracks_id_fk foreign key (tracks_id) references tracks (id)
        on delete restrict;
```

1.3. Индивидуальное задание

Задание

Добавить жанры для треков и убрать их у альбомов. Жанр альбома будет собираться из комбинации жанров всех треков, входящих в него.

Также необходимо сохранить уже имеющиеся данные в БД после реорганизации структуры: для каждого трека проставить жанр альбома, в котором он находится на тот момент.

Решение

Для начала необходимо добавить в таблицу с треками новый атрибут – жанр. Также необходимо добавить внешний ключ, чтобы организовать связь с таблицей жанров.

```
alter table tracks
    add column genres_id integer;

alter table tracks
    add constraint genre_id_fk foreign key (genres_id) references genres (id)
    on delete set null;
```

Теперь необходимо обновить каждую запись в таблице треков, проставив жанр, который соответствует альбому. Эта связь организована в таблице *track_list*. Информацию о жанрах нужно взять из таблицы альбомов, но поскольку нам нужны только те записи, которые есть в *track_list* необходимо сделать *join* по идентификатору альбома. Теперь необходимо обновить поле жанров в таблице *tracks* по таблице *albums* – $setgenres\_id = albums.genres\_id$, но сделать это только в тех записях, где атрибут *tracks.id* совпадает с $track_list.tracks_id$. Полный код запроса приведен ниже.

```
update tracks
set genres_id = albums.genres_id
from albums
         join track_list on albums.id = track_list.album_id
where tracks.id = track_list.tracks_id
```

Финальным этапом нужно удалить атрибут *albums.genres_id*.

```
alter table albums
    drop column genres_id;
```

1.4. Выводы

В ходе выполнения данной лабораторной работы проведено ознакомление с основами проектирование схемы базы данных. Были изучены основы создания скриптов на языке SQL. С помощью SQL-DDL описаны структуры разрабатываемой схемы БД. Было проведено знакомство с первичными и внешними ключами.

2. Лабораторная работа №2. Генерация тестовых данных.

2.1. Цель работы

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

2.2. Программа работы

1. Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
2. Частные требования к генератору, набору данных и результирующему набору данных:
   (a) количество записей в справочных таблицах должно соответствовать ограничениям предметной области
   (b) количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
   (c) значения для внешних ключей необходимо брать из связанных таблиц
   (d) сохранение уже имеющихся данных в базе данных

2.3. Ход работы

В качестве языка для написания генератора данных был выбран Python, библиотека, обеспечивающая подключение к БД, – psycopg2. Для генерации данных (имен, названий, дат) использовалась библиотека mimesis. Для того, чтобы не передавать объект подключения всем функциям заполнения, но все равно заносить все сделанные изменения в таблицу была использована функция autocommit $con.autocommit = True$

Алгоритм работы скрипта генерации

1. Запрос на ввод параметра для генерации данных;
2. Подключение к базе данных;
3. Объявление объекта *cursor* для выполнения запросов и получения их результатов;
4. Заполнение генерационными данными таблицы;
   (a) Получение уже имеющихся записей в заполняемой таблице (только значимые атрибуты);
   (b) Генерация и заполнение таблицы с учетом результатов предыдущего пункта
   (c) При дальнейшем использовании генерированных данных для заполнения внешних ключей других таблиц производится операция $select\ id\ from\ name\_table$ и результат выполнения возвращается.
5. Повторение п.4 для все таблиц БД;
6. Закрытие курсора и закрытие соединения с БД.

Подключение к базе данных

Для подлкючения к БД использовалась функция *psycopg2.connect*, которая принимает на вход параметры: хост, имя базы данных, имя пользователя, пароль. Хранение перечисленных параметров было организовано в отдельном файле, и при запуске программы происходило его чтение в словарь, где ключом являлось название параметра.

```
conn_params = dict()

    for r in f:
        pair_dict = r.split('\n')[0]
        conn_params[pair_dict.split(':')[0]] = pair_dict.split(':')[1]


    # connect to the db

    con = psycopg2.connect(
        host=conn_params['host'],
        database=conn_params['database'],
        user=conn_params['user'],
        password=conn_params['password']
    )
    con.autocommit = True
    cursor = con.cursor()
```

Заполнение таблицы генерационными данными

В разработанной БД таблицы можно условно разделить на 3 группы: справочные, основные и для организации связи "многие ко многим". Рассмотрим процесс заполнения для каждой из них.

Самыми первыми заполняются справочные таблицы, т.к. они не содержат в себе никаких внешних ключей. Также на них не действует введенный параметр. При первом вызове скрипта заполнения таблицы, содержащие список стран и жанров будут заполнены всеми имеющимися данными.

Для заполнения таблицы со странами используется csv-файл со всеми странами, из которого берутся только названия стран.

Для заполнения таблицы жанров использовался скрипт, который берет все имеющиеся жанры из википедии и сохраняет их в текстовый файл. При заполнении все данные берутся из этого файла и заносятся в таблицу.

Рассмотрим заполнение таблицы стран. В main-потоке за это отвечают строки кода, приведенные ниже

```
get_countries()
table_countries = insert.insert_countries(cursor, table_countries)
```

Метод *get_countries*() получает уже имеющиеся записи в таблице.

```
def get_countries():
    global table_countries
    cursor.execute("select␣name␣from␣countries")
    table_countries = set(cursor.fetchall())
```

После этого метод *insert_countries* парсит csv-файл, создает добавляемое множество, убирает из него уже имеющиеся записи и производит добавление в БД. В дальнейшем id добавленных записей пригодятся для генерации данных для других таблицы, поэтому сразу из получим и возвратим для хранения в *table_countries*.

```
def insert_countries(cursor, table_countries):
        countries = pd.read_csv('data/world_2.csv', sep=',')
        add_set = set((x,) for x in countries.name)
        add_set = add_set - table_countries

        sql = 'insert into countries (name) values (%s)'

        psycopg2.extras.execute_batch(cursor, sql, list(add_set))

        cursor.execute("select id from countries")
        table_countries = cursor.fetchall()

        return table_countries
```

Важно отметить, что for-циклы в Python очень долгие и лучше их не использовать, если есть уже готовые решения. Поэтому для исполнения большого количества запросов во всех методах добавления используется $psycopg2.extras.execute\_batch(cursor, sql, list(add\_set))$. Эта строчка позволяет выполнить один и тот же запрос для всех параметров, хранящихся в передаваемом листе, но более оптимизировано, чем при использовании for-циклов.

Теперь рассмотрим заполнение таблицы с исполнителями. Аналогичными образом заполняются таблицы *record_labels*, *albums*, *tracks*. Во-первых, здесь при попытке избежать повторения записей при генерации мне кажется неправильным использовать только имена/названия, т.к., например, исполнители могут иметь одинаковое имя/псевдоним, находясь в разных странах. В одной стране встретить исполнителей с одинаковым именем/псевдонимом уже менее вероятно. Во-вторых, для заполнения атрибута *singers.name* используются сгенерированные данные из библиотеки mimesis. Для разнообразия имен были выбраны три паттера генерации (пример будет в коде). И в-третьих, из-за специфики исполнения запроса *insert* (в качестве параметра могут быть переданы только tuple-объекты) методы заполнения содержат дополнительный for-цикл для формирования списка tuple-объектов для добавления. Пример всего вышеописанного приведен во фрагменте кода ниже. Идентификатор страны выбирается случайно из таблицы стран, которая ранее была заполнена.

```
    def insert_singers(self, cursor, table_singers, table_countries):
        add_set = set()
        add_list = []
        for i in range(self.parameter):
            a = random.randint(0, 2)
            if a == 0:
                name = g._person().name() + " " + g._person().surname()
            elif a == 1:
                name = g.text.word().capitalize()
            else:
                name = g.text.word().capitalize() + " " + g.text.word().
                capitalize()
            country_id = table_countries[random.randint(0,
                                        len(table_countries) - 1)]
            add_set.add((name, country_id))

        add_set = add_set - table_singers
        for x in add_set:
            a = random.randint(0, 2)
            if a > 1:
                bio = None
                photo = g.person.avatar()
            else:
                bio = g.text.text(random.randint(1, 3))
```

```
25                  photo = None
26              add_list.append((x[0], bio, x[1], photo))
27
28          sql = 'insert into singers (name, bio, countries_id, photo) ' \
29                'values (%s, %s, %s, %s)'
30
31          psycopg2.extras.execute_batch(cursor, sql, add_list)
32          cursor.execute("select id from singers")
33          table_singers = cursor.fetchall()
34          return table_singers
```

Финальным этапом рассмотрим заполнение таблицы связи "многие-ко-многим". У каждой таблицы есть свои особенности, которые необходимо учесть.

- При заполнении таблицы *track_list* для каждой записи в таблице альбомов будет сгенерировано случайное количество (из некоторого диапазона; например, от 1 до 30) записей со случайным идентификатором трека.
- При заполнении таблицы *track_singer* будет сгенерировано в 2 раза больше записей, чем было задано пользователем, т.к. таким образом я хочу добиться использования каждого идентификатора больше одного раза
- При заполнении *label_singer* точно так же используется увеличенное в 2 раза значение параметра, а также в таблице имеются атрибуты для хранения даты, которые необходимо учесть при анализе имеющихся записей

Перечисленные выше особенности не показались мне сложными в воплощении, поэтому не считаю необходимым расписывать их в отчете. В качестве примера ниже код для заполнения таблицы *track_list*.

```
1     def insert_track_list(cursor, table_tracks, table_albums,
2                                                 table_track_list):
3         add_set = set()
4         for id in table_albums:
5             for i in range(1, 30):
6                 t = table_tracks[random.randint(0, len(table_tracks) - 1)]
7                 add_set.add((id, t))
8
9         add_set = add_set - table_track_list
10        sql = 'insert into track_list (album_id, tracks_id) values (%s, %s)'
11        psycopg2.extras.execute_batch(cursor, sql, list(add_set))
```

2.4. Выводы

В ходе выполнения данной лабораторной работы был реализован генератор в виде скрипта на языке Python, который позволяет сформировать набор связанных данных в каждой таблице. Были получены практические навыки взаимодействия с базой данных с использованием фреймворка psycopg2.

3. Лабораторная работа №3. Язык SQL-DML.

3.1. Цель работы

Познакомиться с языком создания запросов управления данными SQL-DML.

3.2. Программа работы

1. Изучение SQL-DML.
2. Выполнение всех запросов из списка стандартных запросов.
3. Выполнение индивидуального задания.

3.3. Ход работы

Первый пунктом работы было написание стандартных запросов в различных вариация с использованием SELECT, INSERT, DELETE, UPDATE. Файл со всеми стандартными запросами `http://gitlab.icc.spbstu.ru/aoame/db_course_2020/blob/master/SQL%20scripts/lab3/SQLdml_queries.sql`

SELECT

Оператор используется для извлечения записей из таблиц. С помощью различных конструкций можно задать различные условия выбора. На листинге ниже приведены некоторые варианты запросов с использование SELECT:

1. использование операции *like*;
2. вложенный запрос с вычисляемым полем;
3. вычисление в одном запросе нескольких совокупных характеристик;
4. использование JOIN; выборка с учетом ограничения на вычисляемое поле.

```
1  select name
2  from albums
3  where name like '%OO%';
4
5  select *
6  from tracks
7  where duration > (select avg(duration) from tracks)
8  order by duration desc;
9
10 select min(date_from), max(date_to)
11 from label_singer;
12
13 select *
14 from genres
15          left join tracks t on genres.id = t.genres_id
16 order by genres.id;
17
18
19 select tracks_id, count(tracks_id) as sum_singers
20 from tracks_singers
21 group by tracks_id
22 having count(tracks_id) > 4;
```

INSERT

Оператор используется для добавления записей в таблицы. Пример использования оператора приведен на листинге ниже для добавления записи с использование вложенного запроса.

```
1 insert into tracks_singers
2 values ( (select id from tracks where name = 'test track')
3        , (select id from singers where name = 'Test Singer'));
```

DELETE

Используется для удаление одной или нескольких записей из таблицы. Пример, указанный ниже, используется для удаления альбомов, для которых не указаны треки.

```
1
2 delete
3 from albums
4 where id not in (select album_id from track_list group by album_id);
```

UPDATE

Оператор используется для модификации существующих записей. В примере ниже для всех исполнителей, удовлетворяющих условию $where countries\_id = (select id from countries where countries.name = 'Canada')$, ставится новое значение атрибута *countries_id*

```
1 update singers
2 set countries_id = (select id from countries
3                     where countries.name = 'Test country')
4 where countries_id = (select id from countries
5                       where countries.name = 'Canada');
```

3.3.1. Индивидуальное задание

1. Для каждого лейбла вывести исполнителей, которые стали наименее интересны лейблу для сотрудничества. Интерес перестают представлять исполнители, которые в течение последнего года выпустили менее 5 новых треков. Дату выпуска трека оценить по дате выпуска самого первого альбома среди тех, в которые этот трек входит.;

2. Вывести артистов, которые теряют продуктивность. Таковыми считать тех, кто за последние 3 года с каждым годом выпускает меньше треков, чем в предыдущем. Оценку даты выпуска трека производить аналогично п.1.;

3. Для каждого исполнителя вывести его жанр на основании наиболее часто встречающемся жанре его треков.;

Задание 1 Первым этапом в составлении этого запроса является сопоставление треков с датой их релиза.

```
1 select album_id, tracks_id, release_date
2 from track_list
3         join tracks t on track_list.tracks_id = t.id
4         join albums a on track_list.album_id = a.id;
```

Как указано в задании, необходимо для каждого трека найти самый самый ранний альбом. Дата релиза этого альбома и будет являться датой релиза трека. Для этого для каждый пары альбом-трек находится минимальная дата и сопоставляется паре исполнитель-трек

```
with t1 as (select album_id, tracks_id, release_date
            from track_list
                     join tracks t on track_list.tracks_id = t.id
                     join albums a on track_list.album_id = a.id)

select s.id as sid,
       d.tracks_id as tid,
       min(extract(year from d.release_date))
from tracks_singers
         join singers s on tracks_singers.singers_id = s.id
         join t1 d on tracks_singers.tracks_id = d.tracks_id
group by d.tracks_id, s.id
order by s.id, min(extract(year from d.release_date)) desc, d.tracks_id
```

Финальным шагом является сопоставление лейблов и артистов. Обязательными условиями является продолжающееся по н.в. сотрудничество, выпуск трека после начала сотрудничества и выпуск трека за прошедший год (указанный).

```
explain analyze
with t1 as (select album_id, tracks_id, release_date
            from track_list
                     join tracks t on track_list.tracks_id = t.id
                     join albums a on track_list.album_id = a.id),
     t2 as (select s.id as sid, d.tracks_id as tid, min(d.release_date) as rdate
            from tracks_singers
                     join singers s on tracks_singers.singers_id = s.id
                     join t1 d on tracks_singers.tracks_id = d.tracks_id
            group by d.tracks_id, s.id)

select record_label_id as rid,
       sid,
       count(*)        as total_tracks
from label_singer
         join t2 iim on label_singer.singer_id = iim.sid
where date_to IS NULL
  and iim.rdate > date_from
  and extract(year from iim.rdate) = 2020
group by rid, sid
having count(*) < 5
order by total_tracks desc, sid;
```

Задание 2 В этом задании, аналогично предыдущему, первым делом находим для трека его дату релиза и сопоставляем результаты с исполнителями. После этого полученный результат группируем по годам, чтобы можно было посчитать кол-во треков в год. в примере ниже подсчеты проводятся для 2017 года. Важный момент в этом шаге состоит в том, что необходимо учитывать нулевое кол-во треков за год. Агрегатная функция *count* вместо нулевого значения ставит *null*. Эту ситуацию исправляет $coalesce(count(*), 0)$.

```
with t1 as (select album_id,
                   tracks_id,
                   release_date
            from track_list
                     join tracks t on track_list.tracks_id = t.id
                     join albums a on track_list.album_id = a.id
```

```
),
     t2 as (select t.id                                        as tid,
                   s.id                                        as sid,
                   extract(year from min(d.release_date)) as rdate
            from tracks_singers
                     join singers s on tracks_singers.singers_id = s.id
                     join tracks t on tracks_singers.tracks_id = t.id
                     join t1 d
                          on tracks_singers.tracks_id = d.tracks_id
            group by t.id, s.id)

select sid, coalesce(count(*), 0) as t17
from t2
where rdate = '2017'
group by sid
```

Такие конструкции используются для подсчета выпущенных треков для каждого года. После этого составляется выборка из таблиц, где каждому исполнителю соответствует значения 4 атрибутов, содержащих информацию о кол-ве выпущенных треков. На последнем шаге из нее выбираются только те строки, в который есть постоянная тенденция на уменьшение (в каждом след. году треков меньше, чем в предыдущем). Итоговый запрос приведен ниже в листинге.

```
with t1 as (select album_id,
                   tracks_id,
                   release_date
            from track_list
                     join tracks t on track_list.tracks_id = t.id
                     join albums a on track_list.album_id = a.id
),
     t2 as (select t.id                                        as tid,
                   s.id                                        as sid,
                   extract(year from min(d.release_date)) as rdate
            from tracks_singers
                     join singers s on tracks_singers.singers_id = s.id
                     join tracks t on tracks_singers.tracks_id = t.id
                     join t1 d
                          on tracks_singers.tracks_id = d.tracks_id
            group by t.id, s.id),
     t3 as (select sid, count(*) as t17
            from t2
            where rdate = extract(year from current_date) - 3 - 1
            group by sid),
     t4 as (select sid, count(*) as t18
            from t2
            where rdate = extract(year from current_date) - 2 - 1
            group by sid),
     t5 as (select sid, count(*) as t19
            from t2
            where rdate = extract(year from current_date) - 1 - 1
            group by sid),
     t6 as (select sid, count(*) as t20
            from t2
            where rdate = extract(year from current_date)
            group by sid),
     t7 as (select singers.id       as sid,
                   coalesce(t17, 0) as t17,
                   coalesce(t18, 0) as t18,
                   coalesce(t19, 0) as t19,
                   coalesce(t20, 0) as t20
```

```
38              from singers
39                      left join t3 on singers.id = t3.sid
40                      left join t4 on singers.id = t4.sid
41                      left join t5 on singers.id = t5.sid
42                      left join t6 on singers.id = t6.sid),
43      t8 as (select sid, (t18 >= t17 and t19 >= t18 and t20 >= t19) as answ
44              from t7
45              where (t18 >= t17 and t19 >= t18 and t20 >= t19) is False)
46
47
48 select ls.record_label_id as rid, t8.sid
49 from t8
50          join label_singer ls on t8.sid = ls.singer_id
51 where date_to is null
52 order by rid, t8.sid
```

Задание 3 Сначала составим выборку комбинаций исполнитель-трек-жанр и посчитает сколько таких комбинаций имеется. После этого найдем максимальное значение треков жанра для каждого исполнителя. Финальным этапом будет являться выборка таких комбинаций из первого шага, в который кол-во треков определенной жанра соответствует максимальному значению для исполнителя. Листинг запроса приведен ниже

```
1  with t1 as (select s.id as sid, t.genres_id as gid, count(*) as nums
2              from tracks_singers
3                      join singers s on tracks_singers.singers_id = s.id
4                      join tracks t on tracks_singers.tracks_id = t.id
5              group by sid, gid),
6      t2 as (select t1.sid, max(t1.nums) as maxnums
7              from t1
8              group by t1.sid)
9
10 select t1.sid, t1.gid as gid, t1.nums, t2.maxnums
11 from t1
12          join t2 on t1.sid = t2.sid
13 where t1.nums = t2.maxnums
14
15 order by sid, nums desc, gid;
```

EXPLAIN ANALYZE Используя оператор explain перед запросом будет выведен план запроса. Для запроса, его подзапросов, присоединений можно узнать стоимость (общая стоимость вычисляется как

(число_чтений_диска * seq_page_cost)
+ (число_просканированных_строк * cpu_tuple_cost).

По умолчанию, seq_page_cost равно 1.0, а cpu_tuple_cost — 0.01,), методы доступа к записям, вид присоединения, методы сортировки и т.д. При добавлении к оператору $ANALYZE$ в конец плана запроса будет добавлено время планирования и время исполнения.

Для запроса из первого пункта везде используется последовательный доступ к записям, все присоединения выполняются с помощью Hash Join, для сортировки используются quicksort, external merge, время планирования = 0.598 ms,время исполнения = 533.745 ms

Для запроса из второго пункта используется последовательный доступ к записям, CTE Scan, сканирование индексов (primary key),присоединения выполняются с помощью Hash Join, Merge Left Join, для сортировки используются quicksort, external merge, время планирования = 1.304 ms,время исполнения = 661.639 ms

Для запроса из третьего пункта используется последовательный доступ к записям, CTE Scan,присоединения выполняются с помощью Hash Join, для сортировки используются external merge, время планирования = 1.622 ms,время исполнения = 260.539 ms

3.4. Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление с языком создания запросов и управления данными SQL-DML. Была проведена работа с выборкой данных, их вставкой, удалением и модификацией. Проделаны все стандартные запросы и выполнены индивидуальные задания.

4. Лабораторная работа №4. Нагрузка базы данных и оптимизация запросов

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

1. Написание параметризированных типовых запросов пользователей.
2. Моделирование нагрузки базы данных.
3. Снятие показателей работы сервиса и построение соответствующих графиков.
4. Применение возможных оптимизаций запросов и повторное снятие показателей.
5. Сравнительный анализ результатов
6. Демонстрация результатов преподавателю

4.3. Ход работы

Программа была реализована на ЯП Python. Для взаимодействий с базой данных использовалась библиотека psycopg2.

Оптимизация проводилась с помощью **PREPARE** и **CREATE INDEX**, а анализ – с помощью **EXPLAIN ANALYSE**

PREPARE – создаёт подготовленный оператор. Подготовленный оператор представляет собой объект на стороне сервера, позволяющий оптимизировать производительность приложений. Когда выполняется PREPARE, указанный оператор разбирается, анализируется и переписывается. При последующем выполнении команды EXECUTE подготовленный оператор планируется и исполняется. Такое разделение труда исключает повторный разбор запроса, при этом позволяет выбрать наилучший план выполнения в зависимости от определённых значений параметров. Может принимать на вход параметры.

CREATE INDEX – оздает индексы по указанному столбцу. Данный способ может очень сильно ускорить работу, так как в будущем БД будет знать, что на данный столбец создан индекс и начнет поиск сначала по индексу, начиная с корня и спускаясь по узлам до тех пор, пока не найдет искомое значение. В итоге результат может быть найден быстро.

ANALYSE – инструмент анализа запросов. Показывает оценку стоимости выполнения данного узла, которую сделал для него планировщик. Это значение он старается минимизировать. В выводе мы имеем такие показатели, как стоимость до вывода данных и общая стоимость, число строк (до конца) и размер строк в байтах. Точность оценок планировщика можно проверить, используя команду EXPLAIN ANALYSE. С этим параметром EXPLAIN на самом деле выполнит запрос и нам становится доступна дополнительная статистика. Можно увидеть подробный разбор каждого узла выполнения запроса, ключ сортировки, метод сортировки, метод сканирования и другое. Выбирать строки по отдельности дороже, чем читать последовательно. Но если читать нужно не все страницы

таблицы, то выбирать дешевле. Добавление условия уменьшает оценку числа результирующих строк, но не стоимость запроса, так как просматриваться будет тот же набор строк, что и раньше. Стоимость даже может увеличиться. Если таблица слишком маленькая для сканирования по id, происходит последовательное сканирование.

Запросы:

```
1  -- 1
2  explain analyse
3  select t.name,
4         albums.name,
5         albums.release_date
6  from albums
7           join track_list tl on albums.id = tl.album_id
8           join tracks t on t.id = tl.tracks_id
9  where release_date between '2018-01-01' and '2020-01-01'
10 order by albums.name, albums.release_date desc;
11
12 -- 2
13 explain analyse
14 select singers.name as singer_name
15 from singers
16          join countries c on singers.countries_id = c.id
17 where c.name = 'Russia';
18
19
20 -- 3
21 explain analyse
22 select rl.name, s.name, date_from
23 from label_singer
24          join record_labels rl on label_singer.record_label_id = rl.id
25          join singers s on label_singer.singer_id = s.id
26 where date_to is null
27 order by rl.name, s.name;
28
29 -- 4
30 explain analyse
31 with t1 as (select s.id as sid, t.genres_id as gid, count(*) as nums
32              from tracks_singers
33                       join singers s on tracks_singers.singers_id = s.id
34                       join tracks t on tracks_singers.tracks_id = t.id
35              group by sid, gid),
36      t2 as (select t1.sid, max(t1.nums) as maxnums from t1 group by t1.sid),
37      t3 as (select t1.sid, t1.gid as gid
38              from t1
39                       join t2 on t1.sid = t2.sid
40              where t1.nums = t2.maxnums)
41 select g.name, s2.name
42 from t3
43          join genres g on t3.gid = g.id
44          join singers s2 on t3.sid = s2.id
45 order by g.name, s2.name;
46
47 -- 5
48 explain analyse
49 select a.name, t.name
50 from track_list
51          join albums a on track_list.album_id = a.id
52          join tracks t on track_list.tracks_id = t.id
53 where a.name = 'Visions';
```

Количество строк в таблицах было более 10000 (кроме таблиц, содержащих информацию о странах и жанрах).

Количество отправляемых запросов было параметризовано таким образом, что с каждым новым экспериментом оно возрастало экспоненциально. Количество потоков (подключений к БД) также было параметризовано.

Схема проводимых экспериментов:

1. Ввод параметров эксперимента: количество потоков-подключений, максимальное количество запросов в формате 2^n, где n – вводимый параметр;
2. Удаление индексов на случай, если во время каких-то других экспериментов с БД они были добавлены;
3. Старт эксперимента с заданным количеством потоков и снятие показателей и запись в лог;
4. Добавление индексов;
5. Повторение эксперимента;
6. Удаление индексов;
7. Старт экспериментов с использованием **PREPARE**.
8. Добавление индексов;
9. Старт эксперимента с полной оптимизацией.

Схема эксперимента:

1. Запуск эксперимента с 1 запросом для заданного кол-ва подключений.
2. Снятие показателей и запись в лог;
3. Увеличение количества запросов $n = n + 1$
4. Эксперимент с количеством запросов 2^n, если максимальное количество запросов еще не достигнуто, то возврат к пункту 2, иначе – завершение эксперимента.

Ниже (рис. 4.1) приведены результаты эксперимента при запуске 7 потоков, которые отправляли до 16 запросов каждый.

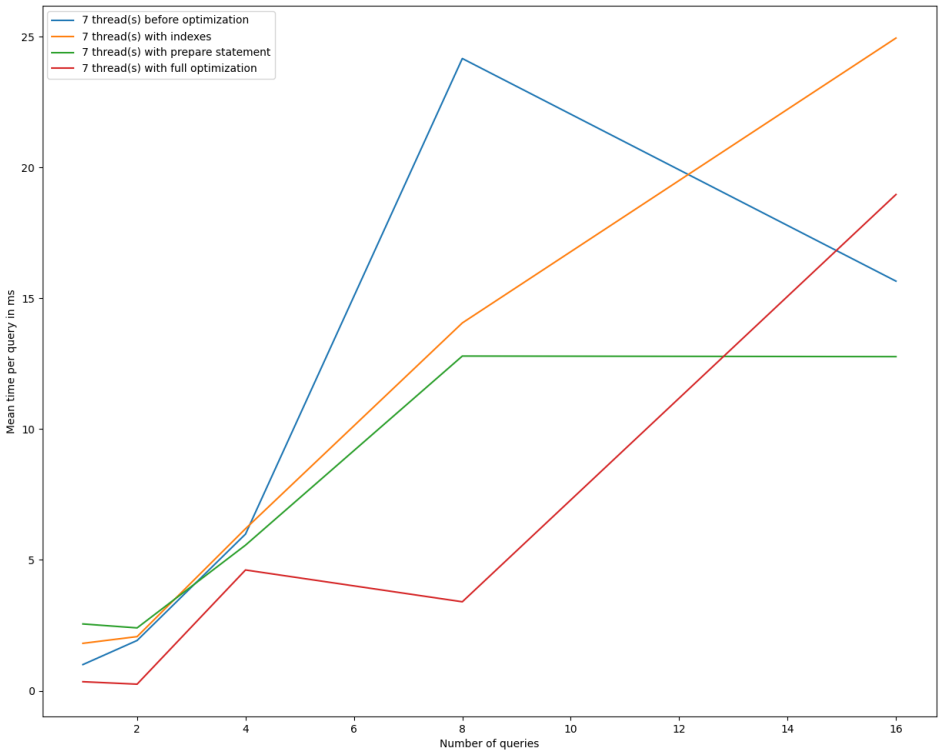

Рис. 4.1. Эксперимент для 7 потоков при отправке до 16 запросов

Можно увидеть, что в данном случае полная оптимизация показала себя лучше других способов оптимизации.

На рис. 4.2 можно увидеть, что использование индексов показало себя выгоднее всего.

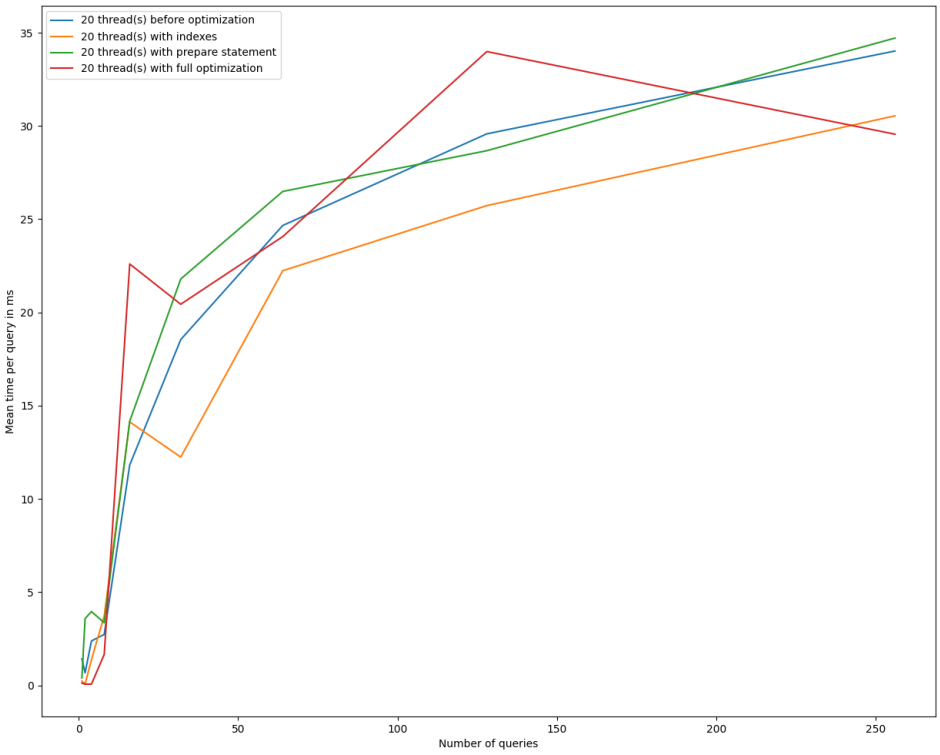

Рис. 4.2. Эксперимент для 20 потоков при отправке до 256 запросов

Рассмотрим зависимость времени выполнения от количества запросов на рис. 4.3. Для работы без оптимизации среднее время было примерно одинаковым на протяжении всех экспериментов. Опция полной оптимизации улучшила показатели при <4 потоков и при >7 потоков по сравнению с безоптимизационным вариантом. Аналогичная картина наблюдается при использовании индексов. А вот вариант использования PREPARE на большей части экспериментов показал себя менее эффективно, чем безоптимизационный вариант.

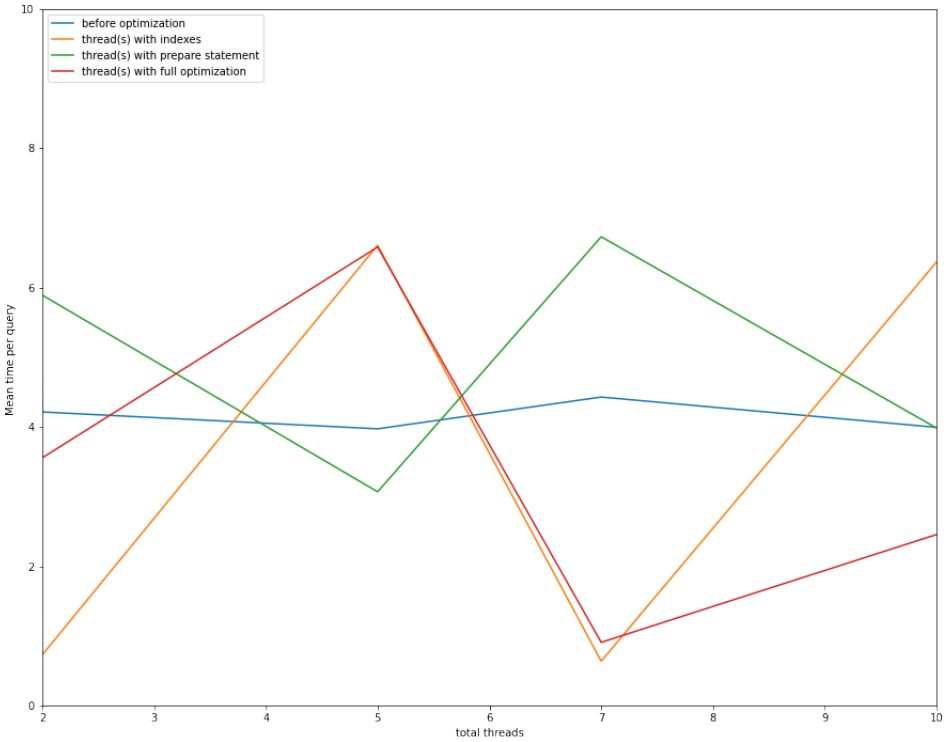

Рис. 4.3. Зависимость среднего времени выполнения запроса от количества потоков

4.4. Выводы

В ходе выполнения данной лабораторной работы было проведено ознакомление со структурой и инструментами, использующимися при оптимизации запросов пользователей к БД. Также значительным выводом этой работы можно назвать получение опыта применения индексов (для каких целей они применяются, как их корректно создавать и проверят их функционирование).

Эксперименты были построены таким образом, что запущенный поток отправлял случайный запрос, дожидался ответа, а потом снова отправлял запрос. Такое поведение не совсем точно повторяет процесс нагрузки реальной базы данных, а скорее просто "закидывает"ее запросами разного рода, с оптимизацией и без. С такой нагрузкой БД справляется достаточно успешно и ее падения добиться не удалось. При увеличении количества потоков и запросов на поток ОС просто уменьшала процессорный ресурс, который был выделен на обработку потока и запросов от него.

Полученные зависимости получились не совсем показательными. В некоторых экспериментах способы оптимизации показали лучший результат, чем результаты выполнения без оптимизации, но разница вышла небольшая.