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 БД

Студент гр. 1304 _____ Шаврин А.П.

Преподаватель _____ Заславский М.М

Санкт-Петербург

2023

Цель работы.

Проведение нагрузочного тестирования на СУБД PostgreSQL при помощи ORM.

Задание.

Вариант 4.

Пусть требуется создать программную систему, предназначенную для организаторов выставки собак. Она должна обеспечивать хранение сведений о собаках - участниках выставки и экспертах. Для каждой собаки в БД должны храниться сведения, о том, к какому клубу она относится, кличка, порода и возраст, сведения о родословной (номер документа, клички родителей), дата последней прививки, фамилия, имя, отчество и паспортные данные хозяина. На каждый клуб отводится участок номеров, под которыми будут выступать участники выставки. Сведения об эксперте должны включать фамилию и имя, номер ринга, который он обслуживает; клуб, название клуба, в котором он состоит. Каждый ринг могут обслуживать несколько экспертов. Каждая порода собак выступает на своем ринге, но на одном и том же ринге в разное время могут выступать разные породы. Итогом выставки является определение медалистов по каждой породе. Организатор выставки должен иметь возможность добавить в базу нового участника или нового эксперта, снять эксперта с судейства, заменив его другим, отстранить собаку от участия в выставке. Организатору выставки могут потребоваться следующие сведения;

- На каком ринге выступает заданный хозяин со своей собакой?
- Какими породами представлен заданный клуб?
- Какие медали и сколько заслужены клубом?
- Какие эксперты обслуживают породу?
- Количество участников по каждой породе?

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

- 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.
- 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

Выполнение работы.

1. Сперва был изменен файл заполнения базы данных fill_db.ts Для генерации фейковых данных использован faker-js

```
import * as models from "./models/models.js"
import { faker } from "@faker-js/faker"
const COUNT = 100;
const UNIQUE_VALUES = uniqueValues();


export async function fill_db() {
    // fill Rings
    let rings: any[] = createRings()

    await models.Ring.bulkCreate(
        rings.map((ring) => (ring)),
        { returning: false }
    );

    //fill Breeds
    let breeds: any[] = createBreeds();

    await models.Breed.bulkCreate(
        breeds.map((breed) => (breed)),
        { returning: false }
    );

    //fill clubs
    let clubs: any[] = faker.helpers.multiple(createClub, {count: COUNT});

    await models.Club.bulkCreate(
        clubs.map((club) => (club)),
        { returning: false }
    );

    //fill experts
    let experts: any[] = faker.helpers.multiple(createExpert, {count: COUNT});

    await models.Expert.bulkCreate(
        experts.map((expert) => (expert)),
        { returning: false }
    );
```

```
//fill owners
let owners: any[] = createOwners();

await models.Owner.bulkCreate(
    owners.map((owner) => (owner)),
    { returning: false }
);

//fill dogs
let dogs: any[] = faker.helpers.multiple(createDog, {count: COUNT});

await models.Dog.bulkCreate(
    dogs.map((dog) => (dog)),
    { returning: false }
);

//fill club numbers
let club_numbers: any[] = createClubNumbers();

await models.ClubNumber.bulkCreate(
    club_numbers.map((club_number) => (club_number)),
    { returning: false }
);

//fill dogs experts estimates
let dogs_experts_estimates: any[] = createDogsExpertsEstimates();

await models.DogExpertEstimate.bulkCreate(
    dogs_experts_estimates.map((dog_expert_estimate) => (dog_expert_estimate)),
    { returning: false }
);
```

```
function getPassport(): string {
    return faker.finance.accountNumber({length: 4}) + ' ' + faker.finance.accountNumber({length: 6})
}

function uniqueValues(): any {
    const unique_values = {
        'breed_names': faker.helpers.uniqueArray(faker.animal.dog, COUNT),
        'passports': faker.helpers.uniqueArray(getPassport, COUNT)
    };
    return unique_values;
}

function createRings(): any[] {
    let rings = [];
    for (let i = 1; i <= COUNT; i++){
        rings.push(i)
    }
    return rings;
}

function createBreeds(): any[] {
    let breeds = [];
    for (let breed_name of UNIQUE_VALUES.breed_names){
        breeds.push({
            'breed_name': breed_name,
            'ring_number': faker.number.int({min: 1, max: COUNT})
        })
    }
    return breeds;
}

function createClub(): any {
    return {
        'club_name': faker.company.name(),
        'participants_number': faker.number.int({min: 0, max: Math.round(COUNT/3)})
    };
}
```

```
function createExpert(): any {
    return {
        'club_id': faker.number.int({min: 1, max: COUNT}),
        'ring_number': faker.number.int({min: 1, max: COUNT}),
        'surname': faker.person.lastName(),
        'name': faker.person.firstName()
    }
}

function createOwners(): any[] {
    let owners = []
    for (let passport of UNIQUE_VALUES.passports){
        owners.push({
            'passport': passport,
            'surname': faker.person.lastName(),
            'name': faker.person.firstName(),
            'patronymic': faker.person.middleName()
        })
    }
    return owners;
}

function createDog(): any {
    //choise owner passport
    return {
        'owner_passport': faker.helpers.arrayElement(UNIQUE_VALUES.passports),
        'breed_name':  faker.helpers.arrayElement(UNIQUE_VALUES.breed_names),
        'pedigree_document_number': faker.finance.accountNumber({length: 7}),
        'mother_nickname': faker.person.firstName('female'),
        'father_nickname': faker.person.firstName('male'),
        'nickname': faker.person.firstName(),
        'age': faker.number.int({min: 1, max: 25}),
        'vaccination_date': faker.date.recent()
    }
}
```

```
function createClubNumbers(): any {
    let club_numbers = [];
    let dog_numbers_shuffled = [];
    for (let i = 1; i <= COUNT; i++) {
        dog_numbers_shuffled.push(i)
    }
    dog_numbers_shuffled = faker.helpers.shuffle(dog_numbers_shuffled);

    for (let i = 1; i <= COUNT; i++){
        club_numbers.push({
            'club_id': i,
            'dog_number': dog_numbers_shuffled[i-1]
        })
    }
    return club_numbers;
}

function createDogsExpertsEstimates(): any {
    let dogs_experts_estimates = [];
    let dog_numbers_shuffled = [];
    for (let i = 1; i <= COUNT; i++) {
        dog_numbers_shuffled.push(i)
    }
    dog_numbers_shuffled = faker.helpers.shuffle(dog_numbers_shuffled);

    for (let i = 1; i <= COUNT; i++){
        dogs_experts_estimates.push({
            'dog_number': dog_numbers_shuffled[i-1],
            'expert_id': i,
            'estimate': faker.number.int({min: 0, max: 10})
        })
    }
    return dogs_experts_estimates;
}
```

Рис 1. Fill_db.ts

2. После был изменен файл db.ts в котором создается объект Sequelize (убрано логирование для уменьшения вывода в консоль).

```
import { Sequelize } from 'sequelize-typescript';
import * as dotenv from 'dotenv';


dotenv.config();
export const db: Sequelize = new Sequelize(
    {
        database: process.env.DB_NAME,
        username: process.env.DB_USER,
        password: process.env.DB_PASSWORD,
        dialect: process.env.DB_DIALECT,
        host: process.env.DB_HOST,
        port: process.env.DB_PORT,
        logging: false
    }
)
```

Рис 2. Db.ts

3. После изменен файл запросов requests.ts, для замера времени работы запросов. (Запросы 1 и 5 реализованы с помощью ORM findAll, а запросы 2, 3, 4

реализованы с помощью ORM RAW SQL. В 5м запросе не используется JOIN, а в 3м используется WITH)

4. Тестирование для 1го запроса

	100	1000	10000	100000
NO ORDER	8.855ms	8.658ms	8.956ms	9.924ms
ORDER	10.866ms	11.318ms	11.365ms	12.775ms

5. Тестирование для 2го запроса

	100	1000	10000	100000
NO ORDER	1.783ms	1.967ms	1.434ms	1.948ms
ORDER	3.371ms	1.382ms	1.978ms	3.098ms

6. Тестирование для 3го запроса

	100	1000	10000	100000
NO ORDER	3.841ms	10.153ms	78.01ms	1.105s
ORDER	3.097ms	9.891ms	94.89ms	1.165s

7. Тестирование для 4го запроса

	100	1000	10000	100000
NO ORDER	1.528ms	2.904ms	28.692ms	95.421ms
ORDER	2.018ms	2.911ms	32.75ms	121.788ms

8. Тестирование для 5го запроса

	100	1000	10000	100000
NO ORDER	2.253ms	2.555ms	9.773ms	41.704ms
ORDER	2.463ms	2.871ms	7.161ms	43.876ms

9. Оценка

- Как можно заметить, запросы, реализуемые через RAW SQL, выполняются бытрее как при сортировке, так и без нее (если не используют WITH).
- Сортировка в среднем увеличивает время выполнения запроса
- Запросы, использующие WITH, самые долгие
- Запросы использующие JOIN в среднем выполняются дольше

10. Далее были добавлены индексы во все модели и проведены тестирования всех запросов повторно

11. Тестирование для 1го запроса

	100	1000	10000	100000
NO ORDER	8.973ms	8.96ms	10.14ms	10.65ms
ORDER	9.75ms	9.814ms	10.229ms	11.039ms

12. Тестирование для 2го запроса

	100	1000	10000	100000
NO ORDER	1.35ms	1.504ms	1.876ms	1.782ms
ORDER	1.725ms	1.787ms	2.035ms	2.77ms

13. Тестирование для 3го запроса

	100	1000	10000	100000
NO ORDER	3.476ms	8.843ms	70.72ms	1.210s
ORDER	3.032ms	9.579ms	75.388ms	1.177s

14. Тестирование для 4го запроса

	100	1000	10000	100000
NO ORDER	1.172ms	3.635ms	25.08ms	70.085ms
ORDER	1.176ms	3.937ms	27.008ms	73.248ms

15.Тестирование для 5го запроса

	100	1000	10000	100000
NO ORDER	2.448ms	2.649ms	5.576ms	34.433ms
ORDER	1.417ms	2.938ms	5.605ms	35.692ms

16.Оценка

- Как можно заметить, запросы с использованием индексов играют значительную роль на больших данных, а на малом не имеет особого значения
- Оценки для тестов без индексов также применимы к тестам с индексами

17. По всем тестам присутствуют выбросы, являющиеся результатом случайного наполнения базы данных.

Ссылка на PR см. в приложении А.

Выводы.

Проведено нагрузочное тестирование на PostgreSQL с помощью ORM. По итогам тестовом сделаны соответствующие выводы.

ПРИЛОЖЕНИЕ А

ССЫЛКИ

https://github.com/moevm/sql-2023-1304/pull/70