

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

_____ А.Ю. Ламтев
(подпись)

Преподаватель

_____ А.В. Мяснов
(подпись)

«___» _____ 2019 г.

Санкт-Петербург
2019

Содержание

1	Цели работы	3
2	Программа работы	3
3	Разработка генератора	3
4	Выводы	8
	Приложение 1. Исходный код	8

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.

```
1 {  
2   "url": "jdbc:postgresql://localhost:5432/postgres",  
3   "user": "postgres",  
4   "password": "postgres"  
5 }
```

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

```
1 {  
2   "usersCount": 10000,  
3   "femalePercentage": 55,  
4   "moviesCount": 25000,  
5   "seriesCountSeasonsEpisodes": [  
6     [100, 3, 15],  
7     [150, 4, 50],  
8     [200, 2, 25],
```

```

9      [300, 1, 7],
10     [100, 5, 10]
11 ],
12 "percentageOfUsersWhoBoughtMovies": 64,
13 "minMoviesPerUser": 5,
14 "maxMoviesPerUser": 100,
15 "percentageOfUsersWhoBoughtSeries": 35,
16 "minSeriesPerUser": 2,
17 "maxSeriesPerUser": 100,
18 "yearsSinceFirstSubscription": 5,
19 "minSubscriptionsPerUser": 7,
20 "maxSubscriptionsPerUser": 25,
21 "durationPriceNMoviesMSeasons": [
22     [30, 5, 5, 1], [60, 9, 5, 1], [90, 13, 5, 1], [180, 25, 5, 1], [365, 40, 5, 1],
23     [30, 7, 10, 2], [60, 13, 10, 2], [90, 25, 10, 2], [180, 45, 10, 2], [365, 75, 10, 2],
24     [30, 10, 15, 3], [60, 18, 15, 3], [90, 35, 15, 3], [180, 65, 15, 3], [365, 100, 15, 3],
25     [30, 12, 30, 5], [60, 21, 30, 5], [90, 40, 30, 5], [180, 70, 30, 5], [365, 120, 30, 5]
26 ],
27 "moviesSubscriptionsPercentage": 25
28 }

```

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.

```

1 plugins {
2     java
3 }
4
5 group = "com.lamtev.movie-service"
6 version = "1.0.RELEASE"
7
8 repositories {
9     jcenter()
10 }
11
12 dependencies {
13     compile("com.intellij:annotations:12.0")
14     compile("org.postgresql:postgresql:42.2.5")
15     compile("com.github.javafaker:javafaker:0.16")
16     compile("net.sf.trove4j:trove4j:3.0.3")
17     compile("com.google.code.gson:gson:2.8.5")
18 }
19
20 configure<JavaPluginConvention> {
21     sourceCompatibility = JavaVersion.VERSION_11
22 }
23
24 val fatJar = task("fatJar", type = Jar::class) {
25     baseName = "${project.group}.${project.name}"
26     manifest {
27         attributes["Implementation-Title"] = "Movie service data generator"
28         attributes["Implementation-Version"] = version
29         attributes["Main-Class"] = "com.lamtev.movie_service.datagen.Launcher"
30     }
31     from(configurations["compile"].map { if (it.isDirectory) it else zipTree(it) })
32     with(tasks["jar"] as CopySpec)
33 }
34
35 tasks {
36     "build" {
37         dependsOn(fatJar)
38     }
39 }

```

Листинг 3: build.gradle.kts

```

1 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.

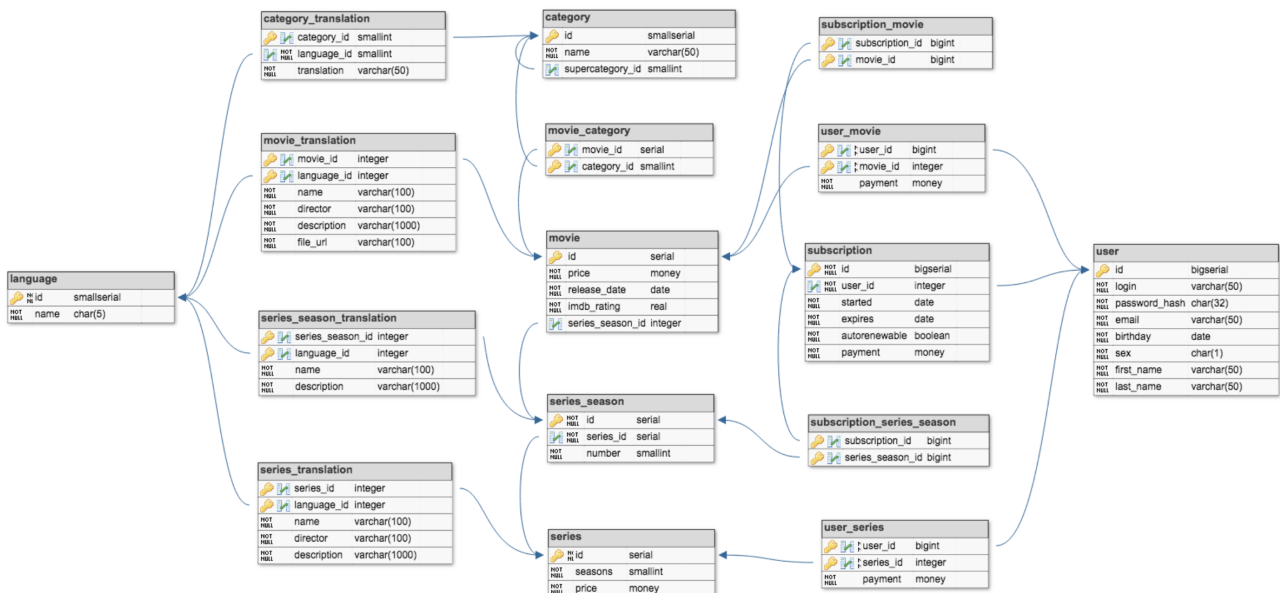


Рис. 3.1: Схема БД

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

```
1 package com.lamtev.movie_service.datagen;
2
3 import com.lamtev.movie_service.datagen.cli_args.ArgumentsParser;
4 import com.lamtev.movie_service.datagen.generator.LanguageTableGenerator;
5 import com.lamtev.movie_service.datagen.generator.StorageDAO;
6 import com.lamtev.movie_service.datagen.generator.Utils;
7 import com.lamtev.movie_service.datagen.generator.category.CategoryTableGenerator;
8 import com.lamtev.movie_service.datagen.generator.movie.MovieTableGenerator;
9 import com.lamtev.movie_service.datagen.generator.series.SeriesTableGenerator;
10 import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionMovieTableGenerator
11 ;
12 import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionSeriesSeasonTableGenerator;
13 import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionTableDAO;
14 import com.lamtev.movie_service.datagen.generator.subscription.SubscriptionTableGenerator;
15 import com.lamtev.movie_service.datagen.generator.user.UserMovieTableGenerator;
16 import com.lamtev.movie_service.datagen.generator.user.UserSeriesTableGenerator;
17 import com.lamtev.movie_service.datagen.generator.user.UserTableGenerator;
18
19 import java.sql.DriverManager;
20 import java.sql.SQLException;
21
22 final class Launcher {
23     public static void main(String[] args) {
24         try {
25             Class.forName("org.postgresql.Driver");
26         } catch (ClassNotFoundException e) {
27             e.printStackTrace();
28         }
29         final var argumentsParser = new ArgumentsParser(args);
30         final var endpoint = argumentsParser.endpoint();
31         final var parameters = argumentsParser.parameters();
32         if (endpoint == null || parameters == null) {
33             System.err.println("Wrong arguments!");
34             return;
35         }
36         try (final var connection = DriverManager.getConnection(endpoint.url(), endpoint.user
37 (), endpoint.password())) {
38             if (parameters.isGenAll()) {
39                 final var language = new LanguageTableGenerator();
40                 language.updateTableUsing(connection);
41                 System.out.println("language generated");
42
43                 final var category = new CategoryTableGenerator();
44                 category.updateTableUsing(connection);
45                 System.out.println("category generated");
46             }
47             if (parameters.isGenAll() || parameters.isGenMoviesOnly()) {
48                 final var movie = new MovieTableGenerator(parameters.moviesCount());
49                 movie.updateTableUsing(connection);
50                 System.out.println("movie generated");
51             }
52             if (parameters.isGenAll() || parameters.isGenSeriesOnly()) {
```



```

54         final var series = new SeriesTableGenerator(parameters.
seriesCountSeasonsEpisodes());
55         series.updateTableUsing(connection);
56         System.out.println("series generated");
57     }
58
59     if (parameters.isGenAll() || parameters.isGenUsersOnly()) {
60         final var user = new UserTableGenerator(parameters.usersCount(), parameters.
femalePercentage());
61         user.updateTableUsing(connection);
62         System.out.println("user generated");
63     }
64
65     if (parameters.isGenAll()) {
66         final var userMovie = new UserMovieTableGenerator(parameters.
percentageOfUsersWhoBoughtMovies(),
67             parameters.minMoviesPerUser(), parameters.maxMoviesPerUser());
68         userMovie.updateTableUsing(connection);
69         System.out.println("user_movie generated");
70
71         final var seriesMovie = new UserSeriesTableGenerator(parameters.
percentageOfUsersWhoBoughtSeries(),
72             parameters.minSeriesPerUser(), parameters.maxSeriesPerUser());
73         seriesMovie.updateTableUsing(connection);
74         System.out.println("user_series generated");
75     }
76
77     if (parameters.isGenAll() || parameters.isGenSubscriptionsToMoviesOnly()
78         || parameters.isGenSubscriptionsToSeriesOnly()) {
79         final var subscription = new SubscriptionTableGenerator(parameters.usersCount
(), parameters.minSubscriptionsPerUser(), parameters.
80             parameters.maxSubscriptionsPerUser(), parameters.
durationPriceNMoviesMSeasons(), parameters.yearsSinceFirstSubscription());
81         subscription.updateTableUsing(connection);
82         System.out.println("subscription generated");
83
84         final var subscriptionIdsNMoviesMSeasons = SubscriptionTableDAO.instance().
idsNMoviesOrMSeasonsContainingInIds(
85             connection, parameters.durationPriceNMoviesMSeasons(), subscription.
getGeneratedIds());
86         final var subscriptionIdsNMovies = new int[2][0];
87         final var subscriptionIdsMSeasons = new int[2][0];
88         Utils.split(subscriptionIdsNMoviesMSeasons, parameters.
moviesSubscriptionsPercentage(), subscriptionIdsNMovies, subscriptionIdsMSeasons);
89
90         if (parameters.isGenAll() || parameters.isGenSubscriptionsToMoviesOnly()) {
91             final var movieIds = StorageDAO.instance().ids(connection, "movie");
92             final var subscriptionMovie = new SubscriptionMovieTableGenerator(
subscriptionIdsNMovies, movieIds);
93             subscriptionMovie.updateTableUsing(connection);
94             System.out.println("subscription_movie generated");
95         }
96
97         if (parameters.isGenAll() || parameters.isGenSubscriptionsToSeriesOnly()) {
98             final var seriesSeasonIds = StorageDAO.instance().ids(connection, "
series_season");
99             final var subscriptionSeriesSeason = new
SubscriptionSeriesSeasonTableGenerator(subscriptionIdsMSeasons, seriesSeasonIds);
100             subscriptionSeriesSeason.updateTableUsing(connection);
101             System.out.println("subscription_series_season generated");
102         }
103     }
104 } catch (SQLException e) {
105     e.printStackTrace();
106 }
107 }
108 }
109 }

```

Листинг 5: Launcher.java

```

1 package com.lamtev.movie_service.datagen.cli_args;
2
3 import com.google.gson.*;
4 import org.jetbrains.annotations.NotNull;

```

```

5 import org.jetbrains.annotations.Nullable;
6
7 import java.io.FileReader;
8 import java.lang.reflect.Type;
9 import java.util.Arrays;
10
11 public class ArgumentsParser {
12
13     @NotNull
14     private final String[] args;
15     @NotNull
16     private final Gson gson;
17
18     public ArgumentsParser(final @NotNull String[] args) {
19         this.args = args;
20         this.gson = new GsonBuilder()
21             .serializeNulls()
22             .registerTypeAdapter(EndpointInfo.class, new Deserializer<EndpointInfo>())
23             .registerTypeAdapter(Parameters.class, new Deserializer<Parameters>())
24             .create();
25     }
26
27     @Nullable
28     public EndpointInfo endpoint() {
29         try (final var fileReader = new FileReader(args[0])) {
30             return gson.fromJson(fileReader, EndpointInfo.class);
31         } catch (Exception e) {
32             System.err.println(e.getMessage());
33             e.printStackTrace();
34             return null;
35         }
36     }
37
38     @Nullable
39     public Parameters parameters() {
40         try (final var fileReader = new FileReader(args[1])) {
41             final var params = gson.fromJson(fileReader, Parameters.class);
42             if (args.length == 3) {
43                 switch (args[2]) {
44                     case "onlyMovies":
45                         params.setGenMoviesOnly(true);
46                         params.setGenAll(false);
47                         break;
48                     case "onlySeries":
49                         params.setGenSeriesOnly(true);
50                         params.setGenAll(false);
51                         break;
52                     case "onlyUsers":
53                         params.setGenUsersOnly(true);
54                         params.setGenAll(false);
55                         break;
56                     case "onlyMovieSubscriptions":
57                         params.setGenSubscriptionsToMoviesOnly(true);
58                         params.setGenAll(false);
59                         break;
60                     case "onlySeriesSubscriptions":
61                         params.setGenSubscriptionsToSeriesOnly(true);
62                         params.setGenAll(false);
63                         break;
64                     default:
65                         params.setGenAll(true);
66                         break;
67                 }
68             } else {
69                 params.setGenAll(true);
70             }
71             return params;
72         } catch (Exception e) {
73             System.err.println(e.getMessage());
74             e.printStackTrace();
75             return null;
76         }
77     }
78
79     class Deserializer<T> implements JsonDeserializer<T> {
80

```

```

81     public T deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext
context) throws JsonParseException {
82         final T obj = new Gson().fromJson(json, typeOfT);
83
84         final var badField = Arrays.stream(obj.getClass().getDeclaredFields())
85             .filter(field -> {
86                 try {
87                     field.setAccessible(true);
88                     return field.get(obj) == null;
89                 } catch (IllegalAccessException | IllegalArgumentException ignored) {
90                     return false;
91                 }
92             })
93             .findFirst();
94
95         if (badField.isPresent()) {
96             throw new JsonParseException("Missing field: " + badField.get().getName());
97         }
98
99         return obj;
100     }
101 }
102
103 }

```

Листинг 6: ArgumentsParser.java

```

1 package com.lamtev.movie_service.datagen.cli_args;
2
3 import org.jetbrains.annotations.NotNull;
4
5 public class EndpointInfo {
6
7     @NotNull
8     private final String url;
9     @NotNull
10    private final String user;
11    @NotNull
12    private final String password;
13
14    public EndpointInfo(@NotNull final String url,
15                        @NotNull final String user,
16                        @NotNull final String password) {
17        this.url = url;
18        this.user = user;
19        this.password = password;
20    }
21
22    @NotNull
23    public String url() {
24        return url;
25    }
26
27    @NotNull
28    public String user() {
29        return user;
30    }
31
32    @NotNull
33    public String password() {
34        return password;
35    }
36
37 }

```

Листинг 7: EndpointInfo.java

```

1 package com.lamtev.movie_service.datagen.cli_args;
2
3 import org.jetbrains.annotations.NotNull;
4
5 public final class Parameters {
6
7     private final int usersCount;

```

```

8 private final int femalePercentage;
9 private final int moviesCount;
10 /**
11  * {{1000, 3, 15}, ...} - 1000 series, each consists of 3 seasons with 15 episodes
12  */
13 @NotNull
14 private final int[][] seriesCountSeasonsEpisodes;
15 private final int percentageOfUsersWhoBoughtMovies;
16 private final int minMoviesPerUser;
17 private final int maxMoviesPerUser;
18 private final int percentageOfUsersWhoBoughtSeries;
19 private final int minSeriesPerUser;
20 private final int maxSeriesPerUser;
21 private final int minSubscriptionsPerUser;
22 private final int maxSubscriptionsPerUser;
23 /**
24  * {{duration in days, price in USD, number of movies, number of series seasons}, ... }
25  */
26 @NotNull
27 private final int[][] durationPriceNMoviesMSeasons;
28 private final int moviesSubscriptionsPercentage;
29 private final int yearsSinceFirstSubscription;
30
31 private boolean genMoviesOnly = false;
32 private boolean genSeriesOnly = false;
33 private boolean genUsersOnly = false;
34 private boolean genSubscriptionsToMoviesOnly = false;
35 private boolean genSubscriptionsToSeriesOnly = false;
36 private boolean genAll = true;
37
38 public Parameters(int usersCount,
39                  int femalePercentage,
40                  int moviesCount,
41                  final @NotNull int[][] seriesCountSeasonsEpisodes,
42                  int percentageOfUsersWhoBoughtMovies,
43                  int minMoviesPerUser,
44                  int maxMoviesPerUser,
45                  int percentageOfUsersWhoBoughtSeries,
46                  int minSeriesPerUser,
47                  int maxSeriesPerUser,
48                  int yearsSinceFirstSubscription,
49                  int minSubscriptionsPerUser,
50                  int maxSubscriptionsPerUser,
51                  @NotNull int[][] durationPriceNMoviesMSeasons,
52                  int moviesSubscriptionsPercentage) {
53     this.usersCount = usersCount;
54     this.femalePercentage = femalePercentage;
55     this.moviesCount = moviesCount;
56     this.seriesCountSeasonsEpisodes = seriesCountSeasonsEpisodes;
57     this.percentageOfUsersWhoBoughtMovies = percentageOfUsersWhoBoughtMovies;
58     this.minMoviesPerUser = minMoviesPerUser;
59     this.maxMoviesPerUser = maxMoviesPerUser;
60     this.percentageOfUsersWhoBoughtSeries = percentageOfUsersWhoBoughtSeries;
61     this.minSeriesPerUser = minSeriesPerUser;
62     this.maxSeriesPerUser = maxSeriesPerUser;
63     this.yearsSinceFirstSubscription = yearsSinceFirstSubscription;
64     this.minSubscriptionsPerUser = minSubscriptionsPerUser;
65     this.maxSubscriptionsPerUser = maxSubscriptionsPerUser;
66     this.durationPriceNMoviesMSeasons = durationPriceNMoviesMSeasons;
67     this.moviesSubscriptionsPercentage = moviesSubscriptionsPercentage;
68 }
69
70 public int femalePercentage() {
71     return femalePercentage;
72 }
73
74 public int percentageOfUsersWhoBoughtMovies() {
75     return percentageOfUsersWhoBoughtMovies;
76 }
77
78 public int minMoviesPerUser() {
79     return minMoviesPerUser;
80 }
81
82 public int maxMoviesPerUser() {
83     return maxMoviesPerUser;

```

```

84     }
85
86     public int percentageOfUsersWhoBoughtSeries() {
87         return percentageOfUsersWhoBoughtSeries;
88     }
89
90     public int minSeriesPerUser() {
91         return minSeriesPerUser;
92     }
93
94     public int maxSeriesPerUser() {
95         return maxSeriesPerUser;
96     }
97
98     public int minSubscriptionsPerUser() {
99         return minSubscriptionsPerUser;
100    }
101
102    public int maxSubscriptionsPerUser() {
103        return maxSubscriptionsPerUser;
104    }
105
106    @NotNull
107    public int [][] durationPriceNMoviesMSeasons() {
108        return durationPriceNMoviesMSeasons;
109    }
110
111    public int moviesSubscriptionsPercentage() {
112        return moviesSubscriptionsPercentage;
113    }
114
115    public int usersCount() {
116        return usersCount;
117    }
118
119    public int moviesCount() {
120        return moviesCount;
121    }
122
123    @NotNull
124    public int [][] seriesCountSeasonsEpisodes() {
125        return seriesCountSeasonsEpisodes;
126    }
127
128    public int yearsSinceFirstSubscription() {
129        return yearsSinceFirstSubscription;
130    }
131
132    public boolean isGenMoviesOnly() {
133        return genMoviesOnly;
134    }
135
136    public void setGenMoviesOnly(boolean genMoviesOnly) {
137        this.genMoviesOnly = genMoviesOnly;
138    }
139
140    public boolean isGenSeriesOnly() {
141        return genSeriesOnly;
142    }
143
144    public void setGenSeriesOnly(boolean genSeriesOnly) {
145        this.genSeriesOnly = genSeriesOnly;
146    }
147
148    public boolean isGenUsersOnly() {
149        return genUsersOnly;
150    }
151
152    public void setGenUsersOnly(boolean genUsersOnly) {
153        this.genUsersOnly = genUsersOnly;
154    }
155
156    public boolean isGenSubscriptionsToMoviesOnly() {
157        return genSubscriptionsToMoviesOnly;
158    }
159

```

```

160     public void setGenSubscriptionsToMoviesOnly(boolean genSubscriptionsToMoviesOnly) {
161         this.genSubscriptionsToMoviesOnly = genSubscriptionsToMoviesOnly;
162     }
163
164     public boolean isGenSubscriptionsToSeriesOnly() {
165         return genSubscriptionsToSeriesOnly;
166     }
167
168     public void setGenSubscriptionsToSeriesOnly(boolean genSubscriptionsToSeriesOnly) {
169         this.genSubscriptionsToSeriesOnly = genSubscriptionsToSeriesOnly;
170     }
171
172     public boolean isGenAll() {
173         return genAll;
174     }
175
176     public void setGenAll(boolean genAll) {
177         this.genAll = genAll;
178     }
179 }
180

```

Листинг 8: Parameters.java

```

1 package com.lamtev.movie_service.datagen.generator;
2
3 import com.github.javafaker.Faker;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.util.Locale;
8 import java.util.Random;
9
10 public interface TableGenerator {
11     @NotNull
12     Random RANDOM = new Random(System.currentTimeMillis());
13     @NotNull
14     Faker FAKER = new Faker(Locale.US, RANDOM);
15     @NotNull
16     Utils UTILS = new Utils(RANDOM, FAKER);
17
18     /**
19      * Updates corresponding table via {@code connection} with newly generated data.
20      *
21      * @param connection {@link Connection} (session) with data base.
22      */
23     void updateTableUsing(final @NotNull Connection connection);
24 }

```

Листинг 9: TableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator;
2
3 import org.jetbrains.annotations.NotNull;
4
5 import java.sql.Connection;
6 import java.sql.SQLException;
7
8 public final class LanguageTableGenerator implements TableGenerator {
9
10     @NotNull
11     private final String[] languages;
12
13     public LanguageTableGenerator(@NotNull String[] languages) {
14         this.languages = languages;
15     }
16
17     public LanguageTableGenerator() {
18         this(new String[]{ "en-US", "ru-RU" });
19     }
20
21     @Override
22     public void updateTableUsing(final @NotNull Connection connection) {
23         try (final var statement = connection.prepareStatement(

```

```

24         "INSERT INTO language (name) VALUES (?)"
25     )) {
26         for (final var language : languages) {
27             statement.setString(1, language);
28             statement.addBatch();
29         }
30         statement.executeBatch();
31     } catch (SQLException e) {
32         e.printStackTrace();
33     }
34 }
35
36 }

```

Листинг 10: LanguageTableGenerator.java

```

1  package com.lamtev.movie_service.datagen.generator.category;
2
3  import com.lamtev.movie_service.datagen.generator.TableGenerator;
4  import org.jetbrains.annotations.NotNull;
5
6  import java.sql.Connection;
7  import java.sql.SQLException;
8  import java.util.LinkedHashMap;
9  import java.util.Map;
10
11 import static java.sql.Statement.RETURN_GENERATED_KEYS;
12
13 public final class CategoryTableGenerator implements TableGenerator {
14
15     private static final Map<String, String> CATEGORY_TO_SUPERCATEGORY = new LinkedHashMap<>()
16     {{
17         put("genre", "");
18         put("comedy", "genre");
19         put("drama", "genre");
20         put("thriller", "genre");
21         put("new", "");
22         put("horror", "genre");
23         put("action", "genre");
24         put("crime", "genre");
25         put("western", "genre");
26         put("popular", "");
27         put("mystery", "genre");
28         put("adventure", "genre");
29         put("classic", "");
30         put("romance", "genre");
31         put("science-fiction", "genre");
32         put("soviet", "");
33         put("hollywood", "");
34     }};
35
36     @Override
37     public void updateTableUsing(final @NotNull Connection connection) {
38         try (final var statement = connection.createStatement()) {
39             final var categoryIds = new int[CATEGORY_TO_SUPERCATEGORY.size()];
40             int i = 0;
41             for (final var entry : CATEGORY_TO_SUPERCATEGORY.entrySet()) {
42                 final var category = entry.getKey();
43                 final var supercategory = entry.getValue();
44
45                 final var query = String.format(
46                     "INSERT INTO category (name, supercategory_id) " +
47                     "SELECT '%s', (SELECT id FROM category WHERE name = '%s' LIMIT
48                     1)", category, supercategory
49                 );
50                 try {
51                     statement.executeUpdate(query, RETURN_GENERATED_KEYS);
52                     final var generatedKeys = statement.getGeneratedKeys();
53                     if (generatedKeys.next()) {
54                         categoryIds[i] = generatedKeys.getInt(1);
55                     }
56                     i++;
57                 } catch (SQLException e) {
58                     e.printStackTrace();
59                 }
60             }
61         }
62     }
63 }

```

```

58         }
59         final var categoryTranslation = new CategoryTranslationTableGenerator(categoryIds)
60     ;
61     categoryTranslation.updateTableUsing(connection);
62 } catch (SQLException e) {
63     e.printStackTrace();
64 }
65 }
66 }

```

Листинг 11: CategoryTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.category;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 public final class CategoryTranslationTableGenerator implements TableGenerator {
10
11     @NotNull
12     private final int[] categoryIds;
13
14     public CategoryTranslationTableGenerator(final @NotNull int[] categoryIds) {
15         this.categoryIds = categoryIds;
16     }
17
18     @Override
19     public void updateTableUsing(final @NotNull Connection connection) {
20         try (final var statement = connection.prepareStatement(
21             "INSERT INTO category_translation (category_id, language_id, translation)
22             VALUES (?, ?, ?)"
23         )) {
24             for (final var categoryId : categoryIds) {
25                 for (int languageId = 1; languageId <= 2; ++languageId) {
26                     int i = 0;
27                     statement.setInt(++i, categoryId);
28                     statement.setInt(++i, languageId);
29                     statement.setString(++i, FAKER.lorem().word());
30                     statement.addBatch();
31                 }
32             }
33             statement.executeBatch();
34         } catch (SQLException e) {
35             e.printStackTrace();
36         }
37     }
38 }

```

Листинг 12: CategoryTranslationTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.movie;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5 import org.postgresql.util.PGmoney;
6
7 import java.sql.Connection;
8 import java.sql.SQLException;
9 import java.sql.Types;
10
11 import static java.sql.Statement.RETURN_GENERATED_KEYS;
12
13 public final class MovieTableGenerator implements TableGenerator {
14
15     private static final short[] MOVIE_PRICES_IN_USD = new short[]{5, 5, 5, 5, 5, 7, 7, 7, 7,
16     10, 10, 10, 15, 15, 20, 25, 35};
17     private final int movieCount;
18     private final int seriesSeasonId;
19     private final int seriesPrice;

```



```

19
20 public MovieTableGenerator(int count) {
21     this(count, 0, 0);
22 }
23
24 public MovieTableGenerator(int movieCount, int seriesSeasonId, int seriesPrice) {
25     this.movieCount = movieCount;
26     this.seriesSeasonId = seriesSeasonId;
27     this.seriesPrice = seriesPrice;
28 }
29
30 @Override
31 public void updateTableUsing(final @NotNull Connection connection) {
32     try (final var statement = connection.prepareStatement(
33         "INSERT INTO movie (price, release_date, imdb_rating, series_season_id) VALUES
34         (?, ?, ?, ?)",
35         RETURN_GENERATED_KEYS
36     )) {
37         final var moviesAreSeriesSeasonEpisodes = seriesSeasonId != 0;
38         final var date = UTILS.randomDate(50);
39         final var rating = UTILS.randomRating();
40         for (int i = 0; i < movieCount; ++i) {
41             int j = 0;
42             statement.setObject(++j, new PGmoney("$" + (
43                 seriesPrice == 0 ?
44                 MOVIE_PRICES_IN_USD[RANDOM.nextInt(MOVIE_PRICES_IN_USD.length)
45                 : seriesPrice
46                 ]));
47             if (moviesAreSeriesSeasonEpisodes) {
48                 statement.setDate(++j, date);
49                 statement.setFloat(++j, rating);
50                 statement.setInt(++j, seriesSeasonId);
51             } else {
52                 statement.setDate(++j, UTILS.randomDate(50));
53                 statement.setFloat(++j, UTILS.randomRating());
54                 statement.setNull(++j, Types.INTEGER);
55             }
56             statement.addBatch();
57         }
58         statement.executeBatch();
59
60         final var movieIds = UTILS.getIdsOfRowsInsertedWith(statement, movieCount);
61
62         final var movieTranslation = new MovieTranslationTableGenerator(movieIds,
63             moviesAreSeriesSeasonEpisodes);
64         movieTranslation.updateTableUsing(connection);
65
66         updateMovieCategoryTableUsing(connection, moviesAreSeriesSeasonEpisodes, movieIds)
67         ;
68     } catch (SQLException e) {
69         e.printStackTrace();
70     }
71 }
72
73 private void updateMovieCategoryTableUsing(@NotNull Connection connection, boolean
74     moviesAreSeriesSeasonEpisodes, int[] movieIds) {
75     try (final var categoriesStatement = connection.createStatement()) {
76         categoriesStatement.executeQuery("SELECT COUNT(*) FROM category");
77         var result = categoriesStatement.getResultSet();
78         if (result != null && result.next()) {
79             int categoriesCount = result.getInt(1);
80             final var categoryIds = new int[categoriesCount - 1];
81             int i = 0;
82             categoriesStatement.executeQuery("SELECT id FROM category WHERE name != 'genre
83             '");
84             result = categoriesStatement.getResultSet();
85             if (result != null) {
86                 while (result.next()) {
87                     categoryIds[i++] = result.getShort(1);
88                 }
89             }
90
91             final var movieCategory = new MovieCategoryTableGenerator(movieIds,
92                 categoryIds, moviesAreSeriesSeasonEpisodes);
93             movieCategory.updateTableUsing(connection);
94         }
95     }
96 }

```

```

88     }
89     } catch (SQLException e) {
90         e.printStackTrace();
91     }
92 }
93
94 }

```

ЛИСТИНГ 13: MovieTableGenerator.java

```

1  package com.lamtev.movie_service.datagen.generator.movie;
2
3  import com.lamtev.movie_service.datagen.generator.TableGenerator;
4  import org.jetbrains.annotations.NotNull;
5
6  import java.sql.Connection;
7  import java.sql.SQLException;
8
9  public final class MovieTranslationTableGenerator implements TableGenerator {
10
11     private static final String VIDEO_URL_TEMPLATE = "https://blob.movie-service.lamtev.com/?vid=";
12
13     @NotNull
14     private final int[] movieIds;
15     private final boolean moviesAreSeriesEpisodes;
16
17     public MovieTranslationTableGenerator(final @NotNull int[] movieIds, boolean moviesAreSeriesEpisodes) {
18         this.movieIds = movieIds;
19         this.moviesAreSeriesEpisodes = moviesAreSeriesEpisodes;
20     }
21
22     @Override
23     public void updateTableUsing(final @NotNull Connection connection) {
24         try (final var statement = connection.prepareStatement(
25             "INSERT INTO movie_translation (movie_id, language_id, name, director,
26             description, file_url) " +
27             "VALUES (?, ?, ?, ?, ?, ?)"
28         )) {
29             final var director = moviesAreSeriesEpisodes ? FAKER.artist().name() : null;
30             for (int movieIdx = 0; movieIdx < movieIds.length; ++movieIdx) {
31                 for (int languageId = 1; languageId <= 2; ++languageId) {
32                     int i = 0;
33                     statement.setInt(++i, movieIds[movieIdx]);
34                     statement.setInt(++i, languageId);
35                     if (moviesAreSeriesEpisodes) {
36                         statement.setString(++i, "Episode " + movieIdx);
37                         statement.setString(++i, director);
38                     } else {
39                         final var movie = FAKER.book();
40                         statement.setString(++i, movie.title());
41                         statement.setString(++i, movie.author());
42                     }
43                     statement.setString(++i, FAKER.lorem().paragraph(10));
44                     statement.setString(++i, randomUrl());
45                     statement.addBatch();
46                 }
47             }
48             statement.executeBatch();
49         } catch (SQLException e) {
50             e.printStackTrace();
51         }
52     }
53
54     private String randomUrl() {
55         return VIDEO_URL_TEMPLATE + RANDOM.nextInt(Integer.MAX_VALUE);
56     }
57 }

```

ЛИСТИНГ 14: MovieTranslationTableGenerator.java

```

1  package com.lamtev.movie_service.datagen.generator.movie;

```

```

2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import gnu.trove.list.TIntList;
5 import gnu.trove.list.array.TIntArrayList;
6 import org.jetbrains.annotations.NotNull;
7
8 import java.sql.Connection;
9 import java.sql.SQLException;
10 import java.util.Arrays;
11
12 public final class MovieCategoryTableGenerator implements TableGenerator {
13
14     @NotNull
15     private final int[] movieIds;
16     @NotNull
17     private final TIntList categoryIds;
18     private final boolean sameCategoriesForAllMovies;
19
20     public MovieCategoryTableGenerator(final @NotNull int[] movieIds, final @NotNull int[]
categoryIds, boolean sameCategoriesForAllMovies) {
21         this.movieIds = movieIds;
22         this.categoryIds = new TIntArrayList(categoryIds.length);
23         Arrays.stream(categoryIds).forEach(this.categoryIds::add);
24         this.sameCategoriesForAllMovies = sameCategoriesForAllMovies;
25     }
26
27     @Override
28     public void updateTableUsing(final @NotNull Connection connection) {
29         try (final var statement = connection.prepareStatement(
30             "INSERT INTO movie_category (movie_id, category_id) VALUES (?, ?)"
31         )) {
32             final var categories = nRandomCategories(3);
33             for (int movieId : movieIds) {
34                 final var differentCategories = nRandomCategories(3);
35                 for (int j = 0; j < categories.length; ++j) {
36                     int i = 0;
37                     statement.setInt(++i, movieId);
38                     if (sameCategoriesForAllMovies) {
39                         statement.setInt(++i, categories[j]);
40                     } else {
41                         statement.setInt(++i, differentCategories[j]);
42                     }
43                     statement.addBatch();
44                 }
45             }
46             statement.executeBatch();
47         } catch (SQLException e) {
48             e.printStackTrace();
49         }
50     }
51
52     @NotNull
53     private int[] nRandomCategories(int n) {
54         categoryIds.shuffle(RANDOM);
55         final var res = new int[n];
56         for (int i = 0; i < n; ++i) {
57             res[i] = categoryIds.get(i);
58         }
59
60         return res;
61     }
62 }
63 }

```

Листинг 15: MovieCategoryTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.series;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import com.lamtev.movie_service.datagen.generator.series.season.SeriesSeasonTableGenerator;
5 import org.jetbrains.annotations.NotNull;
6 import org.postgresql.util.PGmoney;
7
8 import java.sql.Connection;
9 import java.sql.SQLException;

```

```

10
11 import static java.sql.Statement.RETURN_GENERATED_KEYS;
12
13 public final class SeriesTableGenerator implements TableGenerator {
14
15     private static final short[] SERIES_PRICES_IN_USD = new short[]{10, 10, 10, 10, 17, 17,
16     17, 25, 25, 35};
17
18     /**
19     * [[1000, 3, 15], ...] - 1000 series, each consists of 3 seasons with 15 episodes
20     */
21     @NotNull
22     private final int[][] countSeasonsEpisodesArray;
23
24     public SeriesTableGenerator(final @NotNull int[][] countSeasonsEpisodes) {
25         this.countSeasonsEpisodesArray = countSeasonsEpisodes;
26     }
27
28     @Override
29     public void updateTableUsing(final @NotNull Connection connection) {
30         try (final var statement = connection.prepareStatement(
31             "INSERT INTO series (seasons, price) VALUES (?, ?)",
32             RETURN_GENERATED_KEYS
33         )) {
34             for (final var countSeasonsEpisodes : countSeasonsEpisodesArray) {
35                 final int seriesCount = countSeasonsEpisodes[0];
36                 final short seasons = (short) countSeasonsEpisodes[1];
37                 final var seriesPrices = new int[seriesCount];
38                 for (int seriesIdx = 0; seriesIdx < seriesCount; ++seriesIdx) {
39                     final int seriesPrice = SERIES_PRICES_IN_USD[RANDOM.nextInt(
40                     SERIES_PRICES_IN_USD.length)];
41                     seriesPrices[seriesIdx] = seriesPrice;
42                     int i = 0;
43                     statement.setShort(++i, seasons);
44                     statement.setObject(++i, new PGmoney("$" + seriesPrice));
45                     statement.addBatch();
46                 }
47                 statement.executeBatch();
48
49                 final var seriesIds = UTILS.getIdsOfRowsInsertedWith(statement, seriesCount);
50
51                 final var seriesTranslation = new SeriesTranslationTableGenerator(seriesIds);
52                 seriesTranslation.updateTableUsing(connection);
53
54                 final int episodes = countSeasonsEpisodes[2];
55                 final var seriesSeason = new SeriesSeasonTableGenerator(seriesIds,
56                 seriesPrices, seasons, episodes);
57                 seriesSeason.updateTableUsing(connection);
58             }
59         } catch (SQLException e) {
60             e.printStackTrace();
61         }
62     }
63 }

```

Листинг 16: SeriesTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.series;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 public final class SeriesTranslationTableGenerator implements TableGenerator {
10
11     @NotNull
12     private final int[] seriesIds;
13
14     public SeriesTranslationTableGenerator(final @NotNull int[] seriesIds) {
15         this.seriesIds = seriesIds;
16     }
17

```

```

18  @Override
19  public void updateTableUsing(final @NotNull Connection connection) {
20      try (final var statement = connection.prepareStatement(
21          "INSERT INTO series_translation (series_id, language_id, name, director,
22          description) VALUES (?, ?, ?, ?, ?)"
23      )) {
24          for (final var seriesId : seriesIds) {
25              for (int languageId = 1; languageId <= 2; ++languageId) {
26                  int i = 0;
27                  statement.setInt(++i, seriesId);
28                  statement.setInt(++i, languageId);
29                  final var series = FAKER.book();
30                  statement.setString(++i, series.title());
31                  statement.setString(++i, series.author());
32                  statement.setString(++i, FAKER.lorem().paragraph(10));
33                  statement.addBatch();
34              }
35          }
36          statement.executeBatch();
37      } catch (SQLException e) {
38          e.printStackTrace();
39      }
40  }
41  }

```

ЛИСТИНГ 17: SeriesTranslationTableGenerator.java

```

1  package com.lamtev.movie_service.datagen.generator.series.season;
2
3  import com.lamtev.movie_service.datagen.generator.TableGenerator;
4  import com.lamtev.movie_service.datagen.generator.movie.MovieTableGenerator;
5  import org.jetbrains.annotations.NotNull;
6
7  import java.sql.Connection;
8  import java.sql.SQLException;
9
10 import static java.sql.Statement.RETURN_GENERATED_KEYS;
11
12 public final class SeriesSeasonTableGenerator implements TableGenerator {
13
14     @NotNull
15     private final int[] seriesIds;
16     @NotNull
17     private final int[] seriesPrices;
18     private final short seasonsCount;
19     private final int episodesCount;
20
21     public SeriesSeasonTableGenerator(final @NotNull int[] seriesIds,
22                                     final @NotNull int[] seriesPrices, short seasonsCount,
23                                     int episodesCount) {
24         this.seriesIds = seriesIds;
25         this.seriesPrices = seriesPrices;
26         this.seasonsCount = seasonsCount;
27         this.episodesCount = episodesCount;
28     }
29
30     @Override
31     public void updateTableUsing(final @NotNull Connection connection) {
32         try (final var statement = connection.prepareStatement(
33             "INSERT INTO series_season (series_id, number) VALUES (?, ?)",
34             RETURN_GENERATED_KEYS
35         )) {
36             for (final int id : seriesIds) {
37                 for (short season = 0; season < seasonsCount; ++season) {
38                     int i = 0;
39                     statement.setInt(++i, id);
40                     statement.setShort(++i, season);
41                     statement.addBatch();
42                 }
43             }
44             statement.executeBatch();
45
46             final var seasonIds = UTILS.getIdsOfRowsInsertedWith(statement, seriesIds.length *
47                 seasonsCount);

```

```

46         final var seasonTranslation = new SeriesSeasonTranslationTableGenerator(seasonIds)
47     ;
48     seasonTranslation.updateTableUsing(connection);
49
50     for (int seasonIdx = 0; seasonIdx < seasonIds.length; ++seasonIdx) {
51         final var episode = new MovieTableGenerator(episodeCount, seasonIds[seasonIdx]
52     ], seriesPrices[seasonIdx / seasonsCount]);
53         episode.updateTableUsing(connection);
54     }
55     } catch (SQLException e) {
56         e.printStackTrace();
57     }
58 }
59 }

```

Листинг 18: SeriesSeasonTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.series.season;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 public final class SeriesSeasonTranslationTableGenerator implements TableGenerator {
10
11     @NotNull
12     private final int[] seasonIds;
13
14     public SeriesSeasonTranslationTableGenerator(final @NotNull int[] seasonIds) {
15         this.seasonIds = seasonIds;
16     }
17
18     @Override
19     public void updateTableUsing(final @NotNull Connection connection) {
20         try (final var statement = connection.prepareStatement(
21             "INSERT INTO series_season_translation (series_season_id, language_id, name,
22             description) " +
23             " VALUES (?, ?, ?, ?)"
24         )) {
25             for (int languageId = 1; languageId <= 2; ++languageId) {
26                 for (final var seasonId : seasonIds) {
27                     int i = 0;
28                     statement.setInt(++i, seasonId);
29                     statement.setInt(++i, languageId);
30                     statement.setString(++i, FAKER.book().title());
31                     statement.setString(++i, FAKER.lorem().paragraph(10));
32                     statement.addBatch();
33                 }
34             }
35             statement.executeBatch();
36         } catch (SQLException e) {
37             e.printStackTrace();
38         }
39     }
40 }

```

Листинг 19: SeriesSeasonTranslationTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.user;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 import static java.sql.Statement.RETURN_GENERATED_KEYS;
10
11 public final class UserTableGenerator implements TableGenerator {

```

```

12
13 private static byte[] buf = null;
14 private final long count;
15 private final int femalePercent;
16
17 public UserTableGenerator(long count, int femalePercentage) {
18     this.count = count;
19     this.femalePercent = femalePercentage;
20 }
21
22 @Override
23 public void updateTableUsing(final @NotNull Connection connection) {
24     try (final var statement = connection.prepareStatement(
25         "INSERT INTO \"user\" (login, password_hash, email, birthday, sex, first_name,
26         last_name) " +
27         "VALUES (?, ?, ?, ?, ?, ?, ?)",
28         RETURN_GENERATED_KEYS
29     )) {
30         for (int i = 0; i < count; ++i) {
31             int j = 0;
32             final var firstName = FAKER.name().firstName();
33             final var lastName = FAKER.name().lastName();
34             final var username = firstName + "." + lastName + RANDOM.nextInt((int) count);
35             statement.setString(++j, username);
36             statement.setString(++j, Long.toHexString(FAKER.number().randomNumber()));
37             statement.setString(++j, username + "@email.com");
38             statement.setDate(++j, UTILS.randomDate(100));
39             statement.setString(++j, randomSex());
40             statement.setString(++j, firstName);
41             statement.setString(++j, lastName);
42             statement.addBatch();
43         }
44         statement.executeBatch();
45     } catch (SQLException e) {
46         e.printStackTrace();
47     }
48 }
49
50 private String randomSex() {
51     if (buf == null) {
52         buf = new byte[100];
53         for (int i = 0; i < buf.length; ++i) {
54             if (i < femalePercent) {
55                 buf[i] = 0;
56             } else {
57                 buf[i] = 1;
58             }
59         }
60     }
61     return Byte.toString(buf[RANDOM.nextInt(100)]);
62 }
63
64 }

```

Листинг 20: UserTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.user;
2
3 import com.lamtev.movie_service.datagen.generator.StorageDAO;
4 import com.lamtev.movie_service.datagen.generator.TableGenerator;
5 import org.jetbrains.annotations.NotNull;
6 import org.jetbrains.annotations.Nullable;
7 import org.postgresql.util.PGmoney;
8
9 import java.sql.Connection;
10 import java.sql.SQLException;
11
12 public final class UserMovieTableGenerator implements TableGenerator {
13
14     private final int percentageOfUsersWhoBoughtMovies;
15     private final int minMovies;
16     private final int maxMovies;
17
18     public UserMovieTableGenerator(int percentageOfUsersWhoBoughtMovies, int minMovies, int

```

```

18     maxMovies) {
19         this.percentageOfUsersWhoBoughtMovies = percentageOfUsersWhoBoughtMovies;
20         this.minMovies = minMovies;
21         this.maxMovies = maxMovies;
22     }
23
24     @Override
25     public void updateTableUsing(final @NotNull Connection connection) {
26         final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
27         final var movieIdsPrices = movieIdsPrices(connection);
28         if (userIds.length == 0 || movieIdsPrices == null) {
29             return;
30         }
31
32         try (final var statement = connection.prepareStatement(
33             "INSERT INTO user_movie (user_id, movie_id, payment) VALUES (?, ?, ?)"
34         )) {
35             for (final var userId : userIds) {
36                 if (userId % 100 < percentageOfUsersWhoBoughtMovies) {
37                     final var nMovies = RANDOM.nextInt(maxMovies - minMovies + 1) + minMovies;
38                     final var movieIdx = RANDOM.nextInt(movieIdsPrices[0].length - nMovies);
39                     for (int i = 0; i < nMovies; ++i) {
40                         int j = 0;
41                         statement.setLong(++j, userId);
42                         statement.setInt(++j, movieIdsPrices[0][movieIdx + i]);
43                         statement.setObject(++j, new PGmoney("$" + movieIdsPrices[1][movieIdx
44 + i]));
45                         statement.addBatch();
46                     }
47                 }
48                 statement.executeBatch();
49             } catch (SQLException e) {
50                 e.printStackTrace();
51             }
52         }
53
54         @Nullable
55         private int[][] movieIdsPrices(final @NotNull Connection connection) {
56             try (final var statement = connection.createStatement()) {
57                 int count = StorageDAO.instance().count(connection, "movie");
58                 int[][] movieIdsPrices = new int[2][count];
59                 statement.executeQuery("SELECT id, price FROM movie");
60                 final var result = statement.getResultSet();
61                 int i = 0;
62                 if (result != null) {
63                     while (result.next()) {
64                         movieIdsPrices[0][i] = result.getInt(1);
65                         movieIdsPrices[1][i] = result.getInt(2);
66                         i++;
67                     }
68                 }
69
70                 return movieIdsPrices;
71             } catch (SQLException e) {
72                 e.printStackTrace();
73             }
74
75             return null;
76         }
77     }
78 }

```

Листинг 21: UserMovieTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.user;
2
3 import com.lamtev.movie_service.datagen.generator.StorageDAO;
4 import com.lamtev.movie_service.datagen.generator.TableGenerator;
5 import org.jetbrains.annotations.NotNull;
6 import org.jetbrains.annotations.Nullable;
7 import org.postgresql.util.PGmoney;
8
9 import java.sql.Connection;
10 import java.sql.SQLException;

```



```

11
12 public final class UserSeriesTableGenerator implements TableGenerator {
13
14     private final int percentageOfUsersWhoBoughtSeries;
15     private final int minSeries;
16     private final int maxSeries;
17
18     public UserSeriesTableGenerator(int percentageOfUsersWhoBoughtSeries, int minSeries, int
maxSeries) {
19         this.percentageOfUsersWhoBoughtSeries = percentageOfUsersWhoBoughtSeries;
20         this.minSeries = minSeries;
21         this.maxSeries = maxSeries;
22     }
23
24     //TODO: get rid of duplicates
25     @Override
26     public void updateTableUsing(final @NotNull Connection connection) {
27         final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
28         final var seriesIdsPrices = seriesIdsPrices(connection);
29         if (userIds.length == 0 || seriesIdsPrices == null) {
30             return;
31         }
32         try (final var statement = connection.prepareStatement(
33             "INSERT INTO user_series (user_id, series_id, payment) VALUES (?, ?, ?)"
34         )) {
35             for (final var userId : userIds) {
36                 if (userId % 100 < percentageOfUsersWhoBoughtSeries) {
37                     final var nSeries = RANDOM.nextInt(maxSeries - minSeries + 1) + minSeries;
38                     final var seriesIdx = RANDOM.nextInt(seriesIdsPrices[0].length - nSeries);
39                     for (int i = 0; i < nSeries; ++i) {
40                         int j = 0;
41                         statement.setLong(++j, userId);
42                         statement.setInt(++j, seriesIdsPrices[0][seriesIdx + i]);
43                         statement.setObject(++j, new PGmoney("$" + seriesIdsPrices[1][
seriesIdx + i]));
44                         statement.addBatch();
45                     }
46                 }
47             }
48             statement.executeBatch();
49         } catch (SQLException e) {
50             e.printStackTrace();
51         }
52     }
53
54     @Nullable
55     private int[][] seriesIdsPrices(final @NotNull Connection connection) {
56         try (final var statement = connection.createStatement()) {
57             int count = StorageDAO.instance().count(connection, "series");
58             int[][] seriesIdsPrices = new int[2][count];
59             statement.executeQuery("SELECT id, price FROM series");
60             final var result = statement.getResultSet();
61             int i = 0;
62             if (result != null) {
63                 while (result.next()) {
64                     seriesIdsPrices[0][i] = result.getInt(1);
65                     seriesIdsPrices[1][i] = result.getInt(2);
66                     i++;
67                 }
68             }
69
70             return seriesIdsPrices;
71         } catch (SQLException e) {
72             e.printStackTrace();
73         }
74
75         return null;
76     }
77 }
78

```

Листинг 22: UserSeriesTableGenerator.java

```

1 package com.lamtev.movie_service.datagen.generator.subscription;
2

```

```

3 import com.lamtev.movie_service.datagen.generator.StorageDAO;
4 import com.lamtev.movie_service.datagen.generator.TableGenerator;
5 import gnu.trove.set.TIntSet;
6 import org.jetbrains.annotations.NotNull;
7 import org.postgresql.util.PGmoney;
8
9 import java.sql.Connection;
10 import java.sql.Date;
11 import java.sql.SQLException;
12 import java.util.Calendar;
13
14 public final class SubscriptionTableGenerator implements TableGenerator {
15
16     private final long usersCount;
17     private final int minSubscriptionsPerUser;
18     private final int maxSubscriptionsPerUser;
19     @NotNull
20     private final int[][] durationPriceNMoviesMSeasons;
21     private final int yearsSinceFirstSubscription;
22     private TIntSet generatedIds;
23
24     public SubscriptionTableGenerator(long usersCount, int minSubscriptionsPerUser,
25                                     int maxSubscriptionsPerUser, final @NotNull int[][]
26                                     durationPriceNMoviesMSeasons,
27                                     int yearsSinceFirstSubscription) {
28         this.usersCount = usersCount;
29         this.minSubscriptionsPerUser = minSubscriptionsPerUser;
30         this.maxSubscriptionsPerUser = maxSubscriptionsPerUser;
31         this.durationPriceNMoviesMSeasons = durationPriceNMoviesMSeasons;
32         this.yearsSinceFirstSubscription = yearsSinceFirstSubscription;
33     }
34
35     @Override
36     public void updateTableUsing(final @NotNull Connection connection) {
37         final var userIds = StorageDAO.instance().ids(connection, "\"user\"");
38         try (final var statement = connection.prepareStatement(
39             "INSERT INTO subscription (user_id, started, expires, autorenewable, payment)
40             VALUES (?, ?, ?, ?, ?)"
41         )) {
42             RANDOM.ints(usersCount, 0, userIds.length).forEach(idx -> {
43                 final var nSubscriptions = RANDOM.nextInt(maxSubscriptionsPerUser -
44                 minSubscriptionsPerUser + 1) + minSubscriptionsPerUser;
45                 for (int j = 0; j < nSubscriptions; ++j) {
46                     int i = 0;
47                     try {
48                         statement.setLong(++i, userIds[idx]);
49                         final var started = UTILS.randomDate(yearsSinceFirstSubscription);
50                         statement.setObject(++i, started);
51                         final var calendar = Calendar.getInstance();
52                         calendar.setTimeInMillis(started.getTime());
53                         final var durationPrice = durationPriceNMoviesMSeasons[RANDOM.nextInt(
54                         durationPriceNMoviesMSeasons.length)];
55                         calendar.add(Calendar.DATE, durationPrice[0]);
56                         statement.setObject(++i, new Date(calendar.getTimeInMillis()));
57                         statement.setBoolean(++i, RANDOM.nextBoolean());
58                         statement.setObject(++i, new PGmoney("$" + durationPrice[1]));
59                         statement.addBatch();
60                     } catch (SQLException e) {
61                         e.printStackTrace();
62                     }
63                 }
64             });
65             statement.executeBatch();
66             generatedIds = UTILS.getIdsOfRowsInsertedWith(statement);
67         } catch (SQLException e) {
68             e.printStackTrace();
69         }
70     }
71
72     public TIntSet getGeneratedIds() {
73         return generatedIds;
74     }
75 }

```

Листинг 23: SubscriptionTableGenerator.java

```
1 package com.lamtev.movie_service.datagen.generator.subscription;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 public final class SubscriptionMovieTableGenerator implements TableGenerator {
10
11     @NotNull
12     private final int[][] subscriptionIdsNMovies;
13     @NotNull
14     private final int[] movieIds;
15
16     public SubscriptionMovieTableGenerator(final @NotNull int[][] subscriptionIdsNMovies,
17     final @NotNull int[] movieIds) {
18         this.subscriptionIdsNMovies = subscriptionIdsNMovies;
19         this.movieIds = movieIds;
20     }
21
22     @Override
23     public void updateTableUsing(final @NotNull Connection connection) {
24         try (final var statement = connection.prepareStatement(
25             "INSERT INTO subscription_movie (subscription_id, movie_id) VALUES (?, ?)"
26         )) {
27             for (int j = 0; j < subscriptionIdsNMovies[0].length; ++j) {
28                 final var nMovies = subscriptionIdsNMovies[1][j];
29                 final var movieIdsIdxs = UTILS.nUniqueRandomInts(nMovies, movieIds.length);
30                 for (final var movieIdsIdx : movieIdsIdxs) {
31                     int i = 0;
32                     statement.setLong(++i, subscriptionIdsNMovies[0][j]);
33                     statement.setInt(++i, movieIds[movieIdsIdx]);
34                     statement.addBatch();
35                 }
36                 statement.executeBatch();
37             } catch (SQLException e) {
38                 e.printStackTrace();
39             }
40         }
41     }
42 }
```

Листинг 24: SubscriptionMovieTableGenerator.java

```
1 package com.lamtev.movie_service.datagen.generator.subscription;
2
3 import com.lamtev.movie_service.datagen.generator.TableGenerator;
4 import org.jetbrains.annotations.NotNull;
5
6 import java.sql.Connection;
7 import java.sql.SQLException;
8
9 public final class SubscriptionSeriesSeasonTableGenerator implements TableGenerator {
10
11     @NotNull
12     private final int[][] subscriptionIdsMSeasons;
13     @NotNull
14     private final int[] seriesSeasonIds;
15
16     public SubscriptionSeriesSeasonTableGenerator(final @NotNull int[][]
17     subscriptionIdsMSeasons, final @NotNull int[] seriesSeasonIds) {
18         this.subscriptionIdsMSeasons = subscriptionIdsMSeasons;
19         this.seriesSeasonIds = seriesSeasonIds;
20     }
21
22     @Override
23     public void updateTableUsing(final @NotNull Connection connection) {
24         try (final var statement = connection.prepareStatement(
```

```

24         "INSERT INTO subscription_series_season (subscription_id, series_season_id)
VALUES (?, ?)"
25     )) {
26         for (int j = 0; j < subscriptionIdsMSeasons[0].length; ++j) {
27             final var nSeriesSeasons = subscriptionIdsMSeasons[1][j];
28             final var seriesIdsIdxs = UTILS.nUniqueRandomInts(nSeriesSeasons,
seriesSeasonIds.length);
29             for (final var seriesIdsIdx : seriesIdsIdxs) {
30                 int i = 0;
31                 statement.setLong(++i, subscriptionIdsMSeasons[0][j]);
32                 statement.setInt(++i, seriesSeasonIds[seriesIdsIdx]);
33                 statement.addBatch();
34             }
35             statement.executeBatch();
36         } catch (SQLException e) {
37             e.printStackTrace();
38         }
39     }
40 }
41
42 }

```

Листинг 25: SubscriptionSeriesSeasonTableGenerator.java

```

1  package com.lamtev.movie_service.datagen.generator;
2
3  import org.jetbrains.annotations.NotNull;
4
5  import java.sql.Connection;
6  import java.sql.SQLException;
7
8  public final class StorageDAO {
9
10     private StorageDAO() {
11     }
12
13     public static StorageDAO instance() {
14         return Holder.INSTANCE;
15     }
16
17     public final int count(final @NotNull Connection connection, final @NotNull String
tableName) {
18         int count = 0;
19         try (final var statement = connection.createStatement()) {
20             statement.executeQuery("SELECT COUNT(*) FROM " + tableName);
21             var result = statement.getResultSet();
22             if (result != null && result.next()) {
23                 count = result.getInt(1);
24             }
25         } catch (SQLException e) {
26             e.printStackTrace();
27         }
28         return count;
29     }
30
31     @NotNull
32     public final int[] ids(final @NotNull Connection connection, final @NotNull String
tableName) {
33         try (final var statement = connection.createStatement()) {
34             final int count = count(connection, tableName);
35
36             final var ids = new int[count];
37             statement.executeQuery("SELECT id FROM " + tableName);
38
39             final var result = statement.getResultSet();
40             if (result != null) {
41                 int i = 0;
42                 while (result.next()) {
43                     ids[i++] = result.getInt(1);
44                 }
45             }
46             return ids;
47         } catch (SQLException e) {
48             e.printStackTrace();
49         }

```

```

50         return new int [0];
51     }
52
53     private static final class Holder {
54         private static final StorageDAO INSTANCE = new StorageDAO();
55     }
56
57 }

```

Листинг 26: StorageDAO.java

```

1  package com.lamtev.movie_service.datagen.generator.subscription;
2
3
4  import com.lamtev.movie_service.datagen.generator.StorageDAO;
5  import gnu.trove.set.TIntSet;
6  import org.jetbrains.annotations.NotNull;
7
8  import java.sql.Connection;
9  import java.sql.SQLException;
10
11 public final class SubscriptionTableDAO {
12
13     private SubscriptionTableDAO() {
14     }
15
16     public static SubscriptionTableDAO instance() {
17         return Holder.INSTANCE;
18     }
19
20     @NotNull
21     public int [] [] idsNMoviesOrMSeasonsContainingInIds(final @NotNull Connection connection ,
22     final @NotNull int [] [] durationPriceNMoviesMSeasons , final @NotNull TIntSet idsSet) {
23         try (final var statement = connection.createStatement()) {
24             int count = StorageDAO.instance().count(connection , "subscription");
25             final var idsNMoviesOrMSeasons = new int [3][count];
26             statement.executeQuery("SELECT id , (expires - started), payment FROM subscription
27 GROUP BY id");
28             final var result = statement.getResultSet();
29             int i = 0;
30             if (result != null) {
31                 while (result.next()) {
32                     int id = result.getInt(1);
33                     if (idsSet.contains(id)) {
34                         idsNMoviesOrMSeasons[0][i] = id;
35                         final var nm = nMoviesMSeasons(result.getInt(2), result.getInt(3),
36 durationPriceNMoviesMSeasons);
37                         idsNMoviesOrMSeasons[1][i] = nm[0];
38                         idsNMoviesOrMSeasons[2][i] = nm[1];
39                         i++;
40                     }
41                 }
42             }
43             return idsNMoviesOrMSeasons;
44         } catch (SQLException e) {
45             e.printStackTrace();
46         }
47         return new int [0][0];
48     }
49
50     @NotNull
51     private int [] nMoviesMSeasons(int duration , int payment , final @NotNull int [] []
52 durationPriceNMoviesMSeasons) {
53         int n = 0;
54         int m = 0;
55         for (int [] durationPriceNMoviesMSeason : durationPriceNMoviesMSeasons) {
56             if (durationPriceNMoviesMSeason[0] == duration && durationPriceNMoviesMSeason[1]
57 == payment) {
58                 n = durationPriceNMoviesMSeason[2];
59                 m = durationPriceNMoviesMSeason[3];
60                 break;
61             }
62         }
63         return new int [] {n, m};
64     }
65 }

```

```

60     }
61
62     private static final class Holder {
63         private static final SubscriptionTableDAO INSTANCE = new SubscriptionTableDAO();
64     }
65
66 }

```

ЛИСТИНГ 27: SubscriptionTableDAO.java

```

1  package com.lamtev.movie_service.datagen.generator;
2
3  import com.github.javafaker.Faker;
4  import gnu.trove.set.TIntSet;
5  import gnu.trove.set.hash.TIntHashSet;
6  import org.jetbrains.annotations.NotNull;
7
8  import java.sql.Date;
9  import java.sql.SQLException;
10 import java.sql.Statement;
11 import java.util.Random;
12 import java.util.concurrent.TimeUnit;
13
14 public final class Utils {
15
16     @NotNull
17     private final Random random;
18     @NotNull
19     private final Faker faker;
20
21
22     public Utils(@NotNull Random random, @NotNull Faker faker) {
23         this.random = random;
24         this.faker = faker;
25     }
26
27     public static void split(final @NotNull int[][] subscriptionIdsNMoviesMSeasons, int
moviesPercentage, final @NotNull int[][] subscriptionIdsNMovies, final @NotNull int[][]
subscriptionIdsMSeasons) {
28         int moviesIdx = 0;
29         int seasonsIdx = 0;
30         int moviesLength = (int) Math.ceil((double) subscriptionIdsNMoviesMSeasons[0].length /
100) * moviesPercentage;
31         int seasonsLength = subscriptionIdsNMoviesMSeasons[0].length - moviesLength;
32         for (int i = 0; i < 2; ++i) {
33             subscriptionIdsNMovies[i] = new int[moviesLength];
34             subscriptionIdsMSeasons[i] = new int[seasonsLength];
35         }
36         for (int i = 0; i < subscriptionIdsNMoviesMSeasons[0].length; ++i) {
37             if (i % 100 < moviesPercentage) {
38                 subscriptionIdsNMovies[0][moviesIdx] = subscriptionIdsNMoviesMSeasons[0][i];
39                 subscriptionIdsNMovies[1][moviesIdx] = subscriptionIdsNMoviesMSeasons[1][i];
40                 moviesIdx++;
41             } else {
42                 subscriptionIdsMSeasons[0][seasonsIdx] = subscriptionIdsNMoviesMSeasons[0][i];
43                 subscriptionIdsMSeasons[1][seasonsIdx] = subscriptionIdsNMoviesMSeasons[2][i];
44                 seasonsIdx++;
45             }
46         }
47     }
48
49     @NotNull
50     public int[] getIdsOfRowsInsertedWith(final @NotNull Statement statement, int ofLength) {
51         final var keys = new int[ofLength];
52         int i = 0;
53         try (final var generatedKeys = statement.getGeneratedKeys()) {
54             while (generatedKeys.next()) {
55                 keys[i++] = generatedKeys.getInt(1);
56             }
57         } catch (SQLException e) {
58             e.printStackTrace();
59         }
60
61         return keys;
62     }

```

```

63
64 @NotNull
65 public TIntSet getIdsOfRowsInsertedWith(final @NotNull Statement statement) {
66     final var keys = new TIntHashSet();
67     try (final var generatedKeys = statement.getGeneratedKeys()) {
68         while (generatedKeys.next()) {
69             keys.add(generatedKeys.getInt(1));
70         }
71     } catch (SQLException e) {
72         e.printStackTrace();
73     }
74
75     return keys;
76 }
77
78 @NotNull
79 public Date randomDate(int maxYearsAgo) {
80     return new Date(faker.date().past(365 * maxYearsAgo, TimeUnit.DAYS).getTime());
81 }
82
83 public float randomRating() {
84     return 5.0f + random.nextFloat() * (10.0f - 5.0f);
85 }
86
87 @NotNull
88 public int[] nUniqueRandomInts(int n, int bound) {
89     final var ints = new TIntHashSet(n);
90     while (ints.size() != n) {
91         ints.add(random.nextInt(bound));
92     }
93
94     return ints.toArray();
95 }
96
97 }

```

Листинг 28: Utils.java