Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа №3

по дисциплине "Технологии разработки веб приложений"

ТЕМА РАБОТЫ:

Изучение различных типов БД

Выполнил:

студент гр. ПРИ-120

Парахин К.В.

Принял:

преподаватель кафедры ИСПИ

Шамышев А.А

Владимир 2024 г

Цель работы:

Изучить различные типы баз данных.

Задание:

В ходе выполнения работы вам необходимо протестировать скорость работы двух различных СУБД (PostgreSQL и Redis).

Описание задания:

1. Вам необходимо выбрать сущности и их атрибуты, которые вы будете хранить в БД (необходимо выбрать как минимум три связанные сущности с пятью атрибутами);
2. Вам необходимо разработать схему БД для каждой СУБД в отдельности;
3. Вам необходимо создать БД и записать в неё тестовые данные.
4. Вам необходимо программно выполнить различные запросы к БД и измерить скорость их выполнения;

Примечания:

1. Лучше всего проводить измерения на заполненной базе данных (около 1 - 10 млн. записей каждой сущности). Тестовые данные можно сгенерировать случайным образом или воспользоваться открытым датасетом;
2. Перед измерением необходимо оптимизировать каждую СУБД для работы (см. ссылки внизу);
3. Запросы должны быть разного формата: чтение, запись, удаление, чтение связанных сущностей (если СУБД не позволяет делать JOIN, то их реализацию необходимо сделать вручную на том языке, с помощью которого происходит измерение скорости);
4. Если один запрос выполняться быстрее выбранного инструмента измерения скорости, то можно выполнить несколько однотипных запросов подряд и измерить общее время выполнения всех запросов, а потом, поделив на количество запросов, узнать среднее время выполнения одного запроса.
5. СУБД PostgreSQL можно заменить другой реляционной СУБД;
6. СУБД Redis можно заменить другой СУБД класса NoSQL.

Выполнение работы:

Для тестирования я создал Postgre SQL базу данных admission db, состоящую из 3 таблиц (abiturients, directions и abiturient\_direction\_links) – где третья таблица имеет внешние ключи на таблицы абитуриентов и направлений:

BEGIN;

CREATE TYPE admission\_status AS ENUM (

    'request\_in\_progress',

    'enrolled',

    'enrolled\_on\_other',

    'recommended\_on\_other',

    'failed');

CREATE TABLE abiturients(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    email text NOT NULL,

    password text NOT NULL,

    token text NOT NULL,

    is\_admin BOOLEAN DEFAULT FALSE,

    first\_name text NOT NULL,

    second\_name text NOT NULL,

    has\_diplom\_original boolean NOT NULL DEFAULT FALSE,

    is\_requested boolean NOT NULL DEFAULT FALSE,

    is\_enrolled boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_abiturients PRIMARY KEY (id));

CREATE TABLE directions(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    caption text NOT NULL,

    budget\_places\_number INT NOT NULL,

    min\_ball INT NOT NULL DEFAULT 50,

    is\_filled boolean NOT NULL DEFAULT FALSE,

    is\_finalized boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_directions PRIMARY KEY (id));

CREATE TABLE abiturient\_direction\_links(

    id integer GENERATED BY DEFAULT AS IDENTITY,

    abiturient\_id integer NOT NULL,

    direction\_id integer NOT NULL,

    place integer NOT NULL DEFAULT 0,

    mark integer NOT NULL DEFAULT 0,

    admission\_status admission\_status NOT NULL DEFAULT 'request\_in\_progress',

    prioritet\_number integer NOT NULL DEFAULT 1,

    has\_original\_diplom boolean NOT NULL DEFAULT FALSE,

    created\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    updated\_at VARCHAR(100) NOT NULL DEFAULT '2024-09-15 13:02:56',

    CONSTRAINT pk\_abiturient\_directions\_link PRIMARY KEY (id),

    CONSTRAINT fk\_abiturient\_directions\_link\_abiturients\_abiturient\_id

        FOREIGN KEY (abiturient\_id) REFERENCES abiturients (id)

            ON DELETE CASCADE,

    CONSTRAINT fk\_abiturient\_directions\_link\_directions\_direction\_id

        FOREIGN KEY (direction\_id) REFERENCES directions (id)

            ON DELETE CASCADE);

CREATE OR REPLACE FUNCTION check\_duplicate\_abiturient\_direction()

RETURNS TRIGGER AS $$ BEGIN

IF EXISTS ( SELECT 1 FROM abiturient\_directions\_link

WHERE abiturient\_id = NEW.abiturient\_id AND direction\_id = NEW.direction\_id )

THEN RAISE EXCEPTION

    'Duplicate entry for abiturient\_id % and direction\_id %',

    NEW.abiturient\_id,

    NEW.direction\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_duplicate\_abiturient\_direction

BEFORE INSERT ON abiturient\_direction\_links

FOR EACH ROW

EXECUTE FUNCTION check\_duplicate\_abiturient\_direction();

CREATE OR REPLACE FUNCTION restrict\_updates()

RETURNS TRIGGER AS $$ BEGIN

IF NEW.abiturient\_id != OLD.abiturient\_id

    OR NEW.direction\_id != OLD.direction\_id

    OR NEW.place != OLD.place

    OR NEW.has\_original\_diplom != OLD.has\_original\_diplom

THEN RAISE EXCEPTION 'Only mark, admission\_status, and prioritet\_number fields can be updated';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER restrict\_fields\_update

BEFORE UPDATE ON abiturient\_direction\_links

FOR EACH ROW

EXECUTE FUNCTION restrict\_updates();

CREATE OR REPLACE FUNCTION update\_has\_original\_diplom()

RETURNS TRIGGER AS $$ BEGIN

IF NEW.has\_diplom\_original != OLD.has\_diplom\_original

THEN

UPDATE abiturient\_direction\_links

    SET has\_original\_diplom = NEW.has\_diplom\_original,

        updated\_at = CURRENT\_TIMESTAMP

    WHERE abiturient\_id = NEW.id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger\_update\_has\_original\_diplom

AFTER UPDATE OF has\_diplom\_original

ON abiturients

FOR EACH ROW

EXECUTE FUNCTION update\_has\_original\_diplom();

ALTER TABLE abiturient\_direction\_links RENAME COLUMN has\_original\_diplom TO has\_diplom\_original;

COMMIT;

В качестве NoSQL базы данных я взял Redis – в нем он хранит по ключам (идентификаторам пользователя) информацию о его фикстуре в json – формате  
  
Пример json фикстуры в Redis:

"id": 1,

"value": {

        "first\_name": "Кирилл",

        "second\_name": "Парахин",

        "email": "parahinkv@gmail.com",

        "has\_diplom\_original": true,

        "is\_enrolled": false,

        "directions\_links": [

            {

                "direction\_id": 1,

                "direction\_caption": "PRI",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 1

            },

            {

                "direction\_id": 2,

                "direction\_caption": "IST",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 2

            },

            {

                "direction\_id": 3,

                "direction\_caption": "VT",

                "place": 0,

                "mark": 0,

                "admission\_status": "request\_in\_progress",

                "priotitet\_number": 3

            }

        ]

    },

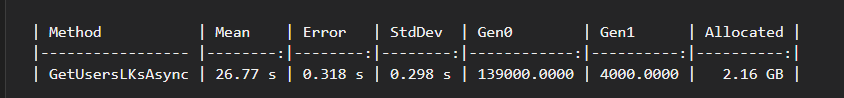
Далее попробуем протестировать скорость работы CRUD операций на примере данных схем данных в 2 СУБД: PostgreSQL и Redis  
  
Для этого я буду использовать BenchmarkDotNet.

Будет в обоих базах такой сценарий:

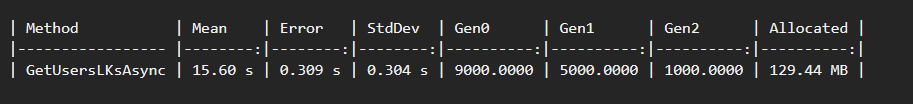
Сначала производим вставку миллиона пользователей, затем вставляем 100 тысяч позиций пользователей в таблицу, ссылающуюся на направления (это для Postgres, по аналогии заполняем и Redis базу)

Сам тест производительности будет связан со чтениеим данных о поданных направлениях для сотни тысяч пользователей (PostgreSQL и Redis – в случае Postgre это будет JOIN запросы на смежные таблицы, в случае Redis – просто чтение по ключам).  
  
Результаты:

1. Postgres



1. Redis



Как можно заметить, для того же кол-ва операция чтения (100 000) – redis в среднем затрачивал 15 секунд, а Postgres – 26 секунд. Но еще сильнее отличается то, сколько памяти было выделено во время исполнения одного теста – в случае Redis это всего 130 мб, а в случае Postgres – 2,16 гб  
Можно сделать вывод, что для целей быстрого доступа к данным пользователя (в данном случае тесты на обоих базах так или иначе получали фикстуру личного кабинета пользователя – но Postgres делал JOIN запросы на SQL уровне, в отличие от Redis – который просто возвращал json-тело по ключу) более оптимизированно будет использовать NoSQL распределенный кэш Redis.  
Но в целом для постоянного хранения лучше подойдет Postgres – так как его дизайн лучше обеспечивает консистентное ссылочное хранение большого количества связанных данных.

Листинг бэнчмарка для Postgres:

[MemoryDiagnoser]

public class BenchmarkPostgres

{

    [GlobalSetup]

    public async Task SetupAsync()

    {

        var services = new ServiceCollection();

        AddPostgresStorage(services);

        \_provider = services.BuildServiceProvider();

        using var scope1 = \_provider.CreateScope();

        var uow1 = scope1.ServiceProvider.GetRequiredService<IAbiturientUnitOfWork>();

        for (int i = 0; i < 1\_000; i++)

        {

            await uow1.AbiturientsRepository.AddAsync(

                new()

                {

                    FirstName = "Kirill",

                    SecondName = "Parakhin",

                    Email = "test mail",

                    Password = "1111",

                    Token = "guid",

                    IsAdmin = false,

                    IsEnrolled = false,

                    IsRequested = false,

                    HasDiplomOriginal = true

                }, CancellationToken.None);

        }

        await uow1.DirectionsRepository.AddAsync(

            new()

            {

                Caption = "PRI",

                BudgetPlacesNumber = 8,

                MinBall = 60,

                IsFilled = true,

                IsFinalized = false

            }, CancellationToken.None);

        await uow1.SaveAsync(CancellationToken.None);

        await uow1.DisposeAsync();

        using var scope2 = \_provider.CreateScope();

        var uow2 = scope2.ServiceProvider.GetRequiredService<IAbiturientUnitOfWork>();

        for (int i = 0; i < 100; i++)

        {

            await uow2.AbiturientDirectionsRepository.AddAsync(

                new()

                {

                    AbiturientId = i + 1,

                    DirectionId = 1,

                    //AdmissionStatus = AdmissionStatus.RequestInProgress,

                    Mark = 100,

                    Place = i + 1,

                    PrioritetNumber = 1,

                    HasDiplomOriginal = true

                }, CancellationToken.None);

        }

        await uow2.SaveAsync(CancellationToken.None)

        await uow2.DisposeAsync();

    }

    [Benchmark]

    public async Task GetUsersLKsAsync()

    {

        \_unitOfWork = \_provider.GetRequiredService<IAbiturientUnitOfWork>();

        for (int i = 0; i < 100\_000; i++)

        {

            await \_unitOfWork.AbiturientsRepository.GetUserLKAsync(i, CancellationToken.None);

        }

        await \_unitOfWork.DisposeAsync();

    }

    private static IServiceCollection AddPostgresStorage(

        IServiceCollection services)

        => services

            .AddDbContext<AdmissionContext>(

                opt =>

                {

                    var dataSourceBuilder = new NpgsqlDataSourceBuilder(

                        "Host=localhost;Port=5432;Database=admission\_test\_db;" +

                        "Username=postgres;Password=root;");

                    \_ = dataSourceBuilder.MapEnum<AdmissionStatus>("admission\_status");

                    var source = dataSourceBuilder.Build();

                    opt.UseNpgsql(source);

                })

            .AddScoped<IAbiturientUnitOfWork, AbiturientUnitOfWork>()

            .AddScoped<IRepositoryContext, RepositoryContext>()

            .AddScoped<IAbiturientsRepository, AbiturientsRepository>()

            .AddScoped<IDirectionsRepository, DirectionsRepository>()

            .AddScoped<IAbiturientDirectionsRepository, AbiturientDirectionsRepository>();

    private static IServiceCollection AddDb1Context(

        IServiceCollection services)

    {

        var dataSourceBuilder = new NpgsqlDataSourceBuilder();

        \_ = dataSourceBuilder.MapEnum<AdmissionStatus>("admission\_status");

        var source = dataSourceBuilder.Build();

        return services.AddDbContext<AdmissionContext>(

            opt => opt.UseNpgsql(source));

    }

    private IServiceProvider \_provider;

    private IAbiturientUnitOfWork \_unitOfWork;

}

Листинг бэнчмарка для Redis:

[MemoryDiagnoser]

public class BenchmarkRedis

{

    [GlobalSetup]

    public async Task SetupAsync()

    {

        var services = new ServiceCollection();

        AddRedisCache(services);

        var provider = services.BuildServiceProvider();

        \_userLKRepository = provider.GetRequiredService<IUserLKRepository>();

        for (int i = 0; i < 100\_000; i++)

        {

            await \_userLKRepository.AddAsync(

                i,

                new()

                {

                    UserId = i + 1,

                    FirstName = "Kirill",

                    SecondName = "Parakhin",

                    Email = "test mail",

                    IsEnrolled = true,

                    DirectionLinks =

                    [

                        new()

                        {

                            DirectionId = 1,

                            Mark = 100,

                            Place = i + 1,

                            PrioritetNumber = 1,

                            //AdmissionStatus = Storage.Models.Postgres.AdmissionStatus.RequestInProgress

                        }

                    ]

                }, CancellationToken.None);

        }

    }

    [Benchmark]

    public async Task GetUsersLKsAsync()

    {

        await \_userLKRepository.GetUserLKs(

            Enumerable.Range(1, 100\_000).Select(Convert.ToInt64).ToList(),

            CancellationToken.None);

    }

    private static IServiceCollection AddRedisCache(

        IServiceCollection services)

        => services

            .AddSingleton<IConnectionMultiplexer, ConnectionMultiplexer>(

                \_ =>

                {

                    return ConnectionMultiplexer.Connect("localhost:6379");

                })

            .AddScoped<IUserLKRepository, UserLKRepository>();

    private IUserLKRepository \_userLKRepository;

}

Вывод:

В результате выполнения работы были освоены принципы работы с SQL и NoSQL базами данных, изучены и применены способы проверки производительности методы работы с БД на примере PostgreSQL и Redis.