

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Базы данных»
ТЕМА: РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ORM

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

В данной лабораторной работе рекомендуется использовать Sequelize (Node.js).

Необходимо развернуть Sequelize на своем ПК и выполнить следующие задачи:

- Описать в виде моделей Sequelize таблицы из 1-й лабораторной работы
- Написать скрипт заполнения тестовыми данными: 5-10 строк на каждую таблицу, обязательно наличие связи между ними, данные приближены к реальности.
- Написать запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием ORM. Вывести результаты в консоль (или иной человеко-читабельный вывод)
- Запустить в репозиторий исходный код проекта, соблюсти .gitignore, убрать исходную базу из проекта (или иные нагенерированные данные бд если они есть).
- Описать процесс запуска: команды, зависимости
- В отчете описать цель, текст задания в соответствии с вариантом, выбранную ORM, инструкцию по запуску, скриншоты (код) моделей ORM, скриншоты на каждый запрос (или группу запросов) на изменение/таблицы с выводом результатов (ответ), ссылку на PR в приложении, вывод

Выполнение работы

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

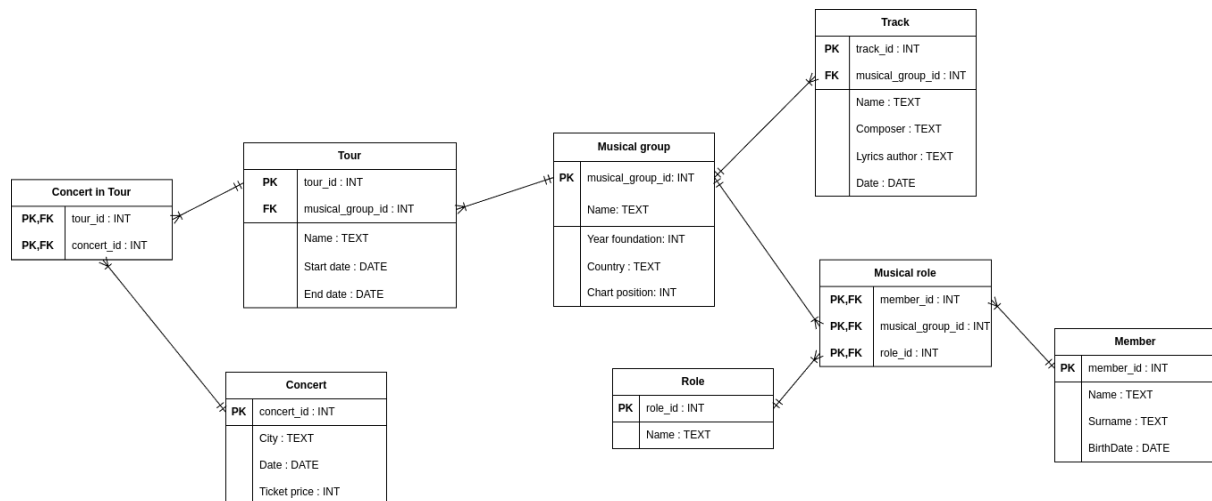


Рисунок 1 – структура БД

Был создан конфигурационный объект, который может использоваться для настройки подключения к базе данных с использованием Sequelize.

```
module.exports = {  
  database: 'mydb',  
  username: 'katya',  
  password: '123',  
  host: 'localhost',  
  dialect: 'postgres'  
};
```

Рисунок 2 – config.js

Также был создан файл, где происходит создание экземпляра Sequelize для подключения к базе данных PostgreSQL с использованием конфигурационных параметров, загруженных из файла ./config/config.

```
const { Sequelize } = require('sequelize');
const config = require('./config/config');

const sequelize = new Sequelize(config.database, config.username, config.password, {
  host: config.host,
  dialect: 'postgres'
});

module.exports = sequelize;
```

Рисунок 3 – sequelize.js

Рассмотрим модели, созданные с помощью Sequelize: используется метод `sequelize.define()`. Модель в Sequelize представляет собой таблицу в базе данных, а её атрибуты (поля) представляют столбцы этой таблицы.

Код с описанием моделей прикреплен в приложении А.

Можно заметить, что каждый файл модели начинается с импорта необходимых модулей: `DataTypes` – объект, содержащий различные типы данных, которые могут быть использованы для определения полей модели; `sequelize` – экземпляр Sequelize, который уже настроен для подключения к базе данных.

Также файлах моделей описаны связи между ними:

Метод `hasMany()` определяет связь "один ко многим" (one-to-many). Это означает, что одна запись в одной таблице может быть связана с несколькими записями в другой таблице.

Метод `belongsTo()` определяет связь "многие к одному" (many-to-one). Это означает, что несколько записей в одной таблице могут быть связаны с одной записью в другой таблице.

Для заполнения базы данных был написан скрипт: он создает таблицы, если они не существуют, и заполняет их тестовыми данными. Метод `bulkCreate` используется для массового создания записей в таблицах. Он принимает массив объектов, каждый из которых представляет запись, которую нужно добавить в таблицу.

Код скрипта прикреплен в приложении А.

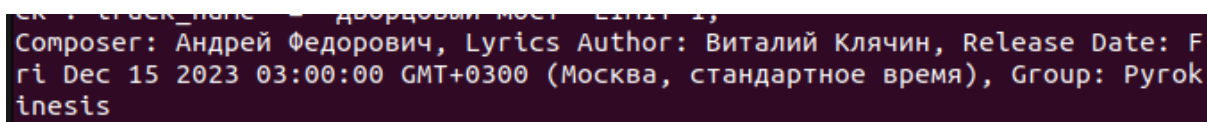
В файле queries.js реализованы запросы, которые описаны в тексте задания.

Код запросов тоже прикреплен в приложении А.

Ниже прикреплены снимки экрана с выводом запросов.

1) Автор текста, композитор и дата создания песни с данным названием? В репертуар какой группы она входит?

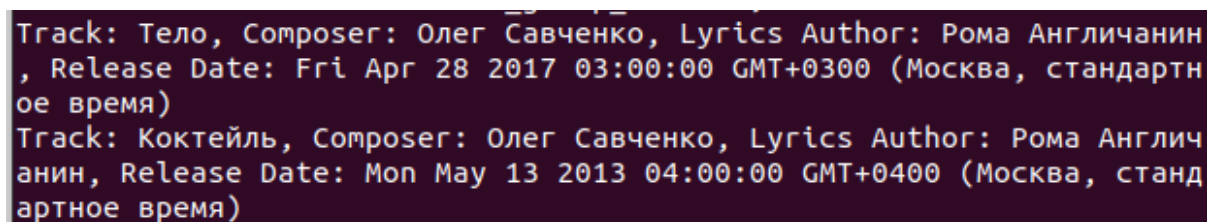
Для теста была выбрана песня “Дворцовый мост”.



```
Track: Дворцовый мост, Composer: Андрей Федорович, Lyrics Author: Виталий Клячин, Release Date: Fri Dec 15 2023 03:00:00 GMT+0300 (Москва, стандартное время), Group: Pyrokinesis
```

Рисунок 4 – Информация об определенной песне

2) Репертуар наиболее популярной группы?

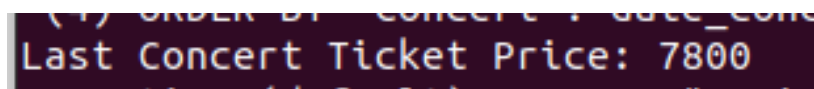


```
Track: Тело, Composer: Олег Савченко, Lyrics Author: Рома Англичанин, Release Date: Fri Apr 28 2017 03:00:00 GMT+0300 (Москва, стандартное время)
Track: Коктейль, Composer: Олег Савченко, Lyrics Author: Рома Англичанин, Release Date: Mon May 13 2013 04:00:00 GMT+0400 (Москва, стандартное время)
```

Рисунок 5 – Репертуар самой популярной группы

3) Цена билета на последний концерт указанной группы?

Для теста была выбрана группа “Монеточка”.

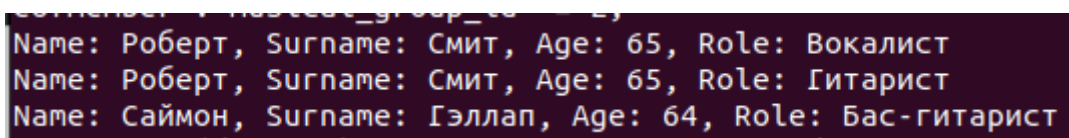


```
Last Concert Ticket Price: 7800
```

Рисунок 6 – Цена билета

4) Состав исполнителей группы с заданным названием, их возраст и амплуа?

Для теста была выбрана группа “The Cure”.



```
Name: Роберт, Surname: Смит, Age: 65, Role: Вокалист
Name: Роберт, Surname: Смит, Age: 65, Role: Гитарист
Name: Саймон, Surname: Гэллп, Age: 64, Role: Бас-гитарист
```

Рисунок 7 – Информация о составе определенной группы

- 5) Место и продолжительность гастролей группы с заданным названием?

Для теста была выбрана группа “ЛСП”.

```
= 5;  
Tour: Бриллианты, City: Тюмень, Duration: 4924800000 ms  
Tour: Бриллианты, City: Омск, Duration: 4924800000 ms  
Executing (default): SELECT "musical_group_id", "group_n
```

Рисунок 8 – Информация о турах заданной группы

- 6) Какие группы в текущем году отмечают юбилей?

```
async getAnniversaryGroups() {  
  const currentYear = new Date().getFullYear();  
  const groups = await MusicalGroup.findAll({  
    where: sequelize.literal(`${currentYear} - year_foundation) % 5 = 0 AND ${currentYear} - year_foundation > 0`)  
  });  
  
  groups.forEach(group => {  
    const anniversary = currentYear - group.year_foundation;  
    console.log(`Group: ${group.group_name}, Foundation Year: ${group.year_foundation}, Anniversary: ${anniversary}`);  
  });  
},
```

Рисунок 9 – Запрос “Юбилей в текущем году”

Никакой информации не получено, но это правильный ответ, ведь по моим тестовым данным никакая группа не отмечает юбилей текущем году.

Для корректной проверки запроса можно задать какой-то конкретный год юбилея:

```
async getAnniversaryGroupsTest(my_year) {  
  const groups = await MusicalGroup.findAll({  
    where: sequelize.literal(`${my_year} - year_foundation) % 5 = 0 AND ${my_year} - year_foundation > 0`)  
  });  
  
  groups.forEach(group => {  
    const anniversary = my_year - group.year_foundation;  
    console.log(`Group: ${group.group_name}, Foundation Year: ${group.year_foundation}, Anniversary: ${anniversary}`);  
  });  
},
```

Рисунок 10 – Запрос “Юбилей в конкретном году”

```
at_foundation > 0;  
Group: Queen, Foundation Year: 1970, Anniversary: 55  
Group: Pink Floyd, Foundation Year: 1965, Anniversary: 60  
Group: Монеточка, Foundation Year: 2015, Anniversary: 10
```

Рисунок 11 – Юбилей в конкретном году

- 7) Самый молодой вокалист? Какую группу он представляет?

```
cn_date DESC LIMIT 1;  
Name: Елизавета, Surname: Гырдымова, Age: 26, Group: Монеточка  
Execution (default): SELECT "artist", "group", "age"
```

Рисунок 12 – Самый молодой вокалист

Для выполнения набора запросов был реализован файл app.js:

```
const sequelize = require('./sequelize');  
const queries = require('./queries');  
  
(async () => {  
  try {  
    await sequelize.authenticate();  
    console.log('Connection has been established successfully.');  
    await queries.getTrackInfo('Дворцовый мост');  
    await queries.getMostPopularGroupRepertoire();  
    await queries.getLastConcertTicketPrice('Монеточка');  
    await queries.getGroupMembers('The Cure');  
    await queries.getGroupTourInfo('ЛСП');  
    await queries.getAnniversaryGroups();  
    await queries.getYoungestVocalist();  
    await queries.getAnniversaryGroupsTest('2025');  
  } catch (error) {  
    console.error('Unable to connect to the database:', error);  
  }  
})();
```

Рисунок 13 – Запуск запросов

В приложении В представлена ссылка на PR.

Вывод

В ходе лабораторной работы были созданы модели ранее спроектированной базы данных с использованием Sequelize. Был написан скрипт для заполнения тестовыми данными с помощью NodeJS и описаны связи между таблицами. Также были реализованы запросы в соответствии с заданием и был проведен анализ ответов: ответы на запросы со второй лабораторной работы совпадают с ответами из третьей лабораторной работы, что говорит о правильности написания запросов (тестовые данные для наполнения таблиц совпадают).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Модель MusicalGroup:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const MusicalGroup = sequelize.define('MusicalGroup', {
  musical_group_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  group_name: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  year_foundation: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  country: {
    type: DataTypes.STRING(30),
    allowNull: false
  },
  chart_position: {
    type: DataTypes.INTEGER,
    allowNull: false
  }
}, {
  tableName: 'musical_group',
  timestamps: false
});

module.exports = MusicalGroup;
```

Модель Track:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');
const MusicalGroup = require('../MusicalGroup');

const Track = sequelize.define('Track', {
  track_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  musical_group_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: MusicalGroup,

```

```

        key: 'musical_group_id'
      }
    },
    track_name: {
      type: DataTypes.STRING(50),
      allowNull: false
    },
    composer: {
      type: DataTypes.STRING(50),
      allowNull: false
    },
    lyrics_author: {
      type: DataTypes.STRING(50),
      allowNull: false
    },
    release_date: {
      type: DataTypes.DATE,
      allowNull: false
    }
  }, {
    tableName: 'track',
    timestamps: false
  });

```

```

Track.belongsTo(MusicalGroup, { foreignKey: 'musical_group_id' });
MusicalGroup.hasMany(Track, { foreignKey: 'musical_group_id' });

```

```

module.exports = Track;

```

Модель Member:

```

const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const Member = sequelize.define('Member', {
  member_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  member_name: {
    type: DataTypes.STRING(30),
    allowNull: false
  },
  surname: {
    type: DataTypes.STRING(30),
    allowNull: false
  },
  birth_date: {
    type: DataTypes.DATE,
    allowNull: false
  }
}, {
  tableName: 'member',
  timestamps: false
});

```

```
module.exports = Member;
```

Модель MusicalRole:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const MusicalRole = sequelize.define('MusicalRole', {
  musical_role_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  role_name: {
    type: DataTypes.STRING(30),
    allowNull: false
  }
}, {
  tableName: 'musical_role',
  timestamps: false
});

module.exports = MusicalRole;
```

Модель RoleOfMember:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');
const Member = require('./Member');
const MusicalGroup = require('./MusicalGroup');
const MusicalRole = require('./MusicalRole');

const RoleOfMember = sequelize.define('RoleOfMember', {
  member_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    references: {
      model: Member,
      key: 'member_id'
    }
  },
  musical_group_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    references: {
      model: MusicalGroup,
      key: 'musical_group_id'
    }
  },
  musical_role_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    references: {
      model: MusicalRole,
      key: 'musical_role_id'
    }
  }
});
```

```

    }
  }, {
    tableName: 'role_of_member',
    timestamps: false
  });

RoleOfMember.belongsTo(Member, { foreignKey: 'member_id' });
RoleOfMember.belongsTo(MusicalGroup, { foreignKey:
'musical_group_id' });
RoleOfMember.belongsTo(MusicalRole, { foreignKey:
'musical_role_id' });

Member.hasMany(RoleOfMember, { foreignKey: 'member_id' });
MusicalGroup.hasMany(RoleOfMember, { foreignKey:
'musical_group_id' });
MusicalRole.hasMany(RoleOfMember, { foreignKey: 'musical_role_id'
});

module.exports = RoleOfMember;

```

Модель Tour:

```

const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');
const MusicalGroup = require('./MusicalGroup');

const Tour = sequelize.define('Tour', {
  tour_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  musical_group_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: MusicalGroup,
      key: 'musical_group_id'
    }
  },
  tour_name: {
    type: DataTypes.STRING(50),
    allowNull: false
  },
  start_day: {
    type: DataTypes.DATE,
    allowNull: false
  },
  end_day: {
    type: DataTypes.DATE,
    allowNull: false
  }
}, {
  tableName: 'tour',
  timestamps: false

```

```
});

Tour.belongsTo(MusicalGroup, { foreignKey: 'musical_group_id' });
MusicalGroup.hasMany(Tour, { foreignKey: 'musical_group_id' });

module.exports = Tour;
```

Модель Concert:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const Concert = sequelize.define('Concert', {
  concert_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  city: {
    type: DataTypes.STRING(30),
    allowNull: false
  },
  date_concert: {
    type: DataTypes.DATE,
    allowNull: false
  },
  ticket_price: {
    type: DataTypes.INTEGER,
    allowNull: false
  }
}, {
  tableName: 'concert',
  timestamps: false
});

module.exports = Concert;
```

Модель ConcertINTour:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');
const Tour = require('./Tour');
const Concert = require('./Concert');

const ConcertInTour = sequelize.define('ConcertInTour', {
  tour_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    references: {
      model: Tour,
      key: 'tour_id'
    }
  },
  concert_id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
```

```

    references: {
      model: Concert,
      key: 'concert_id'
    }
  }, {
    tableName: 'concert_in_tour',
    timestamps: false
  });

ConcertInTour.belongsTo(Tour, { foreignKey: 'tour_id' });
ConcertInTour.belongsTo(Concert, { foreignKey: 'concert_id' });

Tour.hasMany(ConcertInTour, { foreignKey: 'tour_id' });
Concert.hasMany(ConcertInTour, { foreignKey: 'concert_id' });

module.exports = ConcertInTour;

```

Скрипт для заполнения моделей seed.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');

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

  await MusicalGroup.bulkCreate([
    { group_name: 'Queen', year_foundation: 1970, country:
'Великобритания', chart_position: 5 },
    { group_name: 'The Cure', year_foundation: 1978, country:
'Англия', chart_position: 2 },
    { group_name: 'Pink Floyd', year_foundation: 1965, country:
'Англия', chart_position: 4 },
    { group_name: 'Монеточка', year_foundation: 2015, country:
'Россия', chart_position: 3 },
    { group_name: 'ЛСП', year_foundation: 2007, country:
'Беларусь', chart_position: 1 },
    { group_name: 'Pyrokinesis', year_foundation: 2012, country:
'Россия', chart_position: 6 }
  ]);

  await Track.bulkCreate([
    { musical_group_id: 1, track_name: 'Bohemian Rhapsody',
composer: 'Queen', lyrics_author: 'Фредди Меркьюри', release_date:
'1975-10-31' },

```

```

        { musical_group_id: 1, track_name: 'We Will Rock You',
composer: 'Брайан Мэй', lyrics_author: 'Брайан Мэй', release_date:
'1977-10-07' },
        { musical_group_id: 2, track_name: 'Lovesong', composer:
'Роберт Смит', lyrics_author: 'Роберт Смит', release_date:
'1989-08-21' },
        { musical_group_id: 2, track_name: 'Friday I am in Love',
composer: 'Роберт Смит', lyrics_author: 'Роберт Смит',
release_date: '1992-05-11' },
        { musical_group_id: 3, track_name: 'Another Brick in the
Wall', composer: 'Роджер Уотерс', lyrics_author: 'Роджер Уотерс',
release_date: '1979-11-30' },
        { musical_group_id: 3, track_name: 'Time', composer: 'Роджер
Уотерс', lyrics_author: 'Роджер Уотерс', release_date:
'1973-01-01' },
        { musical_group_id: 4, track_name: 'Это было в России',
composer: 'Витя Исаев', lyrics_author: 'Витя Исаев', release_date:
'2024-05-01' },
        { musical_group_id: 4, track_name: 'Каждый раз', composer:
'Елизавета Гырдымова', lyrics_author: 'Елизавета Гырдымова',
release_date: '2018-05-25' },
        { musical_group_id: 5, track_name: 'Тело', composer: 'Олег
Савченко', lyrics_author: 'Рома Англичанин', release_date:
'2017-04-28' },
        { musical_group_id: 5, track_name: 'Коктейль', composer: 'Олег
Савченко', lyrics_author: 'Рома Англичанин', release_date:
'2013-05-13' },
        { musical_group_id: 6, track_name: 'Черное солнышко',
composer: 'Андрей Федорович', lyrics_author: 'Андрей Федорович',
release_date: '2019-08-15' },
        { musical_group_id: 6, track_name: 'Дворцовый мост', composer:
'Андрей Федорович', lyrics_author: 'Виталий Клячин', release_date:
'2023-12-15' }
    ]);

    await Member.bulkCreate([
        { member_name: 'Брайн', surname: 'Мэй', birth_date:
'1947-07-19' },
        { member_name: 'Роджер', surname: 'Тейлор', birth_date:
'1949-07-26' },
        { member_name: 'Роберт', surname: 'Смит', birth_date:
'1959-04-21' },
        { member_name: 'Саймон', surname: 'Гэллп', birth_date:
'1960-06-01' },
        { member_name: 'Ник', surname: 'Мейсон', birth_date:
'1944-01-27' },
        { member_name: 'Роджер', surname: 'Уотерс', birth_date:
'1943-09-06' },
        { member_name: 'Елизавета', surname: 'Гырдымова', birth_date:
'1998-06-01' },
        { member_name: 'Олег', surname: 'Савченко', birth_date:
'1989-07-10' },
        { member_name: 'Петр', surname: 'Клюев', birth_date:
'1989-05-28' },
        { member_name: 'Андрей', surname: 'Федорович', birth_date:
'1995-12-16' }
    ])

```

```

]);

await MusicalRole.bulkCreate([
  { role_name: 'Вокалист' },
  { role_name: 'Барабанщик' },
  { role_name: 'Гитарист' },
  { role_name: 'Бэк-вокалист' },
  { role_name: 'Автор песен' },
  { role_name: 'Бас-гитарист' }
]);

await RoleOfMember.bulkCreate([
  { member_id: 1, musical_group_id: 1, musical_role_id: 3 },
  { member_id: 1, musical_group_id: 1, musical_role_id: 5 },
  { member_id: 2, musical_group_id: 1, musical_role_id: 1 },
  { member_id: 2, musical_group_id: 1, musical_role_id: 2 },
  { member_id: 3, musical_group_id: 2, musical_role_id: 1 },
  { member_id: 3, musical_group_id: 2, musical_role_id: 3 },
  { member_id: 4, musical_group_id: 2, musical_role_id: 6 },
  { member_id: 5, musical_group_id: 3, musical_role_id: 2 },
  { member_id: 6, musical_group_id: 3, musical_role_id: 1 },
  { member_id: 6, musical_group_id: 3, musical_role_id: 6 },
  { member_id: 7, musical_group_id: 4, musical_role_id: 1 },
  { member_id: 7, musical_group_id: 4, musical_role_id: 5 },
  { member_id: 8, musical_group_id: 5, musical_role_id: 1 },
  { member_id: 8, musical_group_id: 5, musical_role_id: 5 },
  { member_id: 9, musical_group_id: 5, musical_role_id: 4 },
  { member_id: 10, musical_group_id: 6, musical_role_id: 1 },
  { member_id: 10, musical_group_id: 6, musical_role_id: 5 }
]);

await Tour.bulkCreate([
  { musical_group_id: 1, tour_name: 'Magic Tour', start_day:
'1986-06-07', end_day: '1986-08-09' },
  { musical_group_id: 2, tour_name: 'Shows of a lost world',
start_day: '2023-11-19', end_day: '2023-12-10' },
  { musical_group_id: 3, tour_name: 'Syd BARRET', start_day:
'2007-05-10', end_day: '2007-05-10' },
  { musical_group_id: 4, tour_name: 'МОЛИТВЫ. Анекдоты. Тосты',
start_day: '2024-11-11', end_day: '2024-11-30' },
  { musical_group_id: 5, tour_name: 'Бриллианты', start_day:
'2024-10-12', end_day: '2024-12-08' },
  { musical_group_id: 6, tour_name: 'Typ-2024', start_day:
'2024-09-06', end_day: '2024-10-27' }
]);

await Concert.bulkCreate([
  { city: 'Париж', date_concert: '1986-06-14', ticket_price:
1000 },
  { city: 'Лондон', date_concert: '1986-07-11', ticket_price:
1000 },
  { city: 'Буэнос-Айрес', date_concert: '2023-11-25',
ticket_price: 5000 },
  { city: 'Монтевидео', date_concert: '2023-11-27',
ticket_price: 5100 },

```



```

    { city: 'Лондон', date_concert: '2007-05-10', ticket_price:
2000 },
    { city: 'Барселона', date_concert: '2024-11-25', ticket_price:
5800 },
    { city: 'Прага', date_concert: '2024-11-30', ticket_price:
7800 },
    { city: 'Тюмень', date_concert: '2024-11-19', ticket_price:
2600 },
    { city: 'Омск', date_concert: '2024-12-08', ticket_price: 3200
},
    { city: 'Орел', date_concert: '2024-10-08', ticket_price: 2500
},
    { city: 'Тамбов', date_concert: '2024-10-11', ticket_price:
2000 }
  ]);

  await ConcertInTour.bulkCreate([
    { tour_id: 1, concert_id: 1 },
    { tour_id: 1, concert_id: 2 },
    { tour_id: 2, concert_id: 3 },
    { tour_id: 2, concert_id: 4 },
    { tour_id: 3, concert_id: 5 },
    { tour_id: 4, concert_id: 6 },
    { tour_id: 4, concert_id: 7 },
    { tour_id: 5, concert_id: 8 },
    { tour_id: 5, concert_id: 9 },
    { tour_id: 6, concert_id: 10 },
    { tour_id: 6, concert_id: 11 }
  ]);

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

```

Запросы queries.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 = {
  async getTrackInfo(trackName) {
    const track = await Track.findOne({
      where: { track_name: trackName },
      include: {
        model: MusicalGroup,
        attributes: ['group_name']
      }
    });
  }
};

```

```

    console.log(`Composer: ${track.composer}, Lyrics Author:
${track.lyrics_author}, Release Date: ${track.release_date}, Group:
${track.MusicalGroup.group_name}`);
  },

  async getMostPopularGroupRepertoire() {
    const mostPopularGroup = await MusicalGroup.findOne({
      order: [['chart_position', 'ASC']]
    });

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

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

  async getLastConcertTicketPrice(groupName) {
    const group = await MusicalGroup.findOne({
      where: { group_name: groupName }
    });

    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']]
    });

    console.log(`Last Concert Ticket Price:
${concerts[0].Concert.ticket_price}`);
  },

  async getGroupMembers(groupName) {
    const group = await MusicalGroup.findOne({
      where: { group_name: groupName }
    });

    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']
        }
      ]
    });
  }
}

```

```

        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}`);
        });
    },

    async getGroupTourInfo(groupName) {
        const group = await MusicalGroup.findOne({
            where: { group_name: groupName }
        });

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

        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`);
            });
        });
    },

    async getAnniversaryGroups() {
        const currentYear = new Date().getFullYear();
        const groups = await MusicalGroup.findAll({
            where: sequelize.literal(`(${currentYear} - year_foundation) % 5 = 0
AND ${currentYear} - year_foundation > 0`)
        });

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

    async getAnniversaryGroupsTest(my_year) {
        const groups = await MusicalGroup.findAll({
            where: sequelize.literal(`(${my_year} - year_foundation) % 5 = 0 AND
${my_year} - year_foundation > 0`)
        });

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

```

```

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}`);
}
};

```

ПРИЛОЖЕНИЕ В

ССЫЛКА НА PR

<https://github.com/moevm/sql-2024-2384/pull/19>