МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Базы данных»

Тема: Нагрузочное тестирование

Студентка гр. 2384 _____ Соц Е.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2024

Цель работы

Реализовать ранее спроектированную базу данных с использованием ORM, сделать запросы в соответствии с заданием.

Задание

Вариант 18

1) Написать скрипт, заполняющий БД большим количеством тестовых данных, рекомендуется использовать faker.js.

2) Измерить время выполнения запросов, написанных в ЛР3.

- Проверить для числа записей:
  - 100 записей в каждой табличке
  - 1.000 записей
  - 1.0000 записей
  - 1.000.000 записей
  - можно больше.
- Все запросы выполнять с фиксированным ограничением на вывод (LIMIT), т.к. запросы без LIMIT всегда будет выполняться O(n) от кол-ва записей
- Проверить влияние сортировки на скорость выполнения запросов.
- Для измерения использовать фактическое (не процессорное и т.п.) время. Для node.js есть console.time и console.timeEnd.

3) Добавить в БД индексы (хотя бы 5 штук). Измерить влияние (или его отсутствие) индексов на скорость выполнения запросов. Обратите внимание на:

- Скорость сортировки больших табличек
- Скорость JOIN
- (но остальные запросы тоже проверьте)

Выполнение работы

В ходе выполнения первой лабораторной работы была описана структура базы данных, предназначенная для менеджера музыкальных групп:

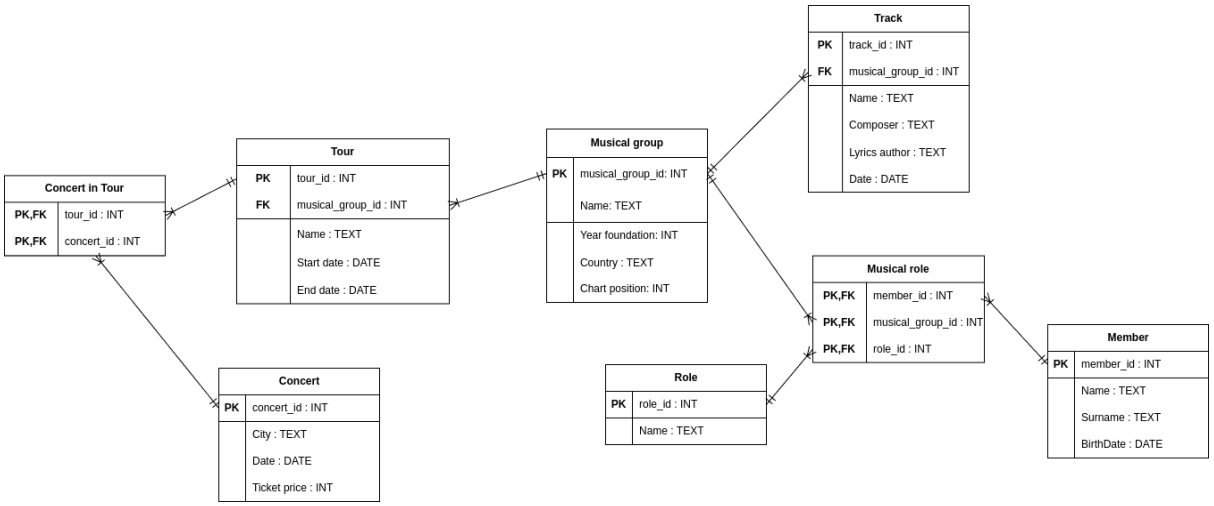

Рисунок 1 – структура БД

В третьей лабораторной работе с помощью Sequelize были созданы модели, соответствующие спроектированной БД.

В данной работе был написан скрипт для заполнения моделей тестовыми данными в большом количестве с использованием библиотеки faker.js. Faker. js — опенсорсный проект, который позволяет генерировать случайные данные для тестирования. Написанный скрипт генерирует уникальные данные для каждой таблицы, учитывая связи между ними. Уникальность достигается с помощью Set, который отслеживает уже сгенерированные значения или комбинации значений. Этот подход позволяет избежать дубликатов и обеспечить целостность данных.

Была протестирована вставка большого количества данных: 10.000 данных в каждую таблицу вставляются без проблем, но при попытке вставить 1.000.000 появлялась ошибка, говорящая о нехватке памяти. Проблема была решена с помощью команды node *--max-old-space-size=16384 faker.js* (выделено 16Гб).

Код скрипта прикреплен в приложении А.

Запросы, написанные в прошлой работе, были протестированы на разных объемах БД. Для некоторых запросов необходимо было знать конкретное название песни, название группы и тд, поэтому перед скриптом father.js был запущен скрипт, реализованный в прошлой работе, который производит вставку данных, написанных собственноручно. Таким образом вставка происходит в начало таблиц. Результаты измерений отображены в табл. 1.

	getTrackInfo	getMostPopularGroupRepertoire	getLastConcertTicketPrice	getGroupMembers	getGroupTourInfo	getAnniversaryGroups	getAnniversaryGroupsTest	getYoungestVocalist
100	19.766	5.968	10.066	8.644	6.261	1.855	2.012	4.05
1000	20.666	6.783	9.121	7.256	6.517	1.904	1.743	4.374
10.000	17.948	8.731	10.039	7.982	7.066	1.84	1.655	3.974
1000.000	109.42	120.183	138.64	121.76	146.8	3.143	2.055	50.522

Таблица 1 – Время выполнения запросов, мс

Также было замерено время выполнения одного и того же запроса с сортировкой и без нее. Результаты замеров приведены в табл. 2.

	с сортировкой	без сортировки
100	2.627	1.855
1000	2.252	1.904
10.000	4.164	1.84
1000.000	16.17	3.143

Таблица 2 – Эксперимент для сортировок, мс

После этого к моделям базы данных были добавлены индексы, а именно к следующим атрибутам: *group_name, chart_position, date_concert, member_name, city.* Затем были снова проведены эксперименты со временем, результаты отображены в таблицах 3 и 4.

	getTrackInfo	getMostPopularGroupRepertoire	getLastConcertTicketPrice	getGroupMembers	getGroupTourInfo	getAnniversaryGroups	getAnniversaryGroupsTest	getYoungestVocalist
100	18.614	4.608	7.331	6.868	6.582	1.878	1.522	3.407
1000	18.17	5.248	7.816	6.932	6.175	1.72	1.577	3.835
10.000	17.162	6.536	9.21	8.411	6.985	1.972	1.711	4.324
1000.000	94.422	58.212	65.83	49.469	60.132	2.098	2.083	49.116

Таблица 3 – Время выполнения запросов с индексами, мс

	с сортировкой	без сортировки
100	2.037	1.878
1000	2.521	1.72
10.000	4.262	1.972
1000.000	4.19	2.098

Таблица 4 – Эксперимент для сортировок с добавлением индексов, мс

Код, написанный для создания индексов, представлен в приложении А.

Графики, полученные после исследований, изображены на рисунках ниже.

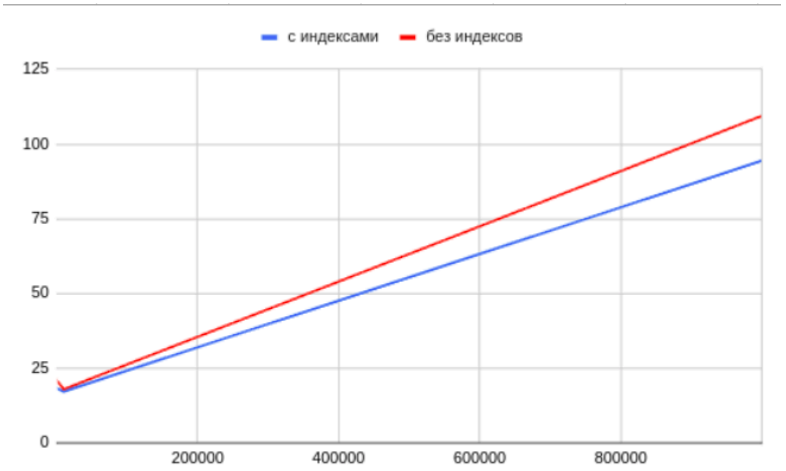

Рисунок 2 – Время выполнения запроса getTrackInfo

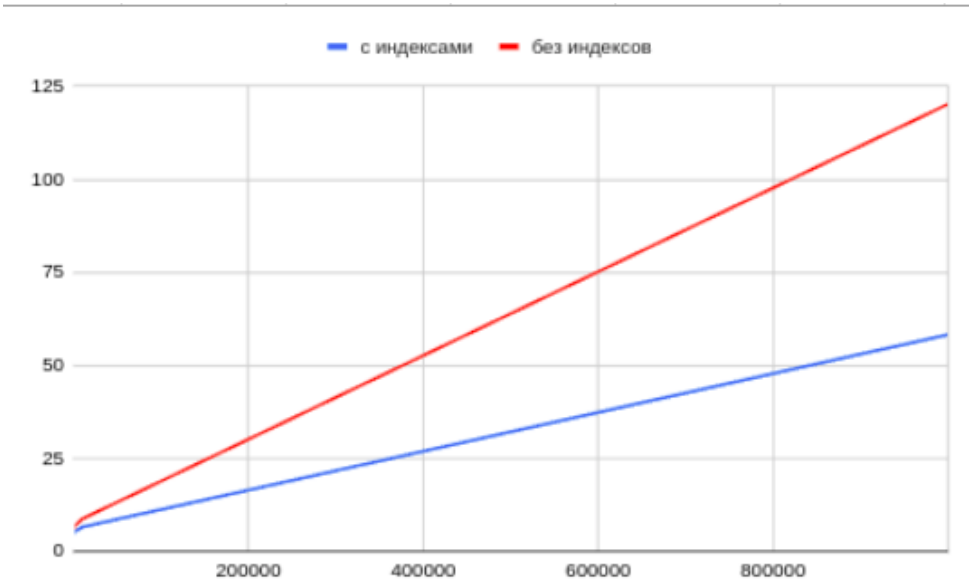

Рисунок 3 – Время выполнения запроса getMostPopularGroupRepertoire

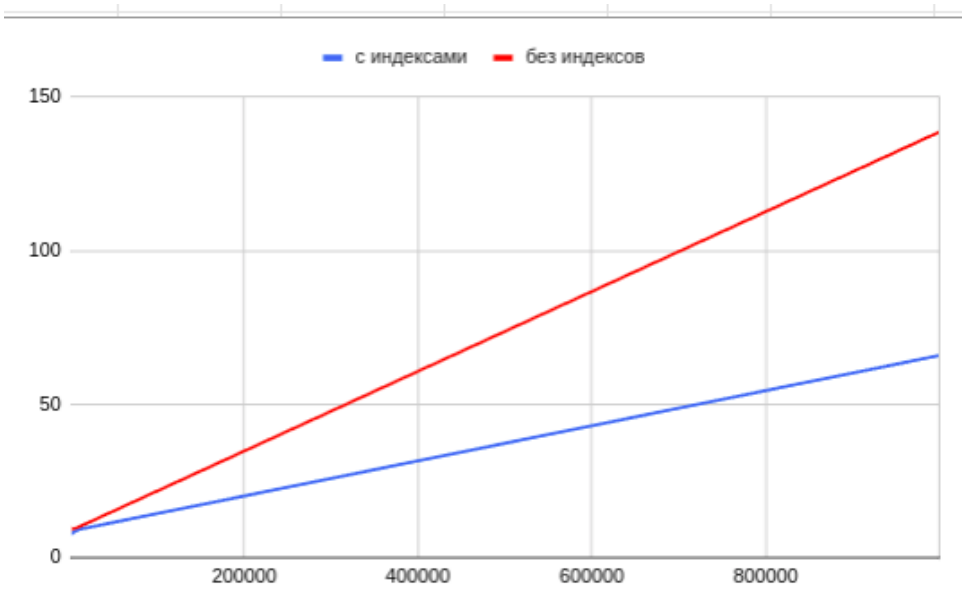

Рисунок 4 – Время выполнения запроса getLastConcertTicketPrice

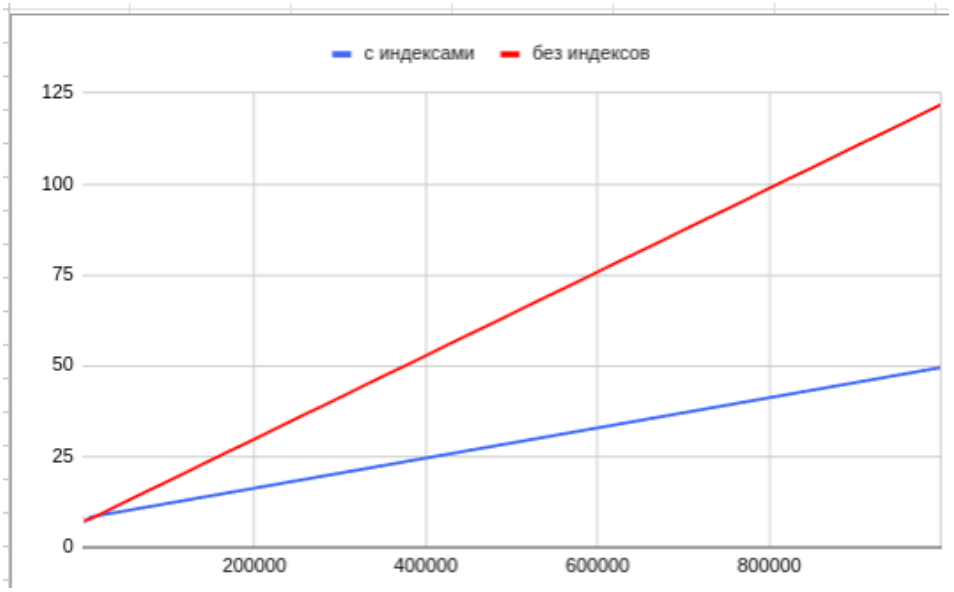

Рисунок 5 – Время выполнения запроса getGroupMembers

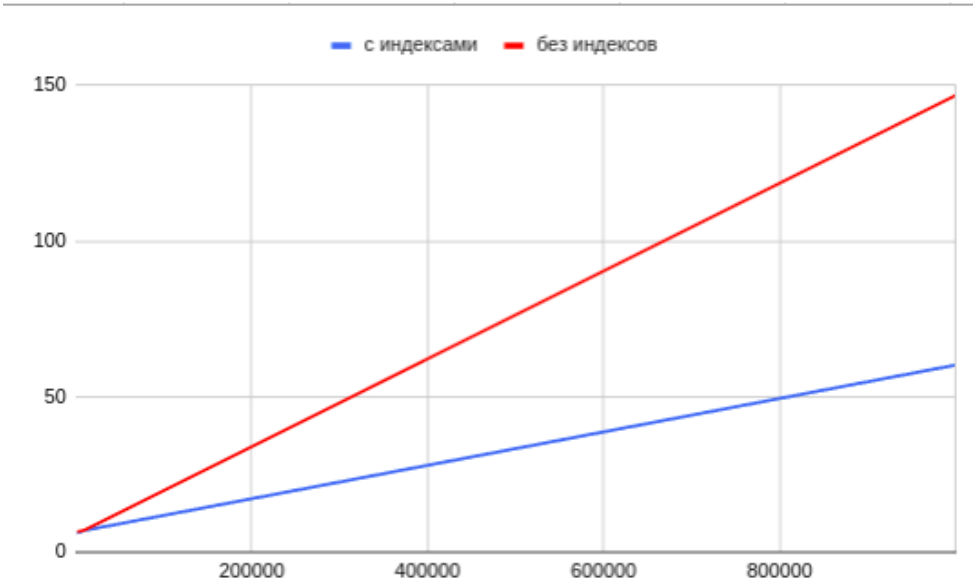

Рисунок 6 – Время выполнения запроса getGroupTourInfo

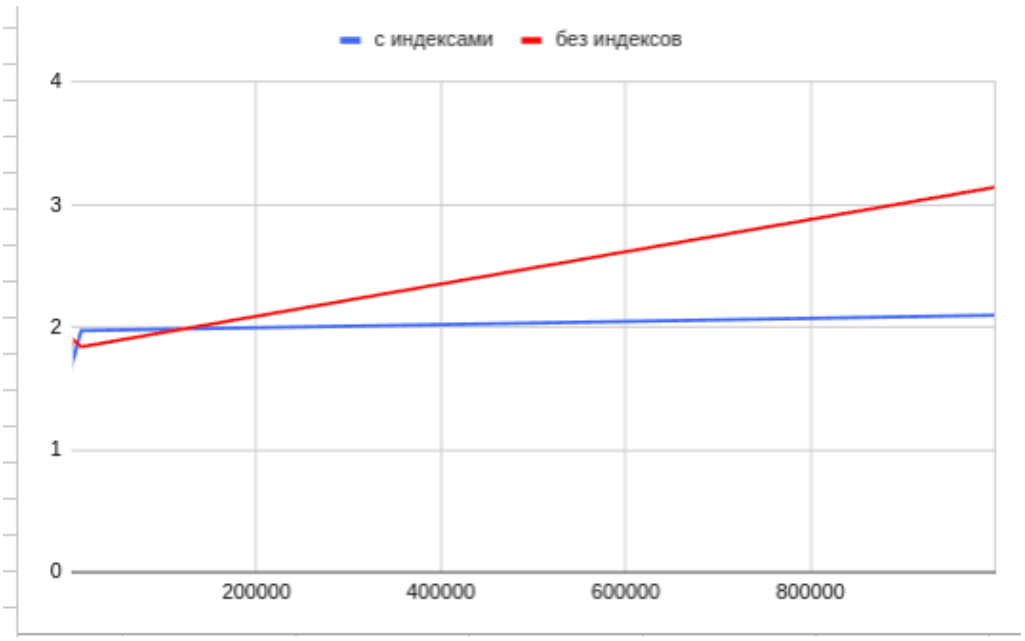

Рисунок 7 – Время выполнения запроса getAnniversaryGroups

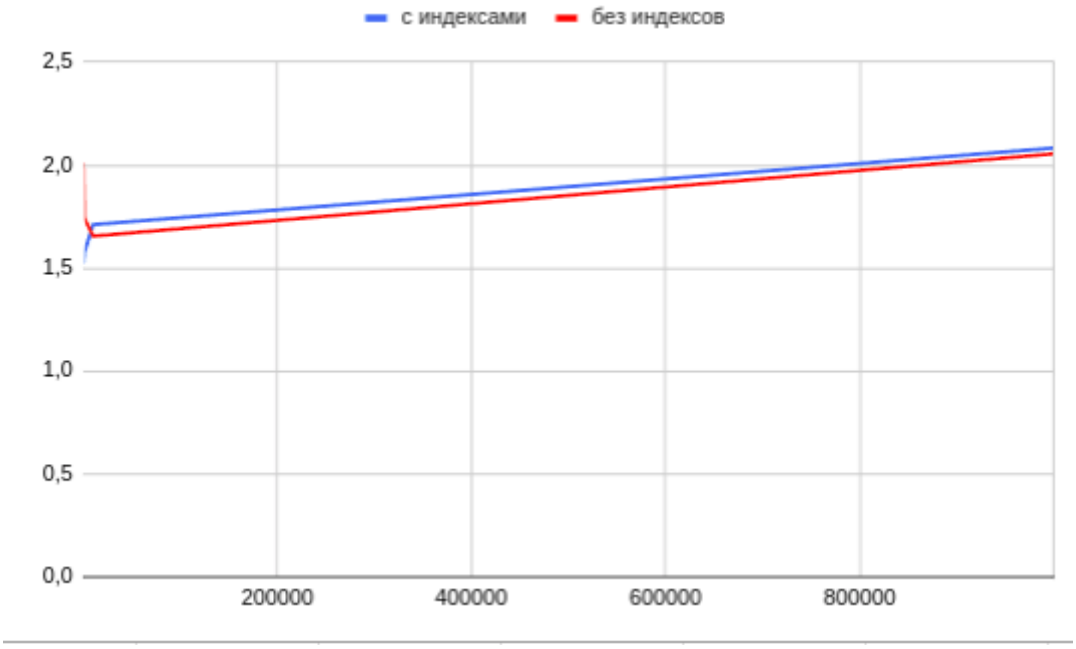

Рисунок 8 – Время выполнения запроса getAnniversaryGroupsTest

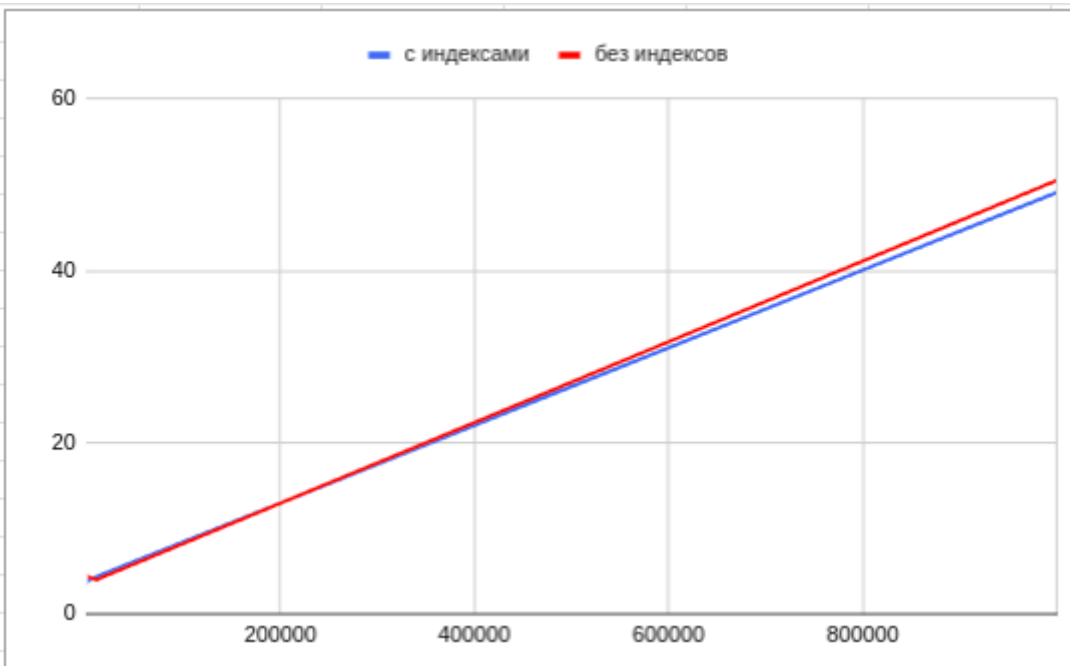

Рисунок 9 – Время выполнения запроса getYoungestVocalist

Можно заметить, что в основном время выполнения запросов с использованием индексов меньше, чем без них. Есть случаи, когда графики совпадают, что говорит о простоте запроса.

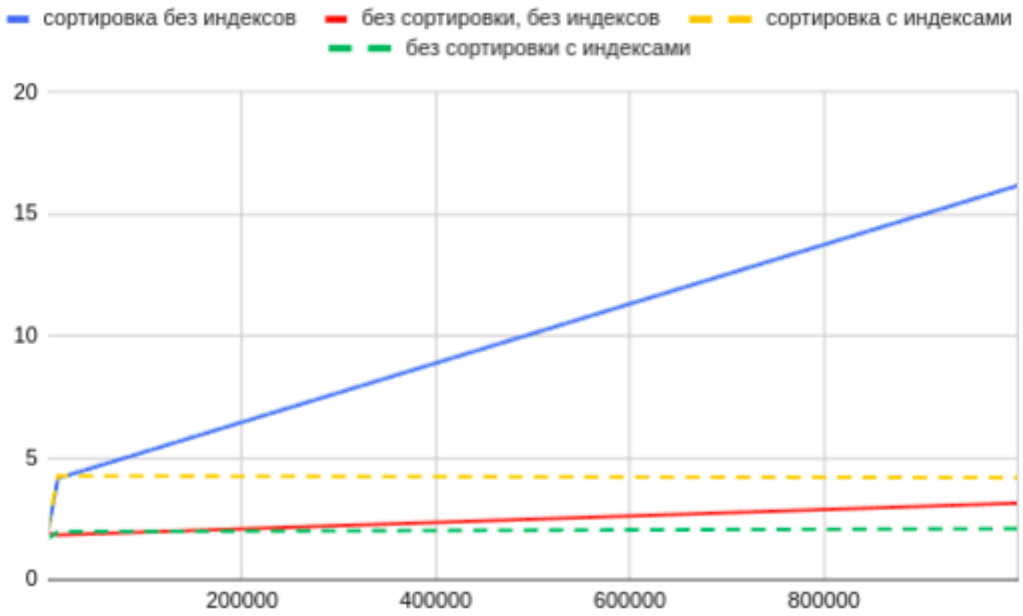

Рисунок 10 – Сравнение времени выполнения запросов с сортировкой и без, с использованием индексов и без

На рисунке 10 явно видно, что время выполнения запроса увеличивается, если в нем присутствует сортировка. Если использовать индексы при сортировке, то время запроса, конечно, уменьшается.

Вывод

В ходе лабораторной работы был создан файл для заполнения БД тестовыми данными с использованием библиотеки faker. Были проведены исследования времени для запросов на разном количестве объемов данных– чем больше количество данных, тем дольше выполняется запрос. Для сокращения времени выполнения запроса можно индексировать БД, что и было сделано. Были проведены исследования, которые отражают, что индексы уменьшают время выполнения запроса. Также были проведены исследования, показывающие, что использование сортировок увеличивает время выполнения запросов. Для всех исследований были сделаны графики.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Скрипт faker.js

```
const sequelize = require('../sequelize');
const MusicalGroup = require('../models/MusicalGroup');
const Track = require('../models/Track');
const Member = require('../models/Member');
const MusicalRole = require('../models/MusicalRole');
const RoleOfMember = require('../models/RoleOfMember');
const Tour = require('../models/Tour');
const Concert = require('../models/Concert');
const ConcertInTour = require('../models/ConcertInTour');
const faker = require('faker');

// Генерация тестовых данных для MusicalGroup
const generateMusicalGroups = (count) => {
  const musicalGroups = [];
  const usedGroupNames = new Set();

  for (let i = 0; i < count; i++) {
    let groupName;
    do {
      groupName = faker.company.companyName();
    } while (usedGroupNames.has(groupName));

    usedGroupNames.add(groupName);

    musicalGroups.push({
      group_name: groupName,
      year_foundation: faker.datatype.number({ min: 1900, max:
2024 }),
      country: faker.address.country(),
      chart_position: faker.datatype.number({ min: 1, max: 100 }),
    });
  }

  return musicalGroups;
};

// Генерация тестовых данных для Track
const generateTracks = (count, musicalGroups) => {
  const tracks = [];
  for (let i = 0; i < count; i++) {
    const randomGroup = musicalGroups[Math.floor(Math.random() *
musicalGroups.length)];
    tracks.push({
      musical_group_id: randomGroup.musical_group_id,
      track_name: faker.lorem.words(3),
      composer: faker.name.findName(),
      lyrics_author: faker.name.findName(),
      release_date: faker.date.past(),
```

```
    });
  }
  return tracks;
};

// Генерация тестовых данных для Member
const generateMembers = (count) => {
  const members = [];
  for (let i = 0; i < count; i++) {
    members.push({
      member_name: faker.name.firstName(),
      surname: faker.name.lastName(),
      birth_date: faker.date.past(),
    });
  }
  return members;
};

// Генерация тестовых данных для MusicalRole
const generateMusicalRoles = (count) => {
  const musicalRoles = [];
  for (let i = 0; i < count; i++) {
    musicalRoles.push({
      role_name: faker.name.jobTitle(),
    });
  }
  return musicalRoles;
};

// Генерация тестовых данных для RoleOfMember
const generateRoleOfMembers = (count, members, musicalGroups,
musicalRoles) => {
  const roleOfMembers = [];
  const usedCombinations = new Set();

  while (roleOfMembers.length < count) {
    const randomMember = members[Math.floor(Math.random() *
members.length)];
    const randomGroup = musicalGroups[Math.floor(Math.random() *
musicalGroups.length)];
    const randomRole = musicalRoles[Math.floor(Math.random() *
musicalRoles.length)];
    const combination =
`${randomMember.member_id}-${randomGroup.musical_group_id}-${rando
mRole.musical_role_id}`;

    if (!usedCombinations.has(combination)) {
      usedCombinations.add(combination);
      roleOfMembers.push({
        member_id: randomMember.member_id,
        musical_group_id: randomGroup.musical_group_id,
        musical_role_id: randomRole.musical_role_id,
      });
    }
  }
```

```
  return roleOfMembers;
};

// Генерация тестовых данных для Tour
const generateTours = (count, musicalGroups) => {
  const tours = [];
  for (let i = 0; i < count; i++) {
    const randomGroup = musicalGroups[Math.floor(Math.random() *
musicalGroups.length)];
    const startDay = faker.date.past();
    const endDay = faker.date.future(1, startDay);
    tours.push({
      musical_group_id: randomGroup.musical_group_id,
      tour_name: faker.lorem.words(3),
      start_day: startDay,
      end_day: endDay,
    });
  }
  return tours;
};

// Генерация тестовых данных для Concert
const generateConcerts = (count) => {
  const concerts = [];
  for (let i = 0; i < count; i++) {
    concerts.push({
      city: faker.address.city(),
      date_concert: faker.date.future(),
      ticket_price: faker.datatype.number({ min: 10, max: 100 }),
    });
  }
  return concerts;
};

// Генерация тестовых данных для ConcertInTour
const generateConcertInTours = (count, tours, concerts) => {
  const concertInTours = [];
  const usedCombinations = new Set();

  while (concertInTours.length < count) {
    const randomTour = tours[Math.floor(Math.random() *
tours.length)];
    const randomConcert = concerts[Math.floor(Math.random() *
concerts.length)];
    const combination =
`${randomTour.tour_id}-${randomConcert.concert_id}`;

    if (!usedCombinations.has(combination)) {
      usedCombinations.add(combination);
      concertInTours.push({
        tour_id: randomTour.tour_id,
        concert_id: randomConcert.concert_id,
      });
    }
  }
```

```
  return concertInTours;
};

(async () => {
  await sequelize.sync({ force: true }); // Удаляет и создает
таблицы заново

  const musicalGroups = await
MusicalGroup.bulkCreate(generateMusicalGroups(1000000));
  const tracks = await Track.bulkCreate(generateTracks(1000000,
musicalGroups));
  const members = await
Member.bulkCreate(generateMembers(1000000));
  const musicalRoles = await
MusicalRole.bulkCreate(generateMusicalRoles(1000000));
  const roleOfMembers = await
RoleOfMember.bulkCreate(generateRoleOfMembers(1000000, members,
musicalGroups, musicalRoles));
  const tours = await Tour.bulkCreate(generateTours(1000000,
musicalGroups));
  const concerts = await
Concert.bulkCreate(generateConcerts(1000000));
  const concertInTours = await
ConcertInTour.bulkCreate(generateConcertInTours(1000000, tours,
concerts));

  console.log('Данные успешно загружены');
})();
```

Файл queries_limit.js

```
const { Op } = require('sequelize');
const sequelize = require('./sequelize');
const MusicalGroup = require('./models/MusicalGroup');
const Track = require('./models/Track');
const Member = require('./models/Member');
const MusicalRole = require('./models/MusicalRole');
const RoleOfMember = require('./models/RoleOfMember');
const Tour = require('./models/Tour');
const Concert = require('./models/Concert');
const ConcertInTour = require('./models/ConcertInTour');

module.exports = {
    //---Автор текста, композитор и дата создания песни с данным
названием? В репертуар какой группы она входит?------
    async getTrackInfo(trackName) {
        const track = await Track.findOne({
          where: { track_name: trackName },
          include: {
            model: MusicalGroup,
            attributes: ['group_name']
          }
        });

        if (track) {
```

```
          console.log(`Composer: ${track.composer}, Lyrics Author:
${track.lyrics_author}, Release Date: ${track.release_date},
Group: ${track.MusicalGroup.group_name}`);
        } else {
          console.log(`Track with name '${trackName}' not
found.`);
        }
      },


//---Репертуар наиболее популярной группы?----------------
  async getMostPopularGroupRepertoire(limit = 10) {
    const mostPopularGroup = await MusicalGroup.findOne({
      order: [['chart_position', 'ASC']]
    });

    const tracks = await Track.findAll({
      where: { musical_group_id: mostPopularGroup.musical_group_id
},
      limit: limit
    });

    tracks.forEach(track => {
      console.log(`Track: ${track.track_name}, Composer:
${track.composer}, Lyrics Author: ${track.lyrics_author}, Release
Date: ${track.release_date}`);
    });
  },


  //---Цена билета на последний концерт указанной
группы?---------------
  async getLastConcertTicketPrice(groupName) {
    const group = await MusicalGroup.findOne({
      where: { group_name: groupName }
    });

    if (!group) {
      console.log(`Group with name '${groupName}' not found.`);
      return;
    }

    const tours = await Tour.findAll({
      where: { musical_group_id: group.musical_group_id }
    });

    const concerts = await ConcertInTour.findAll({
      where: { tour_id: tours.map(tour => tour.tour_id) },
      include: {
        model: Concert,
        attributes: ['date_concert', 'ticket_price']
      },
      order: [[Concert, 'date_concert', 'DESC']],
      limit: 1
    });
```

```
    if (concerts.length > 0) {
      console.log(`Last Concert Ticket Price:
${concerts[0].Concert.ticket_price}`);
    } else {
      console.log(`No concerts found for group '${groupName}'.`);
    }
  },


  //---Состав исполнителей группы с заданным названием, их возраст
и амплуа?------
  async getGroupMembers(groupName, limit = 10) {
   const group = await MusicalGroup.findOne({
    where: { group_name: groupName }
  });

  if (!group) {
    console.log(`Group with name '${groupName}' not found.`);
    return;
  }

  const members = await RoleOfMember.findAll({
    where: { musical_group_id: group.musical_group_id },
    include: [
      {
        model: Member,
        attributes: ['member_name', 'surname', 'birth_date']
      },
      {
        model: MusicalRole,
        attributes: ['role_name']
      }
    ],
    limit: limit
  });

  if (members.length > 0) {
    members.forEach(member => {
      const age = new Date().getFullYear() - new
Date(member.Member.birth_date).getFullYear();
      console.log(`Name: ${member.Member.member_name}, Surname:
${member.Member.surname}, Age: ${age}, Role:
${member.MusicalRole.role_name}`);
    });
  } else {
    console.log(`No members found for group '${groupName}'.`);
  }
},



//---Место и продолжительность гастролей группы с заданным
названием?--------
  async getGroupTourInfo(groupName, limit = 10) {
    const group = await MusicalGroup.findOne({
      where: { group_name: groupName }
```

```
    });

    if (!group) {
      console.log(`Group with name '${groupName}' not found.`);
      return;
    }

    const tours = await Tour.findAll({
      where: { musical_group_id: group.musical_group_id },
      include: {
        model: ConcertInTour,
        include: {
          model: Concert,
          attributes: ['city']
        }
      },
      limit: limit
    });

    if (tours.length > 0) {
      tours.forEach(tour => {
        const duration = new Date(tour.end_day) - new
Date(tour.start_day);
        tour.ConcertInTours.forEach(concertInTour => {
          console.log(`Tour: ${tour.tour_name}, City:
${concertInTour.Concert.city}, Duration: ${duration} ms`);
        });
      });
    } else {
      console.log(`No tours found for group '${groupName}'.`);
    }
  },

  //---Какие группы в текущем году отмечают юбилей?-------
  async getAnniversaryGroups(limit = 10) {
    const currentYear = new Date().getFullYear();
    const groups = await MusicalGroup.findAll({
      where: sequelize.literal(`(${currentYear} - year_foundation)
% 5 = 0 AND ${currentYear} - year_foundation > 0`),
      limit: limit
    });

    groups.forEach(group => {
      const anniversary = currentYear - group.year_foundation;
      console.log(`Group: ${group.group_name}, Foundation Year:
${group.year_foundation}, Anniversary: ${anniversary}`);
    });
  },

        //эта же ф-ия с добавлением сортировки
  async getAnniversaryGroupsSort(limit = 10) {
    const currentYear = new Date().getFullYear();
    const groups = await MusicalGroup.findAll({
      where: sequelize.literal(`(${currentYear} - year_foundation)
% 5 = 0 AND ${currentYear} - year_foundation > 0`),
```

```
      order: [//['year_foundation', 'DESC'], // Сортировка по
убыванию года основания
                ['group_name', 'ASC']],      // Сортировка по
возрастанию названия группы
      limit: limit
    });

    groups.forEach(group => {
      const anniversary = currentYear - group.year_foundation;
      console.log(`Group: ${group.group_name}, Foundation Year:
${group.year_foundation}, Anniversary: ${anniversary}`);
    });
  },


  //---Какие группы в заданном году отмечают юбилей?---------
  async getAnniversaryGroupsTest(my_year, limit = 10) {
    const groups = await MusicalGroup.findAll({
      where: sequelize.literal(`(${my_year} - year_foundation) % 5
= 0 AND ${my_year} - year_foundation > 0`),
      limit: limit
    });

    groups.forEach(group => {
      const anniversary = my_year - group.year_foundation;
      console.log(`Group: ${group.group_name}, Foundation Year:
${group.year_foundation}, Anniversary: ${anniversary}`);
    });
  },


  //---Самый молодой вокалист? Какую группу он
представляет?---------
  async getYoungestVocalist() {
    const vocalist = await RoleOfMember.findOne({
      where: { musical_role_id: 1 },
      include: [
        {
          model: Member,
          attributes: ['member_name', 'surname', 'birth_date']
        },
        {
          model: MusicalGroup,
          attributes: ['group_name']
        }
      ],
      order: [[Member, 'birth_date', 'DESC']]
    });

    const age = new Date().getFullYear() - new
Date(vocalist.Member.birth_date).getFullYear();
    console.log(`Name: ${vocalist.Member.member_name}, Surname:
${vocalist.Member.surname}, Age: ${age}, Group:
${vocalist.MusicalGroup.group_name}`);
  }
};
```

Файл create_index.js

```
const sequelize = require('../sequelize');
async function createIndexes() {
  try {
    await sequelize.query(`
      CREATE INDEX musical_group_group_name_idx ON musical_group
(group_name);
      CREATE INDEX musical_group_chart_position_idx ON
musical_group (chart_position);
      CREATE INDEX concert_date_concert_idx ON concert
(date_concert);
      CREATE INDEX concert_city_idx ON concert (city);
      CREATE INDEX member_member_name_idx ON member (member_name);
    `);
    console.log('Индексы успешно созданы.');
  } catch (error) {
    console.error('Ошибка при создании индексов:', error);
  }
}

createIndexes();
```

ПРИЛОЖЕНИЕ В

ССЫЛКА НА PR

https://github.com/moevm/sql-2024-2384/pull/22