САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе № 3

Дисциплина: «Базы данных»

Тема: «Генерация тестовых данных»

Выполнил студент гр. 43501/3	____________ (подпись)	А.Ю. Ламтев
Преподаватель	____________ (подпись)	А.В. Мяснов

«___» ___________ 2019 г.

Санкт-Петербург
2019

Содержание

1. Цели работы

Сформировать набор данных, позволяющий производить операции на реальных объемах данных.

2. Программа работы

1. Реализация в виде программы параметризуемого генератора, который позволит сформировать набор связанных данных в каждой таблице.
2. Частные требования к генератору, набору данных и результирующему набору данных:
   - количество записей в справочных таблицах должно соответствовать ограничениям предметной области
   - количество записей в таблицах, хранящих информацию об объектах или субъектах должно быть параметром генерации
   - значения для внешних ключей необходимо брать из связанных таблиц

3. Разработка генератора

Генератор выполнен в виде консольного приложения, разработанного на языке `Java` последней версии `11.0.1`. Программа ожидает 3 аргумента командной строки: путь к файлу в формате `json`, в котором содержатся `url Postgres`-сервера, и имя пользователя, и пароль для доступа к нему; и путь к файлу в формате `json`, содержащему параметры генератора (обязательные параметры); и значение из множества `{onlyMovies, onlySeries, onlyUsers, onlyMovieSubscriptions, onlySeriesSubscriptions}`, которое позволяет догенерировать определённые данные (если этот аргумент отсутствует, то генерируются данные для всех таблиц). Примеры этих 2-х файлов представлены в листингах 1 и 2.

```json
{
  "url": "jdbc:postgresql://localhost:5432/postgres",
  "user": "postgres",
  "password": "postgres"
}
```

Листинг 1: Пример параметров доступа к `Postgres`-серверу

```json
{
  "usersCount": 10000,
  "femalePercentage": 55,
  "moviesCount": 25000,
  "seriesCountSeasonsEpisodes": [
    [100, 3, 15],
    [150, 4, 50],
    [200, 2, 25],
    [300, 1, 7],
    [100, 5, 10]
  ],
  "percentageOfUsersWhoBoughtMovies": 64,
  "minMoviesPerUser": 5,
  "maxMoviesPerUser": 100,
  "percentageOfUsersWhoBoughtSeries": 35,
  "minSeriesPerUser": 2,
  "maxSeriesPerUser": 100,
  "yearsSinceFirstSubscription": 5,
  "minSubscriptionsPerUser": 7,
  "maxSubscriptionsPerUser": 25,
  "durationPriceNMoviesMSeasons": [
    [30, 5, 5, 1], [60, 9, 5, 1], [90, 13, 5, 1], [180, 25, 5, 1], [365, 40, 5, 1],
    [30, 7, 10, 2], [60, 13, 10, 2], [90, 25, 10, 2], [180, 45, 10, 2], [365, 75, 10, 2],
    [30, 10, 15, 3], [60, 18, 15, 3], [90, 35, 15, 3], [180, 65, 15, 3], [365, 100, 15, 3],
    [30, 12, 30, 5], [60, 21, 30, 5], [90, 40, 30, 5], [180, 70, 30, 5], [365, 120, 30, 5]
    ],
  "moviesSubscriptionsPercentage": 25
}
```

Листинг 2: Пример параметров генератора

Рассмотрим подробнее параметры генератора:

- `usersCount` — число пользователей
- `femalePercentage` — процент девушек от общего числа пользователей
- `moviesCount` — число самостоятельных фильмов (эпизоды сериалов в это число не входят)
- `seriesCountSeasonsEpisodes` — массив типов сериалов, параметризуемый 3-мя значениями: числом сериалов данного типа, числом сезонов в таких сериалах и количество серий в каждом сезоне (`[100, 3, 15]` означает 100 сериалов, в каждом 3 сезона, состоящих из 15 серий)
- `percentageOfUsersWhoBoughtMovies` — процент пользователей, купивших хотя бы 1 фильм на постоянной основе.
- `minMoviesPerUser` — минимальное число фильмов, которые купил пользователь, входящий в группу, описываемую предыдущим параметром.
- `maxMoviesPerUser` — аналогично предыдущему параметру – максимальное число фильмов.
- `percentageOfUsersWhoBoughtSeries` — процент пользователей, купивших хотя бы 1 сериал на постоянной основе.
- `minSeriesPerUser` — минимальное число сериалов, которые купил пользователь, входящий в группу, описываемую предыдущим параметром.
- `maxSeriesPerUser` — аналогично предыдущему параметру – максимальное число сериалов
- `yearsSinceFirstSubscription` — число лет, прошедших с первой подписки
- `minSubscriptionsPerUser` — минимальное число подписок у пользователя

- `maxSubscriptionsPerUser` — максимальное число подписок у пользователя
- `moviesSubscriptionsPercentage` — процент подписок на фильмы от общего числа подписок (на фильмы и сериалы)
- `durationPriceNMoviesMSeasons` — массив типов подписок, параметризуемый 4-мя значениями: длительностью в днях, стоимостью в $, соответствующему числу фильмов и соответствующему числу сезонов сериалов (`[90, 35, 15, 3]` означает, что подписка на 90 дней, стоимостью $35, и в неё входят либо 15 фильмов, либо 3 сериала).

  Для соединения с базой данных используется `JDBC` драйвер последней версии `42.2.5`.

  В качестве системы сборки и управления зависимостями проекта выбран `Gradle` версии `5.0`, конфигурационные файлы проекта написаны на `Kotlin DSL`. Они представлены в листингах 3 и 4.

```
plugins {
    java
}

group = "com.lamtev.movie-service"
version = "1.0.RELEASE"

repositories {
    jcenter()
}

dependencies {
    compile("com.intellij:annotations:12.0")
    compile("org.postgresql:postgresql:42.2.5")
    compile("com.github.javafaker:javafaker:0.16")
    compile("net.sf.trove4j:trove4j:3.0.3")
    compile("com.google.code.gson:gson:2.8.5")
}

configure<JavaPluginConvention> {
    sourceCompatibility = JavaVersion.VERSION_11
}

val fatJar = task("fatJar", type = Jar::class) {
    baseName = "${project.group}.${project.name}"
    manifest {
        attributes["Implementation-Title"] = "Movie service data generator"
        attributes["Implementation-Version"] = version
        attributes["Main-Class"] = "com.lamtev.movie_service.datagen.Launcher"
    }
    from(configurations["compile"].map { if (it.isDirectory) it else zipTree(it) })
    with(tasks["jar"] as CopySpec)
}

tasks {
    "build" {
        dependsOn(fatJar)
    }
}
```

Листинг 3: build.gradle.kts

```
rootProject.name = "datagen"
```

Листинг 4: settings.gradle.kts

Приложение логически разделено на 2 части:

1. **Обработка аргументов командной строки и парсинг конфигурационных файлов**

   Состоит из класса ArgumentsParser с бизнес-логикой, исходный код которого приведён в листинге 6. А также классов EndpointInfo (листинг 7) и Parameters (листинг 8), которые являются моделью для входных json файлов.

   Для десериализации json файлов в объекты классов используется библиотека Gson.

2. **Генерация данных и заполнение ими БД**

   На рис. 3.1 представлена схема БД, состоящей из 16 таблиц.

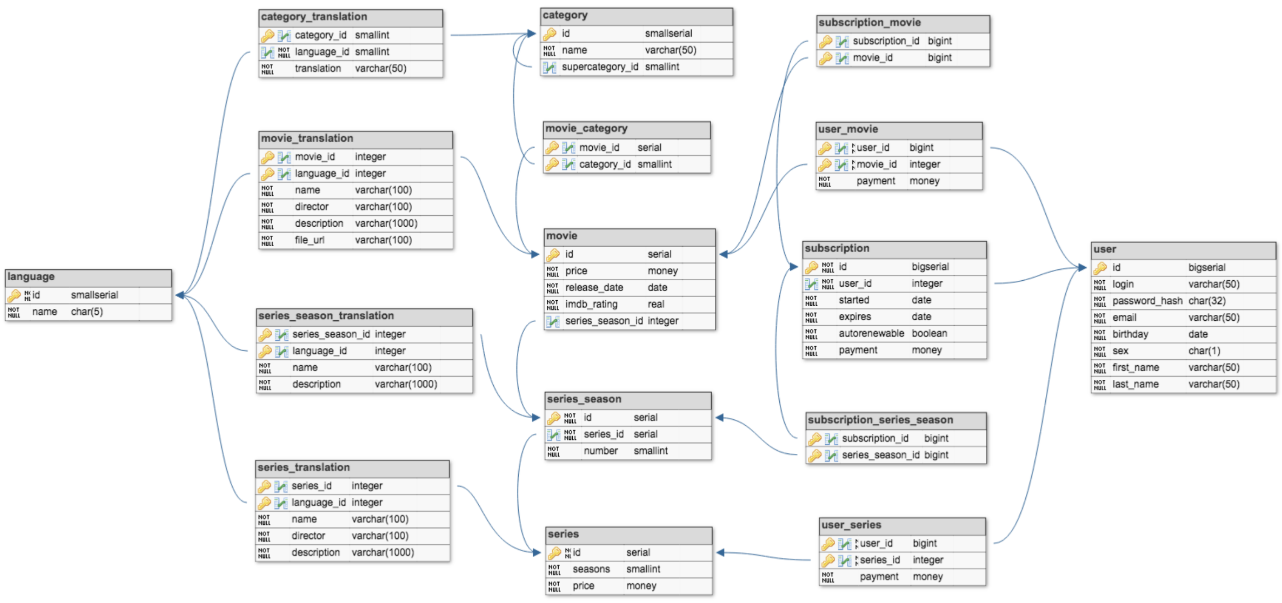

Рис. 3.1: Схема БД

Для заполнения соответствующих таблиц были разработаны классы, реализующие интерфейс TableGenerator (листинг 9):

- LanguageTableGenerator (листинг 10) — генератор данных для таблицы language
- CategoryTableGenerator (листинг 11) — генератор данных для таблицы category
- CategoryTranslationTableGenerator (листинг 12) — генератор данных для таблицы category_translation
- MovieTableGenerator (листинг 13) — генератор данных для таблицы movie

- MovieTranslationTableGenerator (листинг 14) — генератор данных для таблицы movie_translation
- MovieCategoryTableGenerator (листинг 15) — генератор данных для таблицы movie_category
- SeriesTableGenerator (листинг 16) — генератор данных для таблицы series
- SeriesTranslationTableGenerator (листинг 17) — генератор данных для таблицы series_translation
- SeriesSeasonTableGenerator (листинг 18) — генератор данных для таблицы series_season
- SeriesSeasonTranslationTableGenerator (листинг 19) — генератор данных для таблицы series_season_translation
- UserTableGenerator (листинг 20) — генератор данных для таблицы user
- UserMovieTableGenerator (листинг 21) — генератор данных для таблицы user_movie
- UserSeriesTableGenerator (листинг 22) — генератор данных для таблицы user_series
- SubscriptionTableGenerator (листинг 23) — генератор данных для таблицы subscription
- SubscriptionMovieTableGenerator (листинг 24) — генератор данных для таблицы subscription_movie
- SubscriptionSeriesSeasonTableGenerator (листинг 25) — генератор данных для таблицы subscription_series_season

При формировании новых данных иногда требовались данные, уже содержащиеся в таблицах (в частности, значения внешних ключей). Для извлечения из БД этих данных было разработано 2 класса:

- StorageDAO (листинг 26) — класс, в котором реализованы SELECT запросы к базе данных, позволяющие получить число записей в произвольной таблице или получить все первичные ключи таблицы.
- SubscriptionTableDAO (листинг 27) — класс, в котором реализован SELECT запрос, специфичный только для таблицы subscription.

Также был разработан утилитный класс Utils (листинг 28), в котором реализованы вспомогательные функциональности, используемые при генерации данных для разных таблиц.

Для генерации различных данных, таких, как названия фильмов, сериалов, имена пользователей, даты и т.д. использовалась библиотека JavaFaker.

4. Выводы

В результате работы был разработан параметризуемый генератор, с помощью которого БД была заполнена данными. Эти данные состоят из десятков тысяч пользователей; десятков тысяч фильмов; тысяч сериалов, содержащих, десятки тысяч серий; сотен тысяч подписок...

Также был получен опыт организации взаимодействия `Java`-приложений с базами данных с помощью стандарта `JDBC`.

Приложение 1. Исходный код

```java
package com.lamtev.movie_service.datagen;

import com.lamtev.movie_service.datagen.cli_args.ArgumentsParser;
import com.lamtev.movie_service.datagen.generator.LanguageTableGenerator;
import com.lamtev.movie_service.datagen.generator.StorageDAO;
import com.lamtev.movie_service.datagen.generator.Utils;
import com.lamtev.movie_service.datagen.generator.category.CategoryTableGenerator;
import com.lamtev.movie_service.datagen.generator.movie.MovieTableGenerator;
import com.lamtev.movie_service.datagen.generator.series.SeriesTableGenerator;
import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionMovieTableGenerator;
import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionSeriesSeasonTableGenerator;
import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionTableDAO;
import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionTableGenerator;
import com.lamtev.movie_service.datagen.generator.user.UserMovieTableGenerator;
import com.lamtev.movie_service.datagen.generator.user.UserSeriesTableGenerator;
import com.lamtev.movie_service.datagen.generator.user.UserTableGenerator;

import java.sql.DriverManager;
import java.sql.SQLException;

final class Launcher {

    public static void main(String[] args) {
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        final var argumentsParser = new ArgumentsParser(args);
        final var endpoint = argumentsParser.endpoint();
        final var parameters = argumentsParser.parameters();
        if (endpoint == null || parameters == null) {
            System.err.println("Wrong arguments!");
            return;
        }
        try (final var connection = DriverManager.getConnection(endpoint.url(), endpoint.user(), endpoint.password())) {
            if (parameters.isGenAll()) {
                final var language = new LanguageTableGenerator();
                language.updateTableUsing(connection);
                System.out.println("language generated");

                final var category = new CategoryTableGenerator();
                category.updateTableUsing(connection);
                System.out.println("category generated");
            }

            if (parameters.isGenAll() || parameters.isGenMoviesOnly()) {
                final var movie = new MovieTableGenerator(parameters.moviesCount());
                movie.updateTableUsing(connection);
                System.out.println("movie generated");
            }

            if (parameters.isGenAll() || parameters.isGenSeriesOnly()) {
                final var series = new SeriesTableGenerator(parameters.seriesCountSeasonsEpisodes());
                series.updateTableUsing(connection);
                System.out.println("series generated");
            }

            if (parameters.isGenAll() || parameters.isGenUsersOnly()) {
                final var user = new UserTableGenerator(parameters.usersCount(), parameters.femalePercentage());
                user.updateTableUsing(connection);
                System.out.println("user generated");
            }

            if (parameters.isGenAll()) {
                final var userMovie = new UserMovieTableGenerator(parameters.percentageOfUsersWhoBoughtMovies(),
                        parameters.minMoviesPerUser(), parameters.maxMoviesPerUser());
                userMovie.updateTableUsing(connection);
                System.out.println("user_movie generated");

                final var seriesMovie = new UserSeriesTableGenerator(parameters.percentageOfUsersWhoBoughtSeries(),
                        parameters.minSeriesPerUser(), parameters.maxSeriesPerUser());
                seriesMovie.updateTableUsing(connection);
                System.out.println("user_series generated");
            }

            if (parameters.isGenAll() || parameters.isGenSubscriptionsToMoviesOnly()
                    || parameters.isGenSubscriptionsToSeriesOnly()) {
                final var subscription = new SubscriptionTableGenerator(parameters.usersCount(), parameters.minSubscriptionsPerUser(),
                        parameters.maxSubscriptionsPerUser(), parameters.durationPriceNMoviesMSeasons(), parameters.yearsSinceFirstSubscription());
                subscription.updateTableUsing(connection);
                System.out.println("subscription generated");

                final var subscriptionIdsNMoviesMSeasons = SubscriptionTableDAO.instance().idsNMoviesOrMSeasonsContainingInIds(
                        connection, parameters.durationPriceNMoviesMSeasons(), subscription.getGeneratedIds());
                final var subscriptionIdsNMovies = new int[2][0];
                final var subscriptionIdsMSeasons = new int[2][0];
                Utils.split(subscriptionIdsNMoviesMSeasons, parameters.moviesSubscriptionsPercentage(), subscriptionIdsNMovies, subscriptionIdsMSeasons);

                if (parameters.isGenAll() || parameters.isGenSubscriptionsToMoviesOnly()) {
                    final var movieIds = StorageDAO.instance().ids(connection, "movie");
                    final var subscriptionMovie = new SubscriptionMovieTableGenerator(subscriptionIdsNMovies, movieIds);
                    subscriptionMovie.updateTableUsing(connection);
                    System.out.println("subscription_movie generated");
                }

                if (parameters.isGenAll() || parameters.isGenSubscriptionsToSeriesOnly()) {
                    final var seriesSeasonIds = StorageDAO.instance().ids(connection, "series_season");
                    final var subscriptionSeriesSeason = new SubscriptionSeriesSeasonTableGenerator(subscriptionIdsMSeasons, seriesSeasonIds);
                    subscriptionSeriesSeason.updateTableUsing(connection);
                    System.out.println("subscription_series_season generated");
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 5: Launcher.java

```java
package com.lamtev.movie_service.datagen.cli_args;

import com.google.gson.*;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.io.FileReader;
import java.lang.reflect.Type;
import java.util.Arrays;

public class ArgumentsParser {

    @NotNull
    private final String[] args;
    @NotNull
    private final Gson gson;

    public ArgumentsParser(final @NotNull String[] args) {
        this.args = args;
        this.gson = new GsonBuilder()
                .serializeNulls()
                .registerTypeAdapter(EndpointInfo.class, new Deserializer<EndpointInfo>())
                .registerTypeAdapter(Parameters.class, new Deserializer<Parameters>())
                .create();
    }

    @Nullable
    public EndpointInfo endpoint() {
        try (final var fileReader = new FileReader(args[0])) {
            return gson.fromJson(fileReader, EndpointInfo.class);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    @Nullable
    public Parameters parameters() {
        try (final var fileReader = new FileReader(args[1])) {
            final var params = gson.fromJson(fileReader, Parameters.class);
            if (args.length == 3) {
                switch (args[2]) {
                    case "onlyMovies":
                        params.setGenMoviesOnly(true);
                        params.setGenAll(false);
                        break;
                    case "onlySeries":
                        params.setGenSeriesOnly(true);
                        params.setGenAll(false);
                        break;
                    case "onlyUsers":
                        params.setGenUsersOnly(true);
                        params.setGenAll(false);
                        break;
                    case "onlyMovieSubscriptions":
                        params.setGenSubscriptionsToMoviesOnly(true);
                        params.setGenAll(false);
                        break;
                    case "onlySeriesSubscriptions":
                        params.setGenSubscriptionsToSeriesOnly(true);
                        params.setGenAll(false);
                        break;
                    default:
                        params.setGenAll(true);
                        break;
                }
            } else {
                params.setGenAll(true);
            }
            return params;
        } catch (Exception e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

    class Deserializer<T> implements JsonDeserializer<T> {

        public T deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext
    context) throws JsonParseException {
            final T obj = new Gson().fromJson(json, typeOfT);

            final var badField = Arrays.stream(obj.getClass().getDeclaredFields())
                    .filter(field -> {
                        try {
                            field.setAccessible(true);
                            return field.get(obj) == null;
                        } catch (IllegalAccessError | IllegalAccessException ignored) {
                            return false;
                        }
                    })
                    .findFirst();

            if (badField.isPresent()) {
                throw new JsonParseException("Missing field: " + badField.get().getName());
            }

            return obj;
        }
    }

}
```

Листинг 6: ArgumentsParser.java

```java
package com.lamtev.movie_service.datagen.cli_args;

import org.jetbrains.annotations.NotNull;

public class EndpointInfo {

    @NotNull
    private final String url;
    @NotNull
    private final String user;
    @NotNull
    private final String password;

    public EndpointInfo(@NotNull final String url,
                        @NotNull final String user,
                        @NotNull final String password) {
        this.url = url;
        this.user = user;
        this.password = password;
    }

    @NotNull
    public String url() {
        return url;
    }

    @NotNull
    public String user() {
        return user;
    }

    @NotNull
    public String password() {
        return password;
    }

}
```

Листинг 7: EndpointInfo.java

```java
package com.lamtev.movie_service.datagen.cli_args;

import org.jetbrains.annotations.NotNull;

public final class Parameters {

    private final int usersCount;
    private final int femalePercentage;
    private final int moviesCount;
    /**
     * {{1000, 3, 15}, ...} - 1000 series, each consists of 3 seasons with 15 episodes
     */
    @NotNull
    private final int[][] seriesCountSeasonsEpisodes;
    private final int percentageOfUsersWhoBoughtMovies;
    private final int minMoviesPerUser;
    private final int maxMoviesPerUser;
    private final int percentageOfUsersWhoBoughtSeries;
    private final int minSeriesPerUser;
    private final int maxSeriesPerUser;
    private final int minSubscriptionsPerUser;
    private final int maxSubscriptionsPerUser;
    /**
     * {{duration in days, price in USD, number of movies, number of series seasons}, ... }
     */
    @NotNull
    private final int[][] durationPriceNMoviesMSeasons;
    private final int moviesSubscriptionsPercentage;
    private final int yearsSinceFirstSubscription;

    private boolean genMoviesOnly = false;
    private boolean genSeriesOnly = false;
    private boolean genUsersOnly = false;
    private boolean genSubscriptionsToMoviesOnly = false;
    private boolean genSubscriptionsToSeriesOnly = false;
    private boolean genAll = true;

    public Parameters(int usersCount,
                      int femalePercentage,
                      int moviesCount,
                      final @NotNull int[][] seriesCountSeasonsEpisodes,
                      int percentageOfUsersWhoBoughtMovies,
                      int minMoviesPerUser,
                      int maxMoviesPerUser,
                      int percentageOfUsersWhoBoughtSeries,
                      int minSeriesPerUser,
                      int maxSeriesPerUser,
                      int yearsSinceFirstSubscription,
                      int minSubscriptionsPerUser,
                      int maxSubscriptionsPerUser,
                      @NotNull int[][] durationPriceNMoviesMSeasons,
                      int moviesSubscriptionsPercentage) {
        this.usersCount = usersCount;
        this.femalePercentage = femalePercentage;
        this.moviesCount = moviesCount;
        this.seriesCountSeasonsEpisodes = seriesCountSeasonsEpisodes;
        this.percentageOfUsersWhoBoughtMovies = percentageOfUsersWhoBoughtMovies;
        this.minMoviesPerUser = minMoviesPerUser;
        this.maxMoviesPerUser = maxMoviesPerUser;
        this.percentageOfUsersWhoBoughtSeries = percentageOfUsersWhoBoughtSeries;
        this.minSeriesPerUser = minSeriesPerUser;
        this.maxSeriesPerUser = maxSeriesPerUser;
        this.yearsSinceFirstSubscription = yearsSinceFirstSubscription;
        this.minSubscriptionsPerUser = minSubscriptionsPerUser;
        this.maxSubscriptionsPerUser = maxSubscriptionsPerUser;
        this.durationPriceNMoviesMSeasons = durationPriceNMoviesMSeasons;
        this.moviesSubscriptionsPercentage = moviesSubscriptionsPercentage;
    }

    public int femalePercentage() {
        return femalePercentage;
    }

    public int percentageOfUsersWhoBoughtMovies() {
        return percentageOfUsersWhoBoughtMovies;
    }

    public int minMoviesPerUser() {
        return minMoviesPerUser;
    }

    public int maxMoviesPerUser() {
        return maxMoviesPerUser;
    }

    public int percentageOfUsersWhoBoughtSeries() {
        return percentageOfUsersWhoBoughtSeries;
    }

    public int minSeriesPerUser() {
        return minSeriesPerUser;
    }

    public int maxSeriesPerUser() {
        return maxSeriesPerUser;
    }

    public int minSubscriptionsPerUser() {
        return minSubscriptionsPerUser;
    }

    public int maxSubscriptionsPerUser() {
        return maxSubscriptionsPerUser;
    }

    @NotNull
    public int[][] durationPriceNMoviesMSeasons() {
        return durationPriceNMoviesMSeasons;
    }

    public int moviesSubscriptionsPercentage() {
        return moviesSubscriptionsPercentage;
    }

    public int usersCount() {
        return usersCount;
    }

    public int moviesCount() {
        return moviesCount;
    }

    @NotNull
    public int[][] seriesCountSeasonsEpisodes() {
        return seriesCountSeasonsEpisodes;
    }

    public int yearsSinceFirstSubscription() {
        return yearsSinceFirstSubscription;
    }

    public boolean isGenMoviesOnly() {
        return genMoviesOnly;
    }

    public void setGenMoviesOnly(boolean genMoviesOnly) {
        this.genMoviesOnly = genMoviesOnly;
    }

    public boolean isGenSeriesOnly() {
        return genSeriesOnly;
    }

    public void setGenSeriesOnly(boolean genSeriesOnly) {
        this.genSeriesOnly = genSeriesOnly;
    }

    public boolean isGenUsersOnly() {
        return genUsersOnly;
    }

    public void setGenUsersOnly(boolean genUsersOnly) {
        this.genUsersOnly = genUsersOnly;
    }

    public boolean isGenSubscriptionsToMoviesOnly() {
        return genSubscriptionsToMoviesOnly;
    }

```

```java
    public void setGenSubscriptionsToMoviesOnly(boolean genSubscriptionsToMoviesOnly) {
        this.genSubscriptionsToMoviesOnly = genSubscriptionsToMoviesOnly;
    }

    public boolean isGenSubscriptionsToSeriesOnly() {
        return genSubscriptionsToSeriesOnly;
    }

    public void setGenSubscriptionsToSeriesOnly(boolean genSubscriptionsToSeriesOnly) {
        this.genSubscriptionsToSeriesOnly = genSubscriptionsToSeriesOnly;
    }

    public boolean isGenAll() {
        return genAll;
    }

    public void setGenAll(boolean genAll) {
        this.genAll = genAll;
    }

}
```

Листинг 8: Parameters.java

```java
package com.lamtev.movie_service.datagen.generator;

import com.github.javafaker.Faker;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.util.Locale;
import java.util.Random;

public interface TableGenerator {
    @NotNull
    Random RANDOM = new Random(System.currentTimeMillis());
    @NotNull
    Faker FAKER = new Faker(Locale.US, RANDOM);
    @NotNull
    Utils UTILS = new Utils(RANDOM, FAKER);

    /**
     * Updates corresponding table via {@code connection} with newly generated data.
     *
     * @param connection {@link Connection} (session) with data base.
     */
    void updateTableUsing(final @NotNull Connection connection);
}
```

Листинг 9: TableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator;

import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class LanguageTableGenerator implements TableGenerator {

    @NotNull
    private final String[] languages;

    public LanguageTableGenerator(@NotNull String[] languages) {
        this.languages = languages;
    }

    public LanguageTableGenerator() {
        this(new String[]{"en-US", "ru-RU"});
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                    "INSERT INTO language (name) VALUES (?)"
        )) {
            for (final var language : languages) {
                statement.setString(1, language);
                statement.addBatch();
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 10: LanguageTableGenerator.java

```
package com.lamtev.movie_service.datagen.generator.category;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.LinkedHashMap;
import java.util.Map;

import static java.sql.Statement.RETURN_GENERATED_KEYS;

public final class CategoryTableGenerator implements TableGenerator {

    private static final Map<String, String> CATEGORY_TO_SUPERCATEGORY = new LinkedHashMap<>()
     {{
        put("genre", "");
        put("comedy", "genre");
        put("drama", "genre");
        put("thriller", "genre");
        put("new", "");
        put("horror", "genre");
        put("action", "genre");
        put("crime", "genre");
        put("western", "genre");
        put("popular", "");
        put("mystery", "genre");
        put("adventure", "genre");
        put("classic", "");
        put("romance", "genre");
        put("science-fiction", "genre");
        put("soviet", "");
        put("hollywood", "");
    }};

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.createStatement()) {
            final var categoryIds = new int[CATEGORY_TO_SUPERCATEGORY.size()];
            int i = 0;
            for (final var entry : CATEGORY_TO_SUPERCATEGORY.entrySet()) {
                final var category = entry.getKey();
                final var supercategory = entry.getValue();

                final var query = String.format(
                        "INSERT INTO category (name, supercategory_id) " +
                                "SELECT '%s', (SELECT id FROM category WHERE name = '%s' LIMIT
     1)", category, supercategory
                );
                try {
                    statement.executeUpdate(query, RETURN_GENERATED_KEYS);
                    final var generatedKeys = statement.getGeneratedKeys();
                    if (generatedKeys.next()) {
                        categoryIds[i] = generatedKeys.getInt(1);
                    }
                    i++;
                } catch (SQLException e) {
                    e.printStackTrace();
                }
```

```java
            }
            final var categoryTranslation = new CategoryTranslationTableGenerator(categoryIds)
    ;
            categoryTranslation.updateTableUsing(connection);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 11: CategoryTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.category;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class CategoryTranslationTableGenerator implements TableGenerator {

    @NotNull
    private final int[] categoryIds;

    public CategoryTranslationTableGenerator(final @NotNull int[] categoryIds) {
        this.categoryIds = categoryIds;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO category_translation (category_id, language_id, translation)
    VALUES (?, ?, ?)"
        )) {
            for (final var categoryId : categoryIds) {
                for (int languageId = 1; languageId <= 2; ++languageId) {
                    int i = 0;
                    statement.setInt(++i, categoryId);
                    statement.setInt(++i, languageId);
                    statement.setString(++i, FAKER.lorem().word());
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 12: CategoryTranslationTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.movie;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;
import org.postgresql.util.PGmoney;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;

import static java.sql.Statement.RETURN_GENERATED_KEYS;

public final class MovieTableGenerator implements TableGenerator {

    private static final short[] MOVIE_PRICES_IN_USD = new short[]{5, 5, 5, 5, 5, 7, 7, 7, 7,
    10, 10, 10, 15, 15, 20, 25, 35};
    private final int movieCount;
    private final int seriesSeasonId;
    private final int seriesPrice;

    public MovieTableGenerator(int count) {
        this(count, 0, 0);
    }

    public MovieTableGenerator(int movieCount, int seriesSeasonId, int seriesPrice) {
        this.movieCount = movieCount;
        this.seriesSeasonId = seriesSeasonId;
        this.seriesPrice = seriesPrice;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO movie (price, release_date, imdb_rating, series_season_id) VALUES (?, ?, ?, ?)",
                RETURN_GENERATED_KEYS
        )) {
            final var moviesAreSeriesSeasonEpisodes = seriesSeasonId != 0;
            final var date = UTILS.randomDate(50);
            final var rating = UTILS.randomRating();
            for (int i = 0; i < movieCount; ++i) {
                int j = 0;
                statement.setObject(++j, new PGmoney("$" + (
                        seriesPrice == 0 ?
                                MOVIE_PRICES_IN_USD[RANDOM.nextInt(MOVIE_PRICES_IN_USD.length)]
                                : seriesPrice
                )));
                if (moviesAreSeriesSeasonEpisodes) {
                    statement.setDate(++j, date);
                    statement.setFloat(++j, rating);
                    statement.setInt(++j, seriesSeasonId);
                } else {
                    statement.setDate(++j, UTILS.randomDate(50));
                    statement.setFloat(++j, UTILS.randomRating());
                    statement.setNull(++j, Types.INTEGER);
                }
                statement.addBatch();
            }
            statement.executeBatch();

            final var movieIds = UTILS.getIdsOfRowsInsertedWith(statement, movieCount);

            final var movieTranslation = new MovieTranslationTableGenerator(movieIds, moviesAreSeriesSeasonEpisodes);
            movieTranslation.updateTableUsing(connection);

            updateMovieCategoryTableUsing(connection, moviesAreSeriesSeasonEpisodes, movieIds);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private void updateMovieCategoryTableUsing(@NotNull Connection connection, boolean moviesAreSeriesSeasonEpisodes, int[] movieIds) {
        try (final var categoriesStatement = connection.createStatement()) {
            categoriesStatement.executeQuery("SELECT COUNT(*) FROM category");
            var result = categoriesStatement.getResultSet();
            if (result != null && result.next()) {
                int categoriesCount = result.getInt(1);
                final var categoryIds = new int[categoriesCount - 1];
                int i = 0;
                categoriesStatement.executeQuery("SELECT id FROM category WHERE name != 'genre'");
                result = categoriesStatement.getResultSet();
                if (result != null) {
                    while (result.next()) {
                        categoryIds[i++] = result.getShort(1);
                    }
                }

                final var movieCategory = new MovieCategoryTableGenerator(movieIds, categoryIds, moviesAreSeriesSeasonEpisodes);
                movieCategory.updateTableUsing(connection);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 13: MovieTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.movie;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class MovieTranslationTableGenerator implements TableGenerator {

    private static final String VIDEO_URL_TEMPLATE = "https://blob.movie-service.lamtev.com/?
    vid=";

    @NotNull
    private final int[] movieIds;
    private final boolean moviesAreSeriesEpisodes;

    public MovieTranslationTableGenerator(final @NotNull int[] movieIds, boolean
    moviesAreSeriesEpisodes) {
        this.movieIds = movieIds;
        this.moviesAreSeriesEpisodes = moviesAreSeriesEpisodes;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO movie_translation (movie_id, language_id, name, director,
    description, file_url) " +
                        "VALUES (?, ?, ?, ?, ?, ?)"
        )) {
            final var director = moviesAreSeriesEpisodes ? FAKER.artist().name() : null;
            for (int movieIdIdx = 0; movieIdIdx < movieIds.length; ++movieIdIdx) {
                for (int languageId = 1; languageId <= 2; ++languageId) {
                    int i = 0;
                    statement.setInt(++i, movieIds[movieIdIdx]);
                    statement.setInt(++i, languageId);
                    if (moviesAreSeriesEpisodes) {
                        statement.setString(++i, "Episode " + movieIdIdx);
                        statement.setString(++i, director);
                    } else {
                        final var movie = FAKER.book();
                        statement.setString(++i, movie.title());
                        statement.setString(++i, movie.author());
                    }
                    statement.setString(++i, FAKER.lorem().paragraph(10));
                    statement.setString(++i, randomUrl());
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private String randomUrl() {
        return VIDEO_URL_TEMPLATE + RANDOM.nextInt(Integer.MAX_VALUE);
    }

}
```

Листинг 14: MovieTranslationTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.movie;
```

```java

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import gnu.trove.list.TIntList;
import gnu.trove.list.array.TIntArrayList;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.Arrays;

public final class MovieCategoryTableGenerator implements TableGenerator {

    @NotNull
    private final int[] movieIds;
    @NotNull
    private final TIntList categoryIds;
    private final boolean sameCategoriesForAllMovies;

    public MovieCategoryTableGenerator(final @NotNull int[] movieIds, final @NotNull int[]
    categoryIds, boolean sameCategoriesForAllMovies) {
        this.movieIds = movieIds;
        this.categoryIds = new TIntArrayList(categoryIds.length);
        Arrays.stream(categoryIds).forEach(this.categoryIds::add);
        this.sameCategoriesForAllMovies = sameCategoriesForAllMovies;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO movie_category (movie_id, category_id) VALUES (?, ?)"
        )) {
            final var categories = nRandomCategories(3);
            for (int movieId : movieIds) {
                final var differentCategories = nRandomCategories(3);
                for (int j = 0; j < categories.length; ++j) {
                    int i = 0;
                    statement.setInt(++i, movieId);
                    if (sameCategoriesForAllMovies) {
                        statement.setInt(++i, categories[j]);
                    } else {
                        statement.setInt(++i, differentCategories[j]);
                    }
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @NotNull
    private int[] nRandomCategories(int n) {
        categoryIds.shuffle(RANDOM);
        final var res = new int[n];
        for (int i = 0; i < n; ++i) {
            res[i] = categoryIds.get(i);
        }

        return res;
    }

}
```

Листинг 15: MovieCategoryTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.series;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import com.lamtev.movie_service.datagen.generator.series.season.SeriesSeasonTableGenerator;
import org.jetbrains.annotations.NotNull;
import org.postgresql.util.PGmoney;

import java.sql.Connection;
import java.sql.SQLException;

import static java.sql.Statement.RETURN_GENERATED_KEYS;

public final class SeriesTableGenerator implements TableGenerator {

    private static final short[] SERIES_PRICES_IN_USD = new short[]{10, 10, 10, 10, 17, 17,
    17, 25, 25, 35};

    /**
     * [[1000, 3, 15], ...] - 1000 series, each consists of 3 seasons with 15 episodes
     */
    @NotNull
    private final int[][] countSeasonsEpisodesArray;

    public SeriesTableGenerator(final @NotNull int[][] countSeasonsEpisodes) {
        this.countSeasonsEpisodesArray = countSeasonsEpisodes;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO series (seasons, price) VALUES (?, ?)",
                RETURN_GENERATED_KEYS
        )) {
            for (final var countSeasonsEpisodes : countSeasonsEpisodesArray) {
                final int seriesCount = countSeasonsEpisodes[0];
                final short seasons = (short) countSeasonsEpisodes[1];
                final var seriesPrices = new int[seriesCount];
                for (int seriesIdx = 0; seriesIdx < seriesCount; ++seriesIdx) {
                    final int seriesPrice = SERIES_PRICES_IN_USD[RANDOM.nextInt(
    SERIES_PRICES_IN_USD.length)];
                    seriesPrices[seriesIdx] = seriesPrice;
                    int i = 0;
                    statement.setShort(++i, seasons);
                    statement.setObject(++i, new PGmoney("$" + seriesPrice));
                    statement.addBatch();
                }
                statement.executeBatch();

                final var seriesIds = UTILS.getIdsOfRowsInsertedWith(statement, seriesCount);

                final var seriesTranslation = new SeriesTranslationTableGenerator(seriesIds);
                seriesTranslation.updateTableUsing(connection);

                final int episodes = countSeasonsEpisodes[2];
                final var seriesSeason = new SeriesSeasonTableGenerator(seriesIds,
    seriesPrices, seasons, episodes);
                seriesSeason.updateTableUsing(connection);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 16: SeriesTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.series;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class SeriesTranslationTableGenerator implements TableGenerator {

    @NotNull
    private final int[] seriesIds;

    public SeriesTranslationTableGenerator(final @NotNull int[] seriesIds) {
        this.seriesIds = seriesIds;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO series_translation (series_id, language_id, name, director,
    description) VALUES (?, ?, ?, ? , ?)"
        )) {
            for (final var seriesId : seriesIds) {
                for (int languageId = 1; languageId <= 2; ++languageId) {
                    int i = 0;
                    statement.setInt(++i, seriesId);
                    statement.setInt(++i, languageId);
                    final var series = FAKER.book();
                    statement.setString(++i, series.title());
                    statement.setString(++i, series.author());
                    statement.setString(++i, FAKER.lorem().paragraph(10));
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 17: SeriesTranslationTableGenerator.java

```
package com.lamtev.movie_service.datagen.generator.series.season;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import com.lamtev.movie_service.datagen.generator.movie.MovieTableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

import static java.sql.Statement.RETURN_GENERATED_KEYS;

public final class SeriesSeasonTableGenerator implements TableGenerator {

    @NotNull
    private final int[] seriesIds;
    @NotNull
    private final int[] seriesPrices;
    private final short seasonsCount;
    private final int episodesCount;

    public SeriesSeasonTableGenerator(final @NotNull int[] seriesIds,
                                      final @NotNull int[] seriesPrices, short seasonsCount,
    int episodesCount) {
        this.seriesIds = seriesIds;
        this.seriesPrices = seriesPrices;
        this.seasonsCount = seasonsCount;
        this.episodesCount = episodesCount;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO series_season (series_id, number) VALUES (?, ?)",
                RETURN_GENERATED_KEYS
        )) {
            for (final int id : seriesIds) {
                for (short season = 0; season < seasonsCount; ++season) {
                    int i = 0;
                    statement.setInt(++i, id);
                    statement.setShort(++i, season);
                    statement.addBatch();
                }
            }
            statement.executeBatch();

            final var seasonIds = UTILS.getIdsOfRowsInsertedWith(statement, seriesIds.length *
     seasonsCount);
```

```java

            final var seasonTranslation = new SeriesSeasonTranslationTableGenerator(seasonIds)
    ;
            seasonTranslation.updateTableUsing(connection);

            for (int seasonIdx = 0; seasonIdx < seasonIds.length; ++seasonIdx) {
                final var episode = new MovieTableGenerator(episodesCount, seasonIds[seasonIdx
    ], seriesPrices[seasonIdx / seasonsCount]);
                episode.updateTableUsing(connection);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 18: SeriesSeasonTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.series.season;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class SeriesSeasonTranslationTableGenerator implements TableGenerator {

    @NotNull
    private final int[] seasonIds;

    public SeriesSeasonTranslationTableGenerator(final @NotNull int[] seasonIds) {
        this.seasonIds = seasonIds;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO series_season_translation (series_season_id, language_id, name,
    description) " +
                        " VALUES (?, ?, ?, ?)"
        )) {
            for (int languageId = 1; languageId <= 2; ++languageId) {
                for (final var seasonId : seasonIds) {
                    int i = 0;
                    statement.setInt(++i, seasonId);
                    statement.setInt(++i, languageId);
                    statement.setString(++i, FAKER.book().title());
                    statement.setString(++i, FAKER.lorem().paragraph(10));
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 19: SeriesSeasonTranslationTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.user;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

import static java.sql.Statement.RETURN_GENERATED_KEYS;

public final class UserTableGenerator implements TableGenerator {

    private static byte[] buf = null;
    private final long count;
    private final int femalePercent;

    public UserTableGenerator(long count, int femalePercentage) {
        this.count = count;
        this.femalePercent = femalePercentage;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO \"user\" (login, password_hash, email, birthday, sex, first_name,
 last_name) " +
                        "VALUES (?, ?, ?, ?, ?, ?, ?)",
                RETURN_GENERATED_KEYS
        )) {
            for (int i = 0; i < count; ++i) {
                int j = 0;
                final var firstName = FAKER.name().firstName();
                final var lastName = FAKER.name().lastName();
                final var username = firstName + "." + lastName + RANDOM.nextInt((int) count);
                statement.setString(++j, username);
                statement.setString(++j, Long.toHexString(FAKER.number().randomNumber()));
                statement.setString(++j, username + "@email.com");
                statement.setDate(++j, UTILS.randomDate(100));
                statement.setString(++j, randomSex());
                statement.setString(++j, firstName);
                statement.setString(++j, lastName);
                statement.addBatch();
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private String randomSex() {
        if (buf == null) {
            buf = new byte[100];
            for (int i = 0; i < buf.length; ++i) {
                if (i < femalePercent) {
                    buf[i] = 0;
                } else {
                    buf[i] = 1;
                }
            }
        }

        return Byte.toString(buf[RANDOM.nextInt(100)]);
    }

}
```

Листинг 20: UserTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.user;

import com.lamtev.movie_service.datagen.generator.StorageDAO;
import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.postgresql.util.PGmoney;

import java.sql.Connection;
import java.sql.SQLException;

public final class UserMovieTableGenerator implements TableGenerator {

    private final int percentageOfUsersWhoBoughtMovies;
    private final int minMovies;
    private final int maxMovies;

    public UserMovieTableGenerator(int percentageOfUsersWhoBoughtMovies, int minMovies, int
        maxMovies) {
        this.percentageOfUsersWhoBoughtMovies = percentageOfUsersWhoBoughtMovies;
        this.minMovies = minMovies;
        this.maxMovies = maxMovies;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
        final var movieIdsPrices = movieIdsPrices(connection);
        if (userIds.length == 0 || movieIdsPrices == null) {
            return;
        }

        try (final var statement = connection.prepareStatement(
                "INSERT INTO user_movie (user_id, movie_id, payment) VALUES (?, ?, ?)"
        )) {
            for (final var userId : userIds) {
                if (userId % 100 < percentageOfUsersWhoBoughtMovies) {
                    final var nMovies = RANDOM.nextInt(maxMovies - minMovies + 1) + minMovies;
                    final var movieIdx = RANDOM.nextInt(movieIdsPrices[0].length - nMovies);
                    for (int i = 0; i < nMovies; ++i) {
                        int j = 0;
                        statement.setLong(++j, userId);
                        statement.setInt(++j, movieIdsPrices[0][movieIdx + i]);
                        statement.setObject(++j, new PGmoney("$" + movieIdsPrices[1][movieIdx
        + i]));
                        statement.addBatch();
                    }
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Nullable
    private int[][] movieIdsPrices(final @NotNull Connection connection) {
        try (final var statement = connection.createStatement()) {
            int count = StorageDAO.instance().count(connection, "movie");
            int[][] movieIdsPrices = new int[2][count];
            statement.executeQuery("SELECT id, price FROM movie");
            final var result = statement.getResultSet();
            int i = 0;
            if (result != null) {
                while (result.next()) {
                    movieIdsPrices[0][i] = result.getInt(1);
                    movieIdsPrices[1][i] = result.getInt(2);
                    i++;
                }
            }

            return movieIdsPrices;
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }

}
```

Листинг 21: UserMovieTableGenerator.java

```
package com.lamtev.movie_service.datagen.generator.user;

import com.lamtev.movie_service.datagen.generator.StorageDAO;
import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.postgresql.util.PGmoney;

import java.sql.Connection;
import java.sql.SQLException;
```

```java

public final class UserSeriesTableGenerator implements TableGenerator {

    private final int percentageOfUsersWhoBoughtSeries;
    private final int minSeries;
    private final int maxSeries;

    public UserSeriesTableGenerator(int percentageOfUsersWhoBoughtSeries, int minSeries, int
    maxSeries) {
        this.percentageOfUsersWhoBoughtSeries = percentageOfUsersWhoBoughtSeries;
        this.minSeries = minSeries;
        this.maxSeries = maxSeries;
    }

    //TODO: get rid of duplicates
    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
        final var seriesIdsPrices = seriesIdsPrices(connection);
        if (userIds.length == 0 || seriesIdsPrices == null) {
            return;
        }
        try (final var statement = connection.prepareStatement(
                "INSERT INTO user_series (user_id, series_id, payment) VALUES (?, ?, ?)"
        )) {
            for (final var userId : userIds) {
                if (userId % 100 < percentageOfUsersWhoBoughtSeries) {
                    final var nSeries = RANDOM.nextInt(maxSeries - minSeries + 1) + minSeries;
                    final var seriesIdx = RANDOM.nextInt(seriesIdsPrices[0].length - nSeries);
                    for (int i = 0; i < nSeries; ++i) {
                        int j = 0;
                        statement.setLong(++j, userId);
                        statement.setInt(++j, seriesIdsPrices[0][seriesIdx + i]);
                        statement.setObject(++j, new PGmoney("$" + seriesIdsPrices[1][
    seriesIdx + i]));
                        statement.addBatch();
                    }
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Nullable
    private int[][] seriesIdsPrices(final @NotNull Connection connection) {
        try (final var statement = connection.createStatement()) {
            int count = StorageDAO.instance().count(connection, "series");
            int[][] seriesIdsPrices = new int[2][count];
            statement.executeQuery("SELECT id, price FROM series");
            final var result = statement.getResultSet();
            int i = 0;
            if (result != null) {
                while (result.next()) {
                    seriesIdsPrices[0][i] = result.getInt(1);
                    seriesIdsPrices[1][i] = result.getInt(2);
                    i++;
                }
            }

            return seriesIdsPrices;
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }

}
```

Листинг 22: UserSeriesTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.subscription;

```

```java
import com.lamtev.movie_service.datagen.generator.StorageDAO;
import com.lamtev.movie_service.datagen.generator.TableGenerator;
import gnu.trove.set.TIntSet;
import org.jetbrains.annotations.NotNull;
import org.postgresql.util.PGmoney;

import java.sql.Connection;
import java.sql.Date;
import java.sql.SQLException;
import java.util.Calendar;

public final class SubscriptionTableGenerator implements TableGenerator {

    private final long usersCount;
    private final int minSubscriptionsPerUser;
    private final int maxSubscriptionsPerUser;
    @NotNull
    private final int[][] durationPriceNMoviesMSeasons;
    private final int yearsSinceFirstSubscription;
    private TIntSet generatedIds;

    public SubscriptionTableGenerator(long usersCount, int minSubscriptionsPerUser,
                                      int maxSubscriptionsPerUser, final @NotNull int[][] durationPriceNMoviesMSeasons,
                                      int yearsSinceFirstSubscription) {
        this.usersCount = usersCount;
        this.minSubscriptionsPerUser = minSubscriptionsPerUser;
        this.maxSubscriptionsPerUser = maxSubscriptionsPerUser;
        this.durationPriceNMoviesMSeasons = durationPriceNMoviesMSeasons;
        this.yearsSinceFirstSubscription = yearsSinceFirstSubscription;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
        try (final var statement = connection.prepareStatement(
                "INSERT INTO subscription (user_id, started, expires, autorenewable, payment) VALUES (?, ?, ?, ?, ?)"
        )) {
            RANDOM.ints(usersCount, 0, userIds.length).forEach(idx -> {
                final var nSubscriptions = RANDOM.nextInt(maxSubscriptionsPerUser - minSubscriptionsPerUser + 1) + minSubscriptionsPerUser;
                for (int j = 0; j < nSubscriptions; ++j) {
                    int i = 0;
                    try {
                        statement.setLong(++i, userIds[idx]);
                        final var started = UTILS.randomDate(yearsSinceFirstSubscription);
                        statement.setObject(++i, started);
                        final var calendar = Calendar.getInstance();
                        calendar.setTimeInMillis(started.getTime());
                        final var durationPrice = durationPriceNMoviesMSeasons[RANDOM.nextInt(durationPriceNMoviesMSeasons.length)];
                        calendar.add(Calendar.DATE, durationPrice[0]);
                        statement.setObject(++i, new Date(calendar.getTimeInMillis()));
                        statement.setBoolean(++i, RANDOM.nextBoolean());
                        statement.setObject(++i, new PGmoney("$" + durationPrice[1]));
                        statement.addBatch();
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
            });

            statement.executeBatch();
            generatedIds = UTILS.getIdsOfRowsInsertedWith(statement);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public TIntSet getGeneratedIds() {
        return generatedIds;
    }

}
```

Листинг 23: SubscriptionTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.subscription;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class SubscriptionMovieTableGenerator implements TableGenerator {

    @NotNull
    private final int[][] subscriptionIdsNMovies;
    @NotNull
    private final int[] movieIds;

    public SubscriptionMovieTableGenerator(final @NotNull int[][] subscriptionIdsNMovies, final @NotNull int[] movieIds) {
        this.subscriptionIdsNMovies = subscriptionIdsNMovies;
        this.movieIds = movieIds;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO subscription_movie (subscription_id, movie_id) VALUES (?, ?)"
        )) {
            for (int j = 0; j < subscriptionIdsNMovies[0].length; ++j) {
                final var nMovies = subscriptionIdsNMovies[1][j];
                final var movieIdsIdxs = UTILS.nUniqueRandomInts(nMovies, movieIds.length);
                for (final var movieIdsIdx : movieIdsIdxs) {
                    int i = 0;
                    statement.setLong(++i, subscriptionIdsNMovies[0][j]);
                    statement.setInt(++i, movieIds[movieIdsIdx]);
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 24: SubscriptionMovieTableGenerator.java

```java
package com.lamtev.movie_service.datagen.generator.subscription;

import com.lamtev.movie_service.datagen.generator.TableGenerator;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class SubscriptionSeriesSeasonTableGenerator implements TableGenerator {

    @NotNull
    private final int[][] subscriptionIdsMSeasons;
    @NotNull
    private final int[] seriesSeasonIds;

    public SubscriptionSeriesSeasonTableGenerator(final @NotNull int[][] subscriptionIdsMSeasons, final @NotNull int[] seriesSeasonIds) {
        this.subscriptionIdsMSeasons = subscriptionIdsMSeasons;
        this.seriesSeasonIds = seriesSeasonIds;
    }

    @Override
    public void updateTableUsing(final @NotNull Connection connection) {
        try (final var statement = connection.prepareStatement(
                "INSERT INTO subscription_series_season (subscription_id, series_season_id)
    VALUES (?, ?)"
        )) {
            for (int j = 0; j < subscriptionIdsMSeasons[0].length; ++j) {
                final var nSeriesSeasons = subscriptionIdsMSeasons[1][j];
                final var seriesIdsIdxs = UTILS.nUniqueRandomInts(nSeriesSeasons,
    seriesSeasonIds.length);
                for (final var seriesIdsIdx : seriesIdsIdxs) {
                    int i = 0;
                    statement.setLong(++i, subscriptionIdsMSeasons[0][j]);
                    statement.setInt(++i, seriesSeasonIds[seriesIdsIdx]);
                    statement.addBatch();
                }
            }
            statement.executeBatch();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

Листинг 25: SubscriptionSeriesSeasonTableGenerator.java

```
package com.lamtev.movie_service.datagen.generator;

import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class StorageDAO {

    private StorageDAO() {
    }

    public static StorageDAO instance() {
        return Holder.INSTANCE;
    }

    public final int count(final @NotNull Connection connection, final @NotNull String
    tableName) {
        int count = 0;
        try (final var statement = connection.createStatement()) {
            statement.executeQuery("SELECT COUNT(*) FROM " + tableName);
            var result = statement.getResultSet();
            if (result != null && result.next()) {
                count = result.getInt(1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return count;
    }

    @NotNull
    public final int[] ids(final @NotNull Connection connection, final @NotNull String
    tableName) {
        try (final var statement = connection.createStatement()) {
            final int count = count(connection, tableName);

            final var ids = new int[count];
            statement.executeQuery("SELECT id FROM " + tableName);

            final var result = statement.getResultSet();
            if (result != null) {
                int i = 0;
                while (result.next()) {
                    ids[i++] = result.getInt(1);
                }
            }
            return ids;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return new int[0];
    }

    private static final class Holder {
        private static final StorageDAO INSTANCE = new StorageDAO();
    }

}
```

Листинг 26: StorageDAO.java

```
package com.lamtev.movie_service.datagen.generator.subscription;


import com.lamtev.movie_service.datagen.generator.StorageDAO;
import gnu.trove.set.TIntSet;
import org.jetbrains.annotations.NotNull;

import java.sql.Connection;
import java.sql.SQLException;

public final class SubscriptionTableDAO {

    private SubscriptionTableDAO() {
    }

    public static SubscriptionTableDAO instance() {
        return Holder.INSTANCE;
    }

    @NotNull
    public int[][] idsNMoviesOrMSeasonsContainingInIds(final @NotNull Connection connection,
    final @NotNull int[][] durationPriceNMoviesMSeasons, final @NotNull TIntSet idsSet) {
        try (final var statement = connection.createStatement()) {
            int count = StorageDAO.instance().count(connection, "subscription");
            final var idsNMoviesOrMSeasons = new int[3][count];
            statement.executeQuery("SELECT id, (expires - started), payment FROM subscription
    GROUP BY id");
            final var result = statement.getResultSet();
            int i = 0;
            if (result != null) {
                while (result.next()) {
                    int id = result.getInt(1);
                    if (idsSet.contains(id)) {
                        idsNMoviesOrMSeasons[0][i] = id;
                        final var nm = nMoviesMSeasons(result.getInt(2), result.getInt(3),
    durationPriceNMoviesMSeasons);
                        idsNMoviesOrMSeasons[1][i] = nm[0];
                        idsNMoviesOrMSeasons[2][i] = nm[1];
                        i++;
                    }
                }
            }
            return idsNMoviesOrMSeasons;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return new int[0][0];
    }

    @NotNull
    private int[] nMoviesMSeasons(int duration, int payment, final @NotNull int[][]
    durationPriceNMoviesMSeasons) {
        int n = 0;
        int m = 0;
        for (int[] durationPriceNMoviesMSeason : durationPriceNMoviesMSeasons) {
            if (durationPriceNMoviesMSeason[0] == duration && durationPriceNMoviesMSeason[1]
    == payment) {
                n = durationPriceNMoviesMSeason[2];
                m = durationPriceNMoviesMSeason[3];
                break;
            }
        }

        return new int[]{n, m};
```

```java
    }

    private static final class Holder {
        private static final SubscriptionTableDAO INSTANCE = new SubscriptionTableDAO();
    }

}
```

Листинг 27: SubscriptionTableDAO.java

```java
package com.lamtev.movie_service.datagen.generator;

import com.github.javafaker.Faker;
import gnu.trove.set.TIntSet;
import gnu.trove.set.hash.TIntHashSet;
import org.jetbrains.annotations.NotNull;

import java.sql.Date;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public final class Utils {

    @NotNull
    private final Random random;
    @NotNull
    private final Faker faker;


    public Utils(@NotNull Random random, @NotNull Faker faker) {
        this.random = random;
        this.faker = faker;
    }

    public static void split(final @NotNull int[][] subscriptionIdsNMoviesMSeasons, int
    moviesPercentage, final @NotNull int[][] subscriptionIdsNMovies, final @NotNull int[][]
    subscriptionIdsMSeasons) {
        int moviesIdx = 0;
        int seasonsIdx = 0;
        int moviesLength = (int) Math.ceil((double) subscriptionIdsNMoviesMSeasons[0].length /
     100) * moviesPercentage;
        int seasonsLength = subscriptionIdsNMoviesMSeasons[0].length - moviesLength;
        for (int i = 0; i < 2; ++i) {
            subscriptionIdsNMovies[i] = new int[moviesLength];
            subscriptionIdsMSeasons[i] = new int[seasonsLength];
        }
        for (int i = 0; i < subscriptionIdsNMoviesMSeasons[0].length; ++i) {
            if (i % 100 < moviesPercentage) {
                subscriptionIdsNMovies[0][moviesIdx] = subscriptionIdsNMoviesMSeasons[0][i];
                subscriptionIdsNMovies[1][moviesIdx] = subscriptionIdsNMoviesMSeasons[1][i];
                moviesIdx++;
            } else {
                subscriptionIdsMSeasons[0][seasonsIdx] = subscriptionIdsNMoviesMSeasons[0][i];
                subscriptionIdsMSeasons[1][seasonsIdx] = subscriptionIdsNMoviesMSeasons[2][i];
                seasonsIdx++;
            }
        }
    }

    @NotNull
    public int[] getIdsOfRowsInsertedWith(final @NotNull Statement statement, int ofLength) {
        final var keys = new int[ofLength];
        int i = 0;
        try (final var generatedKeys = statement.getGeneratedKeys()) {
            while (generatedKeys.next()) {
                keys[i++] = generatedKeys.getInt(1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return keys;
    }

    @NotNull
    public TIntSet getIdsOfRowsInsertedWith(final @NotNull Statement statement) {
        final var keys = new TIntHashSet();
        try (final var generatedKeys = statement.getGeneratedKeys()) {
            while (generatedKeys.next()) {
                keys.add(generatedKeys.getInt(1));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return keys;
    }

    @NotNull
    public Date randomDate(int maxYearsAgo) {
        return new Date(faker.date().past(365 * maxYearsAgo, TimeUnit.DAYS).getTime());
    }

    public float randomRating() {
        return 5.0f + random.nextFloat() * (10.0f - 5.0f);
    }

    @NotNull
    public int[] nUniqueRandomInts(int n, int bound) {
        final var ints = new TIntHashSet(n);
        while (ints.size() != n) {
            ints.add(random.nextInt(bound));
        }

        return ints.toArray();
    }

}
```

Листинг 28: Utils.java