Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Базы данных

Отчет по лабораторной работе №2

Работу выполнил:
Раскин Андрей
Группа: 43501/3
Преподаватель:
Мяснов А.В.

Санкт-Петербург
2017

1 Цель работы

Получить практические навыки работы с БД путём создания собственного генератора тестовых данных на языке Java.

2 Ход выполнения работы

2.1 Окружение

При разработке использовался язык Java 8. Для описания сущностей базы данных, как объектов языка использовалась технология JPA. JPA (Java Persistence API) это спецификация Java EE и Java SE, описывающая систему управления сохранением java объектов в таблицы реляционных баз данных в удобном виде. Сама Java не содержит реализации JPA, однако есть существует много реализаций данной спецификации от разных компаний (открытых и нет). Это не единственный способ сохранения java объектов в базы данных (ORM систем), но один из самых популярных в Java мире. Для сборки проекта используется Gradle - система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но предоставляющая DSL на языке Groovy вместо традиционной XML-образной формы представления конфигурации проекта.

Для PostgreSQL была создана база данных clinic_db, пользователь использовался стандартный - **postgres**. Для подключения к базе данных необходимо указать хост, порт, имя базы данных и логин/-пароль в специальном файле настроек для Ebean.

2.2 Создание генератора

Заполнение объекта в Java случайными данными на первый взгляд выглядит достаточно просто. Но при более близком рассмотрении, если например модель БД включает множество связанных классов сложность возрастает. Для генерации случайных данных использовался EnhancedRandom API. Это библиотека, предоставляющая методы для настраеваемой генерации различных типов даных. Часть данных генерируется путём взятия случайно строчки из файла. Такой метод используется в тех случаях, когда полноценная случайность не важна: имена, названия лекарств и тп.

```
public static String choose(File f) throws FileNotFoundException
    {
        String result = null;
        Random rand = new Random();
        int n = 0;
        for(Scanner sc = new Scanner(f); sc.hasNext(); )
        {
            ++n;
            String line = sc.nextLine();
            if(rand.nextInt(n) == 0)
                result = line;
        }

        return result;
    }
```

В остальных случаях используется генерирование с помощью EnhancedRandom API.

```
private boolean generate() throws FileNotFoundException {
        File names = new File("names");
        File cities = new File("cities");

        Supplier<Patients.Gender> genderSupplier = () -> {
            switch ((int) (Math.random() * 2 + 1)) {
                case 1: return Patients.Gender.FEMALE;
                case 2: return Patients.Gender.MALE;
                default: return Patients.Gender.MALE;
            }
        };

        random = EnhancedRandomBuilder.aNewEnhancedRandomBuilder()
                .randomize(FieldDefinitionBuilder.field()
                        .named("name").ofType(String.class).get(), (Randomizer<String>) ()
                            -> {
                            try {
                                return choose(names);
                            } catch (FileNotFoundException e) {
                                e.printStackTrace();
```

```
                            }
                            return null;
                        })
                .randomize(FieldDefinitionBuilder.field()
                        .named("birthDate").ofType(LocalDate.class).get(), new
                            ↪ LocalDateRandomizer())
                .randomize(FieldDefinitionBuilder.field()
                        .named("gender").ofType(Patients.Gender.class).get(), genderSupplier
                            ↪ )
                .randomize(FieldDefinitionBuilder.field()
                        .named("city").ofType(String.class).get(), (Randomizer<String>) ()
                            ↪ -> {
                    try {
                        return choose(cities);
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    }
                    return null;
                })
                .exclude(FieldDefinitionBuilder.field()
                    .named("payments").get())
                .exclude(FieldDefinitionBuilder.field()
                    .named("treatments").get())
                .stringLengthRange(5, 50)
                .build();
        return true;
    }
```

Данные генерируются для всех таблиц пропорционально их собственному коэффициенту. Например, при генерации в таблице болезней N строк, в таблице лекарств будет сгенерировано $2N$ строк, в таблице назначений будет сгенерировано $4N$ строк и так далее для каждой таблице. Такой метод генерации сохраняет логику базы данных, ведь количество назначений в карточках обычно больше количества лекарств, а количество лекарств обычно больше количества болезней.

Кроме пропорционального заполнения, генератор обеспечивает логическую целостность данных в отдельных таблицах. Это означает, например, что таблицы показаний и противопоказаний не содержат одинаковых пар лекарство + болезнь; таблица несовместимости лекарств не содержит повторяющихся значений; цены, даты и количества лекарств содержатся в определенных границах и др.

```
public class LoadExampleData {

    private static boolean runOnce;

    private static EbeanServer server = Ebean.getServer(null);

    private static PatientsGenerator patientGen;
    private static DepartmentsGenerator departGen;
    private static DiseasesGenerator disGen;
    private static DiseasesTypesGenerator disTypesGen;
    private static DoctorsGenerator doctorsGen;
    private static DrugsGenerator drugsGen;
    private static GrantsGenerator grantsGen;
    private static PaymentsGenerator paymentsGen;
    private static ServicesGenerator servicesGen;
    private static TreatmentGenerator treatmentGen;

    public static synchronized void load(int num) throws IOException {
        patientGen = new PatientsGenerator();
        departGen = new DepartmentsGenerator();
        disGen = new DiseasesGenerator();
        disTypesGen = new DiseasesTypesGenerator();
        doctorsGen = new DoctorsGenerator();
        drugsGen = new DrugsGenerator();
        grantsGen = new GrantsGenerator();
        paymentsGen = new PaymentsGenerator();
        servicesGen = new ServicesGenerator();
        treatmentGen = new TreatmentGenerator();

        if (runOnce) {
            return;
        }

        final LoadExampleData me = new LoadExampleData();

        server.execute(me::deleteAll);
        generateSome(num);
```

```
38          runOnce = true;
39      }
40
41      private void deleteAll() {
42          Ebean.execute(() -> {
43              server.createUpdate(Departments.class, "delete from departments").execute();
44              server.createUpdate(Diseases.class, "delete from diseases").execute();
45              server.createUpdate(Doctors.class, "delete from doctors").execute();
46              server.createUpdate(Drugs.class, "delete from drugs").execute();
47              server.createUpdate(DiseasesTypes.class, "delete from diseasesTypes").execute();
48              server.createUpdate(Grants.class, "delete from grants").execute();
49              server.createUpdate(Patients.class, "delete from patients").execute();
50              server.createUpdate(Services.class, "delete from services").execute();
51              server.createUpdate(Treatment.class, "delete from treatment").execute();
52          });
53      }
54
55      private static boolean generateSome(int genAmount) {
56          IntegerRangeRandomizer intRandomizer = new IntegerRangeRandomizer(0, genAmount);
57
58          server.execute(() -> {
59              List<Departments> departments = new ArrayList<>();
60              for (int i = 0; i < genAmount/10; i++) {
61                  Departments o = new Departments(departGen.random.nextObject(Departments.
                        ↪ class));
62                  departments.add(o);
63                  o.save();
64
65              }
66              System.out.println(departments);
67
68              List<Doctors> doctors = new ArrayList<>();
69              for (int i = 0; i < genAmount; i++) {
70                  Doctors o = new Doctors(doctorsGen.random.nextObject(Doctors.class));
71                  o.setDepartId(departments.get(new IntegerRangeRandomizer(0, genAmount/10).
                        ↪ getRandomValue()));
72                  doctors.add(o);
73                  o.save();
74              }
75              System.out.println(doctors);
76
77              List<Patients> patients = new ArrayList<>();
78              for (int i = 0; i < genAmount*5; i++) {
79                  Patients o = new Patients(patientGen.random.nextObject(Patients.class));
80                  patients.add(o);
81                  o.save();
82
83              }
84              System.out.println(patients);
85
86              List<DiseasesTypes> diseasesTypes = new ArrayList<>();
87              for (int i = 0; i < genAmount; i++) {
88                  DiseasesTypes o = new DiseasesTypes(disTypesGen.random.nextObject(
                        ↪ DiseasesTypes.class));
89                  diseasesTypes.add(o);
90                  o.save();
91
92              }
93              System.out.println(diseasesTypes);
94
95              List<Diseases> diseases = new ArrayList<>();
96              for (int i = 0; i < genAmount; i++) {
97                  Diseases o = new Diseases(disGen.random.nextObject(Diseases.class));
98                  o.setDisType(diseasesTypes.get(intRandomizer.getRandomValue()));
99                  diseases.add(o);
100                 o.save();
101             }
102             System.out.println(diseases);
103
104             List<Drugs> drugs = new ArrayList<>();
105             for (int i = 0; i < genAmount; i++) {
106                 Drugs o = new Drugs(drugsGen.random.nextObject(Drugs.class));
107                 o.setTypeId(diseasesTypes.get(intRandomizer.getRandomValue()));
108                 switch ((int) (Math.random() * 3 + 1)) {
109                     case 1: o.save();
110
```

```
                    case 2: o.save();

                    case 3: {
                        if (drugs.isEmpty()) {
                            break;
                        }
                        Set<Drugs> drugRestr = new HashSet<>();
                        drugRestr.add(drugs.get(new IntegerRangeRandomizer(0, drugs.size()
                            ↪ -1).getRandomValue()));
                        o.setRestrictionsColl(drugRestr);
                        o.save();
                    }
                    default: o.save();
                }
                drugs.add(o);
            }
            System.out.println(drugs);

            List<Services> services = new ArrayList<>();
            for (int i = 0; i < genAmount; i++) {
                Services o = new Services(servicesGen.random.nextObject(Services.class));
                o.save();
                services.add(o);
            }
            System.out.println(services);

            List<Grants> grants = new ArrayList<>();
            for (int i = 0; i < genAmount; i++) {
                Grants o = new Grants(grantsGen.random.nextObject(Grants.class));
                Drugs drug = drugs.get(intRandomizer.getRandomValue());
                Services service = services.get(intRandomizer.getRandomValue());
                o.setSum(Integer.toUnsignedLong(drug.getPrice() + service.getPrice()));
                o.setDoctor(doctors.get(intRandomizer.getRandomValue()));
                o.setPatient(patients.get(new IntegerRangeRandomizer(0, genAmount*5).
                    ↪ getRandomValue()));
                o.setDrug(drug);
                o.setService(service);
                o.save();
                grants.add(o);
            }
            System.out.println(grants);

            List<Payments> payments = new ArrayList<>();
            for (int i = 0; i < genAmount; i++) {
                Payments o = new Payments(paymentsGen.random.nextObject(Payments.class));
                Set<Patients> patientsTo = new HashSet<>();
                patientsTo.add(patients.get(new IntegerRangeRandomizer(0, genAmount*5).
                    ↪ getRandomValue()));
                o.setPatients(patientsTo);
                o.save();
                payments.add(o);
            }
            System.out.println(payments);

            List<Treatment> treatments = new ArrayList<>();
            for (int i = 0; i < genAmount * 10; i++) {
                Treatment o = new Treatment(treatmentGen.random.nextObject(Treatment.class))
                    ↪ ;

                o.setPatientId(patients.get(new IntegerRangeRandomizer(0, genAmount*5).
                    ↪ getRandomValue()));
                o.setDoctorId(doctors.get(intRandomizer.getRandomValue()));
                o.setDiseaseId(diseases.get(intRandomizer.getRandomValue()));

                Integer drugsToNum = intRandomizer.getRandomValue();
                Set<Drugs> drugsTo = new HashSet<>();
                for (int j = 0; j < drugsToNum; j++) {
                    Drugs drug = drugs.get(intRandomizer.getRandomValue());
                    while (checkRestriction(drug, drugsTo)) {
                        drug = drugs.get(intRandomizer.getRandomValue());
                    }
                    drugsTo.add(drugs.get(intRandomizer.getRandomValue()));
                }
                o.setDrugs(drugsTo);

                Integer servicesToNum = intRandomizer.getRandomValue();
```

```
182                 Set<Services> servicesTo = new HashSet<>();
183                 for (int j = 0; j < servicesToNum; j++) {
184                     servicesTo.add(services.get(intRandomizer.getRandomValue()));
185                 }
186                 o.setServices(servicesTo);
187                 o.save();
188                 treatments.add(o);
189             }
190             System.out.println(treatments);
191 
192         });
193         return true;
194     }
195 
196     public static boolean checkRestriction(Drugs drug1, Set<Drugs> drug2) {
197         for (Drugs dr: drug2) {
198             if (dr.equals(drug1)) {
199                 return true;
200             }
201         }
202         return false;
203     }
204 }
```

3 Выводы

В данной работы было проведено знакомство с библиотекой Enchanced Random для Java, позволяющей наполнять объекты случайными данными. Написание собственного генератора намного более гибкое решение, чем добавление данных вручную. Это обусловлено тем, что при тестировании нас обычно не волнуют точные значения имен, цен и прочих параметров, в то время как пропорции данных между таблицами, контроль повторных значений и неявные зависимости между таблицами важны при тестировании.