FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Generátor testovacích dat
Student:	Karel Ševčík
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je analyzovat požadavky, navrhnout a implementovat aplikaci, která umožní snadno připravit testovací a inicializační data při vývoji nových aplikací.
Testovací data bude možné generovat zcela automaticky, na základě informace o typu osobního údaje (jméno, příjmení, atd) nebo je bude možné vytvářet manuálně prostřednictvím grafického rozhraní.
Aplikace musí nabízet API umožňující spouštět inicializaci databáze v rámci procesů kontinuální integrace (CI).
Pro generování jednotlivých údajů použijte existující nástroj Winch, který již obsahuje celou řadu potřebných slovníků a funkcí.
Implementaci proveďte v jazyce Groovy tak, aby byla snadno rozšiřitelná na další databázové systémy.
V rámci bakalářské práce proveďte implementaci pro databáze Oracle a MSSQL. Implementované řešení otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 25. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Generátor testovacích dat

Karel Ševčík

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

27. června 2021

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Jiřímu Mlejnkovi za cenné rady a vstřícnost při konzultacích. Také bych chtěl poděkovat rodině za jejich podporu po celou dobu studia a při psaní práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 27. června 2021 . . . . . .

Odkaz na tuto práci

Ševčík, Karel. *Generátor testovacích dat.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Práce se věnuje vývoji aplikace pro generování testovacích dat. Cílem práce je navrhnout a implementovat prototyp, který bude podporovat generování dat do databází Oracle a SQL Server a bude v budoucnosti rozšiřitelný o podporu dalších databází. Integrováním s nástrojem GEM Winch aplikace umožní pomocí jeho rozsáhlých slovníků generovat smysluplné údaje.

Klíčová slova generování dat, testovací data, testování aplikací, Winch, Groovy, SQL, Oracle, SQL Server

Abstract

This thesis is devoted to the development of an application for generating test data. The aim of this thesis is to design and implement a prototype that will support generating data for Oracle and SQL Server databases and will be extendable to support other databases in the future. By integrating with GEM Winch, the application will be able to generate meaningful data using its extensive dictionaries.

Keywords data generation, test data, application testing, Winch, Groovy, SQL, Oracle, SQL Server

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Tato bakalářská práce se zabývá tvorbou aplikace pro automatické generování testovacích dat. Testování je podstatnou součástí vývoje softwaru. Jeho cílem je pomoci vývojářům co nejdříve odhalit chyby v jejich aplikacích. V dnešní době většina softwaru využívá k uchování a správě svých dat nějaký databázový systém. Pro jeho otestování, je nejprve nutné nějaká testovací data vytvořit. Ruční tvorba smysluplných testovacích dat je časově velmi náročná, proto je užitečné zamyslet se nad její automatizací.

První kapitola popisuje cíle této bakalářské práce, které vychází ze zadání. Druhá kapitola se zabývá rešerší již existujících nástrojů pro generování dat. Na základě srovnání pak specifikuje požadavky na aplikaci vyvíjenou v rámci této práce. Třetí kapitola popisuje samotný návrh aplikace. Čtvrtá kapitola seznamuje s konečnou implementací. Poslední kapitola krátce shrnuje otestování aplikace. V příloze B je sepsána krátká uživatelská příručka.

Kapitola 1

Cíle práce

Hlavním cílem bakalářské práce je vytvořit prototyp aplikace, která umožní snadno připravit testovací data pro vývoj nových aplikací. Testovací data bude možné generovat automaticky, na základě informace o typu osobního údaje (například jméno, příjmení, rodné číslo) nebo je bude možné zadávat manuálně prostřednictvím grafického rozhraní. Aplikace bude podporovat databázové systémy Oracle a Microsoft SQL, ale v budoucnu ji bude možné rozšířit o podporu dalších systémů. Aplikace bude pro generování jednotlivých údajů využívat existující nástroj GEM Winch.

Dílčím cílem, který je nezbytnou součástí práce, je analýza již existujících aplikací, zabývajících se problémem generování testovacích dat. Na základě analýzy poté sestavit seznam požadavků, které aplikace vytvořena v rámci této práce musí splňovat.

Kapitola 2

Rešerše a analýza požadavků

Kapitola se zabývá srovnáním již existujících aplikací a analýzou požadavků na základě zadání práce.

2.1 Srovnání existujících aplikací

Aplikací zabývajících se problematikou generování testovacích dat existuje mnoho. Ke srovnání byly vybrány čtyři, které data generují do databází. Srovnání je zaměřeno na počet podporovaných DBMS, počet různých typů generátorů, možnosti exportu generovaných dat a možnosti automatizace generování. Každá z nich je v této sekci krátce popsána. Jejich přehled je uveden v tabulce 2.1.

Aplikace	Podporované databáze	Počet generátorů	Export dat	Automatizace
Redgate SQL Data Generator	MSSQL Server	60+	Ne	Ano, CLI
Devart Data Generator	MSSQL Server, Oracle, MySQL	220+	Ano, SQL skript	Ano, CLI
EMS Data Generator	MSSQL Server, MySQL, Oracle, DB2, PostgreSQL, InterBase	6+	Ano, SQL	Ano. CLI
SB Data Generator	Oracle, MSSQL Server, MySQL, PostgreSQL	30+	Ano, CSV	Ne

Tabulka 2.1: Přehled existujících aplikací

2.1.1 Redgate SQL Data Generator

Redgate SQL Data Generator [1] je aplikace vyvíjena firmou Redgate Software, která umožňuje generovat testovací data a vkládat je do databázového systému Microsoft SQL Server. Aplikace je postavena na platformě .NET verze .NET Framework 4.7 a funguje tedy pouze na operačním systému Windows. Generování dat pomocí tohoto nástroje lze velmi rozsáhle nakonfigurovat. Podporuje velké množství typů generovaných dat. Pokročilým uživatelům umožňuje v jazyce Python vytvořit vlastní generátory pro nové typy údajů. Kromě konfigurace a generování pomocí grafického rozhraní umožňuje generování spustit zcela automaticky podle uložené konfigurace využitím příkazové řádky. Aplikace také nabízí integraci do nástroje SQL Server Management Studio.

Největší nevýhodou SQL Data Generatoru je podpora pouze jednoho databázového systému. Nástroj je také zaměřen na anglicky mluvící uživatele a bez vytvoření vlastních generátorů není schopen generovat údaje lokalizované pro Česko.

2.1.2 Devart Data Generator

Devart Data Generator je jedním z nástrojů sady dbForge od firmy Devart [2]. Je postaven na technologii .NET Framework, která podporuje pouze operační systém Windows. Nástroj má tři různé verze, každá z nich je určena pro jiný databázový systém. Devart Data Generator umožňuje generovat data pomocí velkého množství generátorů. Nástroj také umožňuje vytvořit vlastní generátory pomocí XML souborů. Vygenerovaná data je možno exportovat jako SQL skript. Nástroj také umožňuje generování spouštět z příkazové řádky.

Nástroj sice podporuje tři různé databázové systémy, ale tato podpora je pro každý z nich realizována pod jinou aplikací a tedy i jinou licencí.

2.1.3 EMS Data Generator

EMS Data Generator je nástroj vyvíjen firmou EMS Software Development [3]. Podporuje velké množství databázových systémů. Na rozdíl od ostatních nástrojů má pro každou skupinu datových typů v DBMS vždy jen jeden generátor. Generátory lze nastavit do několika různých módů: náhodné generování, inkrementální generování, náhodný výběr ze seznamu, SQL dotaz a výběr z jiného sloupce. Dalším rozdílem od jiných nástrojů je způsob konfigurace. Grafické rozhraní je vytvořeno jako průvodce, konfigurace tedy probíhá v krocích. Vygenerovaná data lze uložit do SQL skriptu. Konfiguraci lze uložit do souboru a ten využít pro generování z příkazové řádky.

Stejně jako u nástroje od firmy Devart je pro podporu daného databázové systému potřeba zakoupit odpovídající licenci.

2.1.4 SB Data Generator

SB Data Generator [4] je aplikace od firmy Softbuilder. Aplikace je určena pro operační systém Windows. Kromě podpory velkého množství databázových systémů aplikace poskytuje přehled nad strukturou databáze formou diagramu databázového schématu. Aplikace obsahuje řadu předem definovaných generátorů a umožňuje uživateli přidat vlastní. Vygenerovaná data lze před vložením do databáze prohlížet, upravit a exportovat do souboru ve formátu CSV.

Oproti ostatním aplikacím SB Data Generator neposkytuje způsob, jak spustit proces generování pomocí příkazové řádky. Není tedy vhodnou volbou pokud generování dat má být součástí automatizovaného procesu.

2.2 Požadavky aplikace

Na základě analýzy existujících aplikací a konzultace s vedoucím práce byl vytvořen následující seznam požadavků. V sekci 2.2.2 je popsána míra splnění těchto požadavků u srovnávaných aplikací.

2.2.1 Seznam požadavků aplikace

1. **Generování testovacích dat**

   Aplikace musí vygenerovat požadovaný počet záznamů odpovídajících struktuře databáze. Data vygenerovaná aplikací musí zachovat referenční integritu.

2. **Konfigurace generování dat**

   Aplikace musí uživateli umožnit nakonfigurovat počet záznamů generovaných pro danou tabulku a typ dat generovaných pro jednotlivé sloupce tabulky. Aplikace dále musí uživateli umožnit tuto konfiguraci uložit pro pozdější použití.

3. **Proces generování dat**

   Aplikace musí generovat data podle uživatelem vytvořené konfigurace a umožnit uživateli k vygenerovaným datům přidat vlastní záznamy. Aplikace musí být schopná proces generování spouštět automaticky jako součást procesu CI.

4. **Podpora různých typů generovaných údajů**

   Aplikace musí být schopná generovat následující typy údajů:

   - jméno,
   - příjmení,
   - emailová adresa,

- rodné číslo,
- pohlaví,
- ulice,
- město,
- okres,
- datum,
- řetězec,
- číslo.

5. **Ukládání vygenerovaných dat**

   Aplikace musí mít možnost vygenerovaná data vložit přímo do databáze. Aplikace také musí umožnit uživateli vygenerovaná data vyexportovat ve formě SQL skriptu.

6. **Podpora DBMS**

   Uživateli musí být umožněno nastavit připojení k databázi. Aplikace musí podporovat databázové systémy Oracle a Microsoft SQL Server.

7. **Rozšiřitelnost**

   Implementace musí být rozšiřitelná o podporu dalších DBMS. Implementace musí být rozšiřitelná o podporu nových typů generovaných údajů.

2.2.2 Přehled splnění požadavků srovnávaných aplikací

V tabulce 2.2 je vidět, že žádná ze srovnávaných aplikací nesplňuje veškeré stanovené požadavky. Především tyto nástroje nepodporují českou lokalizaci a neumožňují generovat specifická data typu jako je např. rodné číslo, které se také využívá jako číslo pojištěnce pro sociální a zdravotní pojištění.

	Číslo požadavku					
Aplikace	1	2	3	4	5	6
Redgate SQL Data Generator	✓	✓	✓	✗	✗	✗
Devart Data Generator	✓	✓	✓	✗	✓	✓
EMS Data Generator	✓	✓	✓	✗	✓	✓
SB Data Generator	✓	✓	✗	✗	✓	✓

Tabulka 2.2: Přehled splnění požadavků u existujících aplikací

Kapitola 3

Návrh aplikace

V této kapitole je popsán návrh aplikace, který splňuje požadavky stanovené v předchozí kapitole.

3.1 Popis nástroje GEM Winch

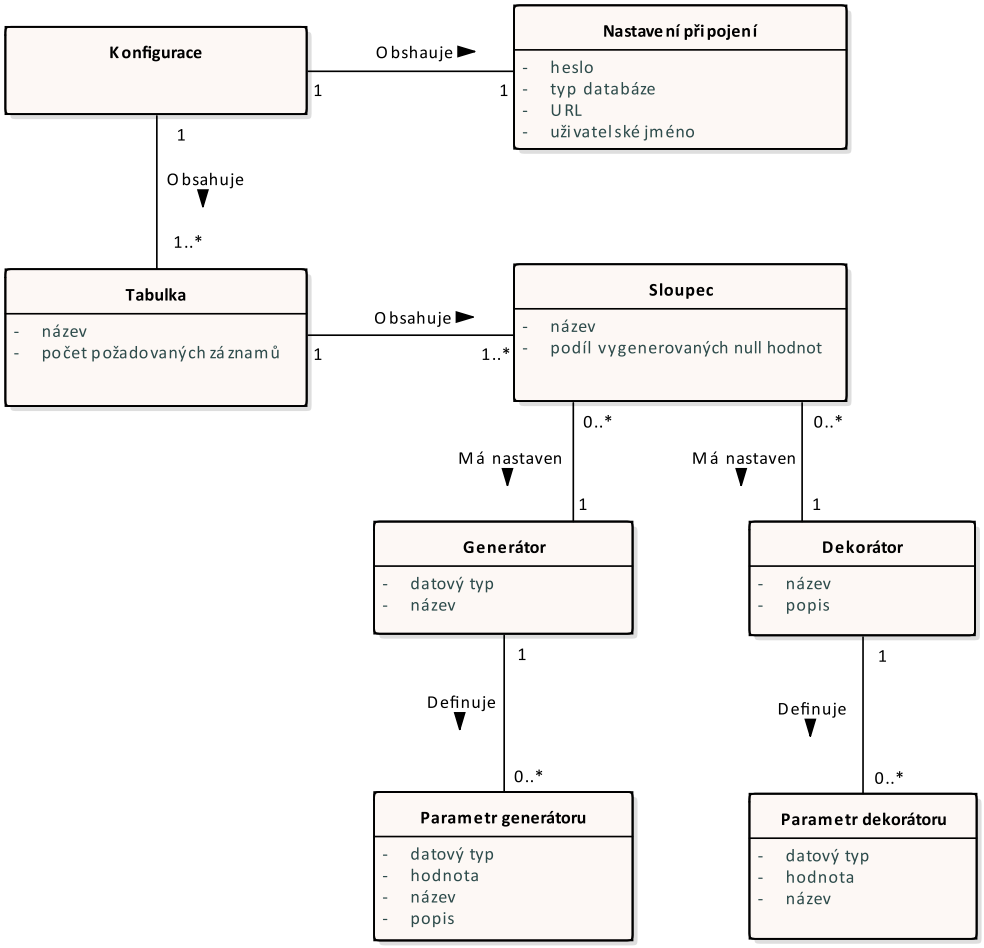
Aplikace bude vyvíjena jako podprojekt nástroje GEM Winch od společnosti GEM System a.s. Nástroj GEM Winch slouží k anonymizaci osobních a citlivých údajů v relačních databázích. Je tvořen dvěma komponentami: *GEM Winch Add-In* a *GEM Winch Actor*. GEM Winch Add-In je rozšíření pro nástroj Enterprise Architect. Add-In slouží k načtení struktury datových objektů pro anonymizaci a řez dat a nastavení parametrů anonymizace. GEM Winch Actor je program spustitelný z příkazové řádky. Actor na základě konfigurace vytvořené v EA provádí jednotlivé kroky procesu anonymizace. [5]

K realizaci této práce bude využívána funkcionalita z komponenty GEM Winch Actor. Actor umožňuje z EA načíst databázové schéma s informacemi potřebnými pro funkci této aplikace. Actor za účelem anonymizace také obsahuje řadu slovníků, které obsahují velké množství užitečných údajů. Rozhraní pro práci s těmito slovníky je dáno třídou `AbstractDictionary`, nacházející se v balíčku `com.gem.winch.dictionary`. Toto rozhraní bude rozšířeno o funkčnost získání náhodného záznamu.

GEM Winch Actor pro anonymizaci jednotlivých údajů používá tzv. „anonymizační funkce“. Každá anonymizační funkce realizuje anonymizace jednoho typu údaje. Tyto funkce jsou reprezentovány třídami, které jsou potomky abstraktní třídy `AnonymizationFunction`. Na rozdíl od anonymizace, která je implementována jako databázová funkce v jazyce SQL, bude generování probíhat přímo v aplikaci na straně klienta. Toto umožní data generovat i bez připojení k databázovému systému a také usnadní možné budoucí rozšíření pro generování dat mimo databáze.

3.2 Objekty doménového modelu

V této sekci jsou krátce popsány některé objekty doménového modelu aplikace. Tento model je vidět na obrázku 3.1.



Obrázek 3.1: Doménový model aplikace

3.2.1 Konfigurace

Konfigurace je datová struktura obsahující informace potřebné ke generování dat. Uchovává nastavení připojení k DB, seznam tabulek, počet požadovaných záznamů pro každou tabulku a sloupce jednotlivých tabulek. Pro každý sloupec má informaci, zda se jedná o primární klíč, zvolený generátor, dekorátor a všechny jejich parametry.

3.2.2 Generátor

Generátory představují základní jádro aplikace. Každý generátor je odpovědný za generování jednoho typu údaje. Generátory mohou na vstupu přijímat parametry ovliňující generovaná data. Například pro generátor křestních jmen bude parametrem pohlaví.

3.2.3 Parametr generátoru

Parametry generátoru ovlivňují data, která generuje. Tyto parametry mohou být dvou druhů: konstanta, nebo sloupec.

3.2.4 Dekorátor

Dekorátor upravuje generátorem vygenerovaná data. Jako příklad lze uvést, že všechna velká písmena v řetězci převede na malá.

3.2.5 Parametr dekorátoru

Parametry dekorátoru ovlivňují způsob, kterým vygenerovaná data upravuje.

3.3 Nastavení generování

Uživateli bude umožněno pro každou tabulku nastavit počet záznamů k vygenerování a ručně zadat data, která se přidají ke generovaným. Dále pro každý sloupec v dané tabulce bude možno nakonfigurovat typ generovaného údaje zvolením příslušného generátoru a jeho parametrů.

3.3.1 GUI

K umožnění tvorby a úpravy konfigurací bude vytvořeno jednoduché grafické rozhraní. Pro vytvoření tohoto rozhraní se jako technologie nabízí JavaFX a Swing. JavaFX je moderní knihovna, která umožňuje vytvářet desktopové aplikace běžící na všech populárních operačních systémech. Původně vznikla jako nástupce technologie Swing [6]. Swing je knihovna sloužící k tvorbě grafických rozhraní. Jedná se o starší technologii.

Pro vývoj GUI aplikace byl zvolen Swing. Swing je součástí standardní knihovny Java a k práci s ním je v jazyce Groovy navíc k dispozici třída `SwingBuilder`. Při použití Swing tedy není nutno nabírat další externí závislosti.

3.3.2 Persistence konfigurace

Pro splnění požadavku na uložení konfigurace je potřeba navrhnout způsob, jakým vytvořenou konfiguraci ukládat. Prvním možným způsobem je seriali-

zace konfigurace do nějakého specifického datového formátu a jeho následné uložení do souboru na disku. Další možný způsob je konfiguraci uložit do databáze, ať už se jedná o SQLite databázi distribuovanou s aplikací, nebo o databázi na vzdáleném serveru.

Z těchto způsobů byla zvolena serializace do souboru na disku. Při zvolení vhodného formátu jsou její hlavní výhody čitelnost a dobrá podpora různých verzovacích systémů. K dispozici existuje celá řada různých formátu, z nich byly k bližšímu rozboru zvoleny formáty *YAML*, *XML* a *JSON*. Pro ukázku každého z těchto formátu je použit příklad konfigurace sloupce s názvem *jmeno*, do kterého mají být generovány křestní jména. Tento generátor má jako parametr pohlaví osoby, pro kterou jméno generujeme. V tomto příkladu je hodnotou parametru konstanta *M*.

3.3.2.1 YAML

YAML je formát pro serializaci strukturovaných dat. Jeho název je rekurzivní zkratka *YAML Ain't Markup Language*. Jako svůj hlavní cíl si klade být čitelný člověkem, ne pouze strojem. Z názvu plyne, že pro strukturování dat YAML nepoužívá značky. Struktura dat v YAML je určena pomocí odražení mezerami. [7]

```yaml
nazevSloupce: jmeno
generator: Jméno
parametryGeneratoru:
  - nazevParametru: Pohlaví
    jeKonstatni: true
    hodnotaParametru: M
```

Výpis kódu 3.1: Ukázka formátu YAML

3.3.2.2 XML

XML je rozšiřitelný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. XML je jednoduchý a rozšiřitelný formát určený ke sdílení dat. Mezi cíle XML patří jeho snadná použitelnost přes internet, podpora široké škály aplikací a čitelnost pro člověka [8]. Stejně jako JSON a YAML se používá pro serializaci dat.

3.3.2.3 JSON

JSON je formát pro výměnu dat založen na podmnožině programovacího jazyka Javascript [9]. Jedná se o člověkem čitelný formát. JSON, stejně jako YAML, k oddělení hodnot a definování struktury dat nepoužívá značky. Tím docílí menšího výsledného souboru než formát XML.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<nazevSloupce>jmeno</nazevSloupce>
<generator>Jméno</generator>
<parametryGeneratoru>
    <nazevParametru>Pohlaví</nazevParametru>
    <jeKonstatni>true</jeKonstatni>
    <hodnotaParametru>M</hodnotaParametru>
</parametryGeneratoru>
```

Výpis kódu 3.2: Ukázka formátu XML

```json
{
  "nazevSloupce": "jmeno",
  "generator": "Jméno",
  "parametryGeneratoru": [
    {
      "nazevParametru": "Pohlaví",
      "jeKonstatni": true,
      "hodnotaParametru": "M"
    }
  ]
}
```

Výpis kódu 3.3: Ukázka formátu JSON

3.3.2.4 Srovnání formátů

Z ukázek 3.1, 3.2 a 3.3 je vidět, že všechny tři formáty by bylo možné použít pro ukládání konfigurace.

Využití formátu YAML bylo vyloučeno hlavně z důvodu, že formát nijak nespecifikuje ukončení hodnoty. Tím může nastat problém, kdy se nedopsaný nebo oříznutý YAML soubor považuje za validní.

Nakonec byl zvolen formát JSON, hlavně z důvodu že nástroj Winch již JSON používá pro komunikaci mezi svými moduly a knihovna pro práci s ním je již k dispozici.

3.4 Generování dat

Při procesu generování se podle konfigurace zvolenému generátoru předají nastavené parametry. Pokud je parametr typu sloupec, generátoru je předána hodnota z daného sloupce právě generovaného záznamu. Toto umožňuje generovat údaje na základě hodnot jiných (např. rodné číslo podle data narození). Kromě těchto parametrů je každému generátoru předána informace jaký podíl

null hodnot má generovat. K vlastnímu generování údajů budou, kde je to vhodné, využity slovníky z nástroje Winch.

Pro zachování entitní a referenční integrity generovaných dat budou existovat speciální generátory. Generátor pro cizí klíče, který na rozdíl od ostatních generátorů má jako svůj parametr tabulku, kterou má referencovat. Generátor pro primární klíče oproti ostatním generuje své hodnoty inkrementálně. Aplikace prozatím podporuje pouze celočíselné primární klíče. PK složené z více sloupců také nejsou podporovány.

Po vygenerování všech záznamů jsou podle konfigurace pro jednotlivé sloupce na jejich data aplikovány zvolené dekorátory.

3.4.1 Seznam generátorů

Generátor jmen na vstupu dostane jako parametr pohlaví. Jména dle něj náhodně vybírá ze slovníků dostupných z nástroje Winch.

Generátor příjmení stejně jako generátor jmen je jeho jediným parametrem pohlaví. Příjmení také vybírá z dostupných slovníků.

Generátor emailů je bez parametrů, generuje náhodné řetězce obsahující znak `@` a končící na `.cz`.

Generátor rodných čísel má tři parametry: datum narození, pohlaví a příznak určující, zda rodné číslo obsahuje lomítko.

Generátor pohlaví nemá parametry, pouze náhodně zvolí mezi mužským a ženským. Vrací hodnoty „m“ pro muže a „f“ pro ženy.

Generátor ulic je bez parametrů. Ulici náhodně vybere ze slovníku.

Generátor měst nemá parametry, města vybírá ze slovníku.

Generátor okresů je bez parametrů, okresy vybírá náhodně ze slovníku.

Generátor datumů jako parametry dostane minimální a maximální hodnoty, vygenerované datum bude ležet mezi nimi.

Generátor řetězců na vstupu dostane požadovanou délku řetězce. Generovaný řetězec je složen z náhodných znaků.

Generátor čísel má dva parametry: minimum a maximum, vygeneruje číslo i tak, aby platilo $minimum \leq i < maximum$.

Generátor primárních klíčů jako parametr dostane číslo, od kterého má začít generovat.

Generátor cizích klíčů má jeden parametr – rodičovskou tabulku. Vrací náhodně vybrané PK záznamů z této tabulky.

Generátor null hodnot nemá parametry, vrací pouze hodnoty *null*. V aplikaci bude sloužit jako výchozí generátor pro všechny sloupce kromě sloupců PK.

3.5 Ukládání vygenerovaných dat

K uložení vygenerovaných dat bude sestaven SQL skript přizpůsoben danému databázovému systému. Pro zachování referenční intergrity bude nutno při vkládání dat vypnout automatické vyplňování PK. Pokud by se tak neučinilo, aplikací vygenerované PK by byly přečíslované a nemusely by se shodovat s hodnotami v dceřiných tabulkách. Po vložení dat a obnovení automatického vyplňování PK je potřeba nastavit hodnotu příštího PK tak, aby nedocházelo ke kolizím s vygenerovanými daty.

Aplikace bude nabízet dvě možnosti jak vygenerovaná data uložit. Buď sestavený SQL skript uloží do souboru na disk pro pozdější použití, nebo se aplikace připojí k databázi a skript spustí.

Kapitola 4

Implementace

Tato kapitola popisuje strukturu a zvolené postupy konečné implementace aplikace. V podkapitole 4.4 jsou popsány nedostatky a nápady na budoucí rozšíření aplikace.

Pro vývoj aplikace vznikl v hlavním projektu nástroje GEM Winch Actor podprojekt `disl-winch-generator`. V průběhu implementace bylo také zasaženo do podprojektu `disl-winch-connector`. Tento podprojekt realizuje hlavní funkčnosti nástroje GEM Winch Actor a některé z nich bylo nutno pro vývoj aplikace rozšířit.

Aplikace je vyvíjena v jazyce Groovy. Groovy je programovací jazyk, kompatibilní s Java platformou. Stejně jako Java je kompilován do *JVM bytecode*, tím snadno umožňuje pracovat s již existujícími knihovnami v jazyce Java. Groovy oproti Javě podporuje dynamické typování a má volnější syntaxi. [10]

4.1 GUI

K implementaci grafického rozhraní aplikace byla využita technologie Swing. Pro usnadnění práce se Swingem byla použita třída `SwingBuilder`, která je součástí standardního balíčku `groovy.swing`. Třídy realizující GUI jsou v balíčku `com.gem.winch.datagen.gui`.

4.1.1 SwingBuilder

`SwingBuilder` umožňuje vytvářet Swing GUI deklarativním způsobem. Stará se o vytváření komplexních objektů automatickým vytvářením instancí GUI komponent, nastavováním všech jejich potřebných parametrů a sestavovaní hierarchie těchto komponent [10].

Použití třídy `SwingBuilder` vede ke kratšímu a přehlednějšímu kódu. Na ukázkách 4.1 a 4.2 je vidět vytvoření okna s jedním tlačítkem, které po stisknutí do konzole vypíše řetězec `"Hello world!"`.

```
import groovy.swing.SwingBuilder

new SwingBuilder().edt {
  frame(title: 'Frame', size: [250, 75], show: true) {
    button(text:'Click!', actionPerformed: {println "Hello world!"})
  }
}
```

Výpis kódu 4.1: Použití `SwingBuilder`

```
import java.awt.event.*
import javax.swing.*

def frame = new JFrame("Frame")
frame.setSize(250, 75)
def button = new JButton("Click!")
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        println "Hello world!"
    }
})
frame.add(button)
frame.setVisible(true)
```

Výpis kódu 4.2: Swing bez použití `SwingBuilder`

4.1.2 Popis tříd realizujících GUI

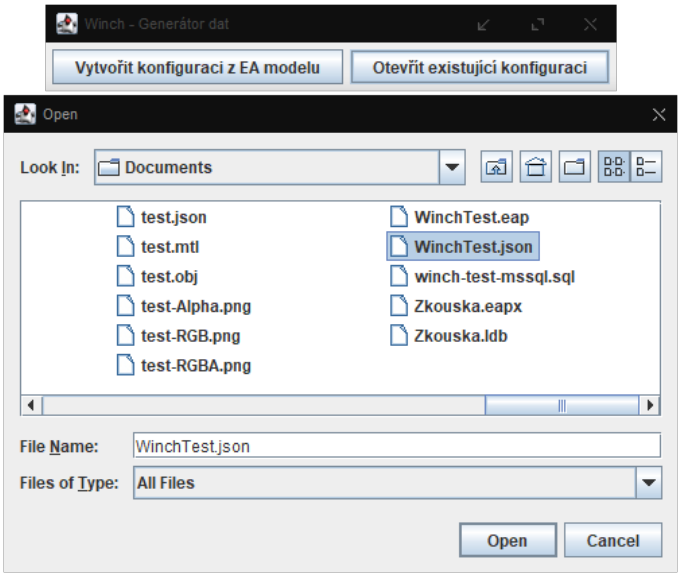
StartWindow

`StartWindow` realizuje okno zobrazované pří spuštění aplikace. Okno uživateli umožňuje buď načíst již existující konfiguraci pro generování, nebo vytvořit novou. Pro vytvoření nové konfigurace je třeba načíst databázové schéma (tabulky, sloupce) z repozitáře vytvořeném v nástroji Enterprise Architect. Pro hledání souborů na disku je využita třída `JFileChooser`. Okno s výběrem souboru s uloženou konfigurací je vidět na obrázku 4.1.

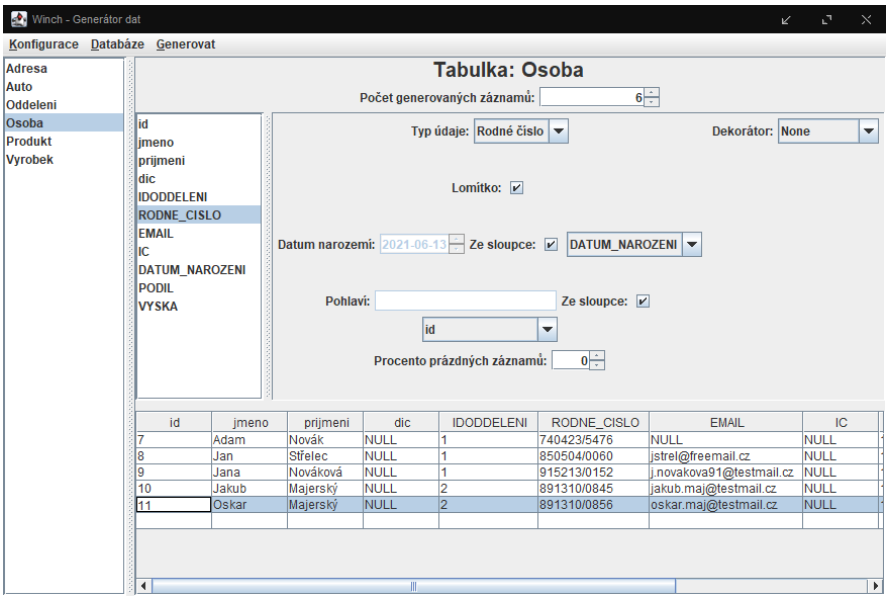
MainWindow

Třída `MainWindow` reprezentuje hlavní okno aplikace (obrázek 4.2). Třída poskytuje metodu `show`, která slouží k sestavení a zobrazení celé hierarchie potřebných komponent. Mezi její přímé zodpovědnosti patří zobrazit seznam tabulek a zobrazit hlavní lištu a zpracovávat události jí vyvolané. Dále pak reagovat na událost vyvolanou zvolením tabulky ze seznamu. Na tuto událost reaguje předáním zvolené tabulky instanci třídy `TablePanel`. Pro zobrazení

seznamu tabulek je využita třída `JList`.



Obrázek 4.1: Okno s výběrem uložené konfigurace



Obrázek 4.2: Hlavní okno – nastavení generování sloupce „RODNE_CISLO“

TablePanel

Třída `TablePanel` je zodpovědná za zobrazování všech informací týkajících se právě zvolené tabulky – název tabulky, počet záznamů k vygenerování, seznam jejích sloupců. Při úpravě některého z těchto údajů změny propisuje do modelu. Zobrazování informací o právě zvoleném sloupci má na starosti třída `ColumnPanel`, jejíž instanci je při vybrání jiného sloupce tento sloupec předán. Při změně zvolené tabulky se tabulka předá instanci `UserDataTablePanel`.

UserDataTablePanel

Třída `UserDataTablePanel` slouží ke zobrazování a úpravě manuálně zadaných dat. Pro tento účel je využita třída `JTable`. Při změně aktuálně zvolené tabulky se instanci `JTable` nastaví datový model na instanci modelu z nově zvolené tabulky.

ColumnPanel

Třída `ColumnPanel` má na starosti zobrazování informací o právě zvoleném sloupci. Uživateli umožňuje nastavit generátor, dekorátor a podíl *null* hodnot pro daný sloupec. Pokud uživatel změní generátor nebo dekorátor, třída předá seznam parametrů nové volby instanci třídy `GeneratorSettingsPanel` resp. `DecoratorSettingsPanel`.

GeneratorSettingsPanel

Třída `GeneratorSettingsPanel` je zodpovědná za zobrazení položek nutných ke konfiguraci zvoleného generátoru. Pro každý parametr podle jeho typu sestaví panel k zadání hodnoty. V tomto panelu se nachází:

- název parametru;
- vstupní pole pro konstantní hodnotu;
- přepínač určující, zda parametr je konstantní;
- pole pro výběr sloupce, ze které se načte hodnota parametru.

Pole pro zadání konstanty je vytvořeno podle typu hodnoty definovaném pro daný parametr. Pro tento účel jsou ve třídě metody `buildXXXXInput`, kde `XXXX` odpovídá danému typu (například `buildIntegerInput` pro celočíselné hodnoty).

DecoratorSettingsPanel

Tato třída obdobně jako `GeneratorSettingsPanel` slouží k zobrazení a nastavení parametrů právě zvoleného dekorátoru. Oproti `GeneratorSettingsPanel`

mohou být hodnotou parametru pouze konstanty, tedy přepínač a pole pro výběr sloupce zde neexistují.

4.2 Konfigurace

K realizaci konfigurace byly vytvořeny tři balíčky:

- `com.gem.winch.datagen.model`, ve kterém jsou třídy datové vrstvy aplikace;
- `com.gem.winch.datagen.configuration`, který obsahuje pomocné třídy pro zjednodušení serializace a deserializace;
- `com.gem.winch.datagen.configuration.persistence` obsahující třídy odpovědné za vlastní serializaci, deserializaci konfigurace a psaní a čtení konfiguračních souborů.

4.2.1 Popis tříd modelu

GenerationModel

Hlavní třída modelu aplikace, slouží jako kořen jeho stromu. Obsahuje instance tříd `TableModel` a instanci třídy `DatabaseConnectionModel`. Poskytuje metody pro sestavení jednotlivých kroků generování dat a SQL skriptu a jejich následné vykonání.

DatabaseConnectionModel

Třída slouží k uchování informace o typu databázového systému a údajů potřebných pro navázání spojení s tímto systémem.

TableModel

Třída udržuje informace o tabulce – název, počet řádků ke generování, seznam sloupců (instance třídy `ColumnModel`) a uživatelem ručně zadaná data. Poskytuje metodu pro nalezení tabulek, na kterých je závislá, pokud některý ze sloupců je cizí klíč. Metoda `getGeneratedData` ze sloupců tabulky přečte jednotlivé vygenerované hodnoty a spojí je do jedné datové struktury.

UserDataModel

Třída udržuje data manuálně zadaná uživatelem. Aby mohla být použita jako datový model pro instanci třídy `JTable`, dědí od třídy `AbstractTableModel` z balíčku `javax.swing.table`. Poskytuje metody pro úpravu a získání hodnot jednotlivých polí tabulky a přidávání nových řádků.

ColumnModel

`ColumnModel` reprezentuje sloupec, udržuje jeho název, zda je PK, podíl *null* hodnot, zvolený dekorátor (instance `DecoratorModel`) a generátor (instance `GeneratorModel`) a vygenerovaná data. Poskytuje metody pro vygenerování dat a pro přistup k těmto datům.

GeneratorModel

Třída reprezentuje generátor. Udržuje informace o třídě zvoleného generátoru a seznam jeho parametrů (instance `GeneratorParameterModel`). Při změně třídy generátoru využitím reflexe načte z anotace generátoru seznam jeho parametrů a svůj seznam aktualizuje. Také poskytuje metodu k získání sloupců, které jsou potřebné pro vygenerování údaje. Sloupec je potřebný pro generování, pokud nějaký z parametrů není konstantní a svou hodnotu bere z tohoto sloupce.

GeneratorParameterModel

Třída reprezentuje parametr generátoru. Udržuje jeho název, datový typ (výčtový typ `GeneratorParamType`), informaci zda se jedná o konstantu, hodnotu této konstanty, sloupec s hodnotou parametru (pokud není konstantní) a tabulku (pokud je typ parametru `GeneratorParamType.TABLE`). Poskytuje metodu `getParameter`, která pokud je hodnota nastavená jako konstantní vrátí tuto konstantu, nebo pokud je typ `GeneratorParamType.TABLE` vrátí seznam PK této tabulky, jinak vrátí hodnotu načtenou ze určeného sloupce.

DecoratorModel

`DecoratorModel` udržuje informace o aktuálně zvoleném dekorátoru – jeho třídu a seznam jeho parametrů (instance třídy `DecoratorParameterModel`). Při změně třídy pomocí reflexe načte seznam parametrů pro novou třídu z anotace a svůj seznam parametrů aktualizuje.

DecoratorParameterModel

Třída `DecoratorParameterModel` reprezentuje parametr dekorátoru. Uchovává informace o jeho názvu, datovému typu a hodnotě.

4.2.2 Popis tříd realizujících serializaci

Balíček com.gem.winch.datagen.configuration

V tomto balíčku se nacházejí pomocné třídy usnadňující serializaci konfigurace. Serializovat přímo třídy modelu není možné, protože obsahují cyklické reference. Při serializaci těchto tříd by došlo k nekonečné smyčce a eventuálnímu

pádu aplikace. Třídy v tomto balíčku udržují stejné informace jako třídy modelu, jen mají odstraněny tyto cykly. U tříd `GeneratorConfiguration` a `DecoratorConfiguration` je ukládán pouze plný název třídy daného generátoru resp. dekorátoru, protože tato informace je dostatečná k její identifikaci a ukládání řetězce knihovna GSON již podporuje.

ConfigurationReaderInterface a ConfigurationWriterInterface

Definují rozhraní pro třídy sloužící ke čtení a zapisování konfigurace. Jejich existence umožní v budoucnu přidat možnost konfiguraci ukládat nejen do souborů na disku.

FileConfigurationReader a FileConfigurationWriter

Abstraktní třídy implementující výše zmíněné rozhraní. Slouží jako předci pro třídy, které pro ukládání a načítání konfigurace používají soubory na disku.

JSONFileConfigurationReader a JSONFileConfigurationWriter

Potomci tříd `FileConfigurationReader` resp. `FileConfigurationWriter` pro ukládání konfigurace používají soubory na disku s obsahem ve formátu JSON. K serializaci a deserializaci je používána knihovna GSON.

4.3 Generování dat a SQL skriptu

Proces generování je spuštěn uživatelem pomocí tlačítka z lišty hlavního okna, GUI na tuto událost reaguje voláním metody `generateSql` instance třídy `GenerationModel`. Metoda nejprve zavolá metodu pro sestavení jednotlivých kroků generování dat, popsaných v sekci 4.3.1.1. Poté s využitím tovární třídy `SqlGeneratorFactory` získá potomka třídy `AbstractSqlGenerator`, který generuje SQL kód přizpůsobený konkrétnímu databázovému systému. Nakonec sestaví kroky generování SQL skriptu (sekce 4.3.1.2) a vytvoří instanci třídy `GenerationTaskList`, které všechny tyto kroky předá a zavolá její metodu `executeTasks`.

4.3.1 Kroky procesu generování

Proces generování je rozdělen do několika různých kroků. Důvodem k tomuto rozdělení je určení správného pořadí vykonání jednotlivých kroků. Při generování dat některých sloupců je totiž nutno již mít k dispozici data jiných sloupců (například pro generování sloupců s cizím klíčem je nutné mít k dispozici primární klíč rodičovské tabulky). K reprezentaci těchto kroků slouží třída `AbstractGenerationTask` a její potomci.

Třída `AbstractGenerationTask` společnou funkcionalitu všech kroků generování. U každé instance se udržuje seznam kroků, které na kroku reprezentovaným instancí závisí a je tedy nutné tento krok vykonat dříve než ty závislé. Třída dále obsahuje metody k přidání závislostí.

4.3.1.1 Kroky generování dat

Generování dat je rozděleno na kroky reprezentované následujícími třídami.

`TablePKGenerationTask` reprezentuje krok generování primárního klíče dané tabulky. Při vykonání nalezne v tabulce sloupec PK a zavolá jeho metodu `generateData`.

`FKColumnGenerationTask` reprezentuje krok generování sloupce s cizím klíčem. Při vykonání pouze volá metodu `generateDate` daného sloupce.

`ColumnGenerationTask` reprezentuje krok generování ostatních sloupců (nejedná se o PK ani FK). Stejně jako `FKColumnGenerationTask` při vykonání volá metodu `generateDate`.

`IncludeUserDataTask` reprezentuje krok připojení uživatelem manuálně zadaných dat k datům vygenerovaných. Při vykonání se z modelu dané tabulky tyto data načtou a přidají se k vygenerovaným datům do modelů sloupců tabulky.

V průběhu sestavování kroků generování dat se jejich závislosti nastavují následujícím způsobem:

1. Každému kroku generování FK se jako závislost přidají všechny kroky generování PK. Zde by stačilo jako závislost kroku generování FK nastavit pouze generování PK rodičovské tabulky, ale toto nastavení výsledek generování nijak neovlivní a není pro něj nutno hledat konkrétní krok generování PK.

2. Každému kroku generování sloupců, které nejsou PK ani FK, se z modelu sloupce nalezne seznam sloupců jejichž hodnoty jsou parametrem generátoru tohoto sloupce (metoda `getRequiredColumns`). Pro každý sloupec v tomto seznamu se nalezne příslušný krok jeho generování a přidá se jako závislost právě zpracovávanému kroku.

3. Nakonec se každému kroku připojení uživatelem zadaných dat nastaví jako závislosti všechny ostatní kroky generování sloupců. Tím se docílí toho, že se kroky týkající se těchto dat vykonají v rámci generování dat jako poslední.

4.3.1.2 Kroky generování SQL skriptu

Generování SQL skriptu je rozděleno na několik kroků reprezentovaných následujícími třídami.

`ClearTablesSqlGenerationTask` reprezentuje krok generování SQL skriptu pro odstranění dat dané tabulky. Při vykonání volá metodu `clearTable` instance potomka `AbstractSqlGenerator` odpovídajícího danému databázovému systému.

`DisableChecksSqlGenerationTask` reprezentuje krok generování SQL kódu pro vypnutí kontrolování dat vložených do databáze. To umožňuje vkládat data do DB v jakémkoliv pořadí, bez ohledu na cizí klíče. Při vykonání volá metodu `disableChecksForTable`.

`TableInsertSqlGenerationTask` reprezentuje krok generování SQL kódu pro vložení vygenerovaných dat dané tabulky. Při vykonání jako první zavolá metodu `disablePKAutoIncrement`, která generuje SQL kód pro vypnutí automatického vyplňování PK dané tabulky. Poté z modelu tabulky načte vygenerovaná data a zavolá metodu `insertRows`, která generuje SQL pro vložení dat. Nakonec voláním metod `enablePKAutoIncrement` a `reseedPKValue` vygeneruje SQL pro zapnutí vyplňování PK a nastavení příští hodnoty PK.

`EnableChecksSqlGenerationTask` reprezentuje krok generování SQL kódu pro zapnutí kontrolování dat vkládaných do databáze. Při vykonání volá metodu `enableChecksForTable`.

Při sestavování těchto kroků jsou jejich závislosti nastaveny následovně:

1. Kroku generování SQL pro vypnutí kontrolování dat se jako závislosti nastaví všechny kroky generování dat a krok generování SQL pro odstranění dat.

2. Každému kroku generování SQL pro vkládání dat se jako závislost nastaví krok generující SQL pro vypnutí kontroly vkládaných dat.

3. Nakonec se kroku generování SQL pro zapnutí kontroly vkládaných nastaví jako závislosti všechny kroky pro vkládání dat.

4.3.1.3 Konání jednotlivých kroků

Po sestavení všech kroků jsou tyto přidány do instance třídy `GenerationTaskList`, která realizuje seznam kroků a umožňuje jejich vykonání ve správném pořadí. K určení správného pořadí využívá třídu `GenerationTaskSorter`.

`GenerationTaskList` je třída reprezentující seznam kroků generování. Poskytuje metody pro přidání nových do seznam a k jejich vykonání. Při volání metody `executeTasks`, která startuje vykonávání jednotlivých kroků, se použitím třídy `GenerationTaskSorter` kroky nejprve seřadí.

`GenerationTaskSorter` je třída zodpovědná za seřazovaní seznamu kroků dle jejich závislostí. K seřazení je použit Kahnův algoritmus pro topologické uspořádání [11].

Algoritmus spočívá v hledání „zdrojů“ (zdroje jsou vrcholy orientovaných grafu, do kterých nevede žádná hrana). Algoritmus postupně odebírá tyto zdroje z grafu, pořadí v jakém jsou odebrané dává topologické uspořádání. Odebráním zdroje z grafu mohou vzniknout nové zdroje, pokud do nějakého vrcholu vedly hrany pouze z odebraného zdroje. Pokud po postupném odebrání všech zdrojů v grafu zůstaly nějaké vrcholy, graf obsahoval cyklus a tedy nemá topologické uspořádání. Implementace algoritmu je vidět v ukázce 4.3.

Pro usnadnění implementace algoritmu byla vytvořena pomocná třída `TaskNode`. Třída reprezentuje vrchol grafu a uchovává počet hran do něj vedoucích, krok generování který reprezentuje a seznam sousedních vrcholů.

4.3.2 Generování jednotlivých údajů

Za generování jednotlivých údajů jsou zodpovědné třídy, které implementují rozhraní `DataGeneratorInterface`. Rozhraní definuje metody pro generování údaje, převádění údaje na textovou reprezentaci a nastavení zdroje náhodných dat. Pro zredukování opakování kódu při implementaci tohoto rozhraní existuje abstraktní třída `AbstractGenerator`, která poskytuje potomkům implementaci metod pro převod na textovou reprezentaci a nastavení zdroje náhodných dat.

Třídy jsou dále označeny anotací `Generator`. Tato anotace k označené třídě přidává název generátoru, datový typ generovaných dat (výčtový typ `GeneratorDataType`) a seznam parametrů generátoru. Parametry generátoru jsou definovány anotací `GeneratorParameter`. Anotace poskytuje název parametru, krátký popis a jeho datový typ (výčtový typ `GeneratorParamType`). Tyto anotace slouží k nalezení generátoru za běhu aplikace pomocí reflexe. Informace, které anotace poskytují, jsou také používané při sestavování GUI.

Definice třídy, která realizuje generátor primárních klíčů, je vidět na ukázce kódu 4.4.

Třídy, které implementují generátory jednotlivých údajů, se nacházejí v balíčku `com.gem.winch.datagen.generator`. Seznam těchto tříd a údajů jimi generovaných je v tabulce 4.1.

Při generování údajů se jako zdroj náhodných dat používá třída implementující rozhraní `RadnomnessProviderInterface`. Toto rozhraní definuje metody pro generování různých typů dat. Ve výpisu kódu 4.5 je vidět využití in-

```
List<AbstractGenerationTask> sortTasks() {
    computeDegreeIn()
    def sources = new LinkedList<TaskNode>() as Queue<TaskNode>
    nodes.each {
        if (it.inDegree == 0)
            sources.add(it)
    }

    List<AbstractGenerationTask> result = []

    for (_ in 0..<nodes.size()) {
        if (sources.empty) {
            List<TaskNode> badNodes = []
            nodes.each {
                if (it.inDegree > 0)
                    badNodes.add(it)
            }
            while (true) {
                def badNodeCount = badNodes.size()
                badNodes.removeIf({ it.neighbors.empty })
                if (badNodes.size() == badNodeCount)
                    break
            }
            def badTasks = []
            badNodes.each { badTasks.add(it.task) }
            throw new CyclicalTaskDependencyException(badTasks)
        }

        def source = sources.poll()
        result.add(source.task)
        source.neighbors.each {
            it.inDegree--
            if (it.inDegree == 0)
                sources.add(it)
        }
    }
    result
}
```

Výpis kódu 4.3: Topologické uspořádaní pomocí Kahnova algoritmu

```
@Generator(
        name = "PK",
        valueType = GeneratorDataType.INTEGER,
        parameters = [
                @GeneratorParameter(
                        name = "Počáteční hodnota",
                        valueType = GeneratorParamType.INTEGER
                )
        ]
)
class PKGenerator extends AbstractGenerator
```

Výpis kódu 4.4: Definice třídy realizující generátor PK

Třída	Generovaný údaj
`FirstNameGenerator`	Jméno
`SurnameGenerator`	Příjmení
`MailGenerator`	Email
`RcGenerator`	Rodné číslo
`GenderGenerator`	Pohlaví
`StreetGenerator`	Ulice
`CityGenerator`	Město
`DistrictGenerator`	Okres
`DateGenerator`	Datum
`TextGenerator`	Řetězec
`IntegerGenerator`	Číslo
`PKGenerator`	PK
`FKGenerator`	FK
`NullGenerator`	Null

Tabulka 4.1: Třídy implementující generátory

stance takové třídy (proměnná `random`) při generování rodného čísla, přesněji části za lomítkem.

4.3.3 Generování SQL skriptu

Pro vygenerování SQL skriptu slouží potomci třídy `AbstractSqlGenerator`. Tato třída definuje metody pro generování jednotlivých částí SQL skriptu. Potomci třídy potom poskytují implementace specifické pro daný databázový systém. Pro získání instance s implementací pro daný DBMS slouží třída `SqlGeneratorFactory`, která obsahuje metodu `geSqlGenerator`. Tato metoda podle předaného typu DMBS vrátí konkrétní implmentaci. V současné

```
def num = 0l
while (true) {
    num = (dateOfBirth.year % 100) * 10000000l +
        (dateOfBirth.monthValue +
            (gender == "f" ? 50 : 0)) * 100000l +
        dateOfBirth.dayOfMonth * 1000l +
        random.getInt(0, 10) * 100l +
        random.getInt(0, 10) * 10l +
        random.getInt(0, 10)
    if (num % 11 != 10) {
        break
    }
}
def formatString = "%09d"
if (dateOfBirth.year >= 1954) {
    num = num * 10 + num % 11
    formatString = "%010d"
}
def numString = String.format(formatString, num)
if (!useSlash)
    return numString
def builder = new StringBuilder(11)
builder.append(numString.substring(0, 6))
       .append('/').append(numString.substring(6))
builder.toString()
```

Výpis kódu 4.5: Generování rodného čísla

implementaci pro DBMS Oracle se předpokládá automatické vyplňování PK pomocí triggeru a sekvence s názvem ve tvaru „[nazevTabulky][nazevSloupce]“ (např. „OsobaID“).

Pro definici typů DBMS je použit výčtový typ `DatabaseType`. Jednotlivé hodnoty udržují název DMBS, třídu JDBC ovladače pro komunikaci s DBMS a prefix URL, který ovladač používá pro připojení.

4.4 Nedostatky a budoucí rozšíření

V této sekci jsou popsány nedostatky současné implementace a návrhy jak tyto nedostatky odstranit. Dále jsou v sekci stručně popsané možnosti dalšího rozšíření aplikace.

4.4.1 Generování unikátních hodnot

V současné implementaci se při generování nijak nehledí na to, zda mají být hodnoty generované pro nějaký sloupec unikátní. Momentálně lze zaručit unikátnost dat pouze vyexportováním SQL skriptu a jeho manuální kontrolou a případnou úpravou.

Možným způsobem jak zaručit unikátnost generovaných hodnot je každou novou hodnotu porovnat s hodnotami již vygenerovanými. Pokud by hodnota existovala, stačilo by potom vygenerovat další a proces opakovat, dokud se nevygeneruje unikátní hodnota. Je zde potřeba důkladně se zamyslet nad způsobem kontrolování unikátnosti vygenerované hodnoty, pomalá implementace by pro větší množství záznamů mohla celý proces generování velmi zpomalit.

4.4.2 Rozšíření typů generovaných údajů

V rámci této práce je implementována menší množina generátorů. Pro vylepšení použitelnosti aplikace se nabízí přidat generátory pro další typy údajů. Například generátory pro čísla bankovních účtů, DIČ, IČ nebo telefonní čísla.

Pro přidání nového generátoru je nutno provést tyto kroky:

1. Kamkoliv do balíčku `com.gem.winch` přidat třídu reprezentující nový generátor (např do balíčku `com.gem.winch.datagen.generator`).
2. Třídu označit anotací `Generator`, vyplnit název, datový typ a seznam parametrů.
3. Ve třídě implementovat rozhraní `DataGeneratorInterface`, buď přímo, nebo pomocí dědění od třídy `AbstractGenerator`.

Není třeba nijak zasahovat do modelu ani GUI aplikace – jak bylo zmíněno v sekci 4.3.2, generátory se hledají automaticky pomocí reflexe.

4.4.3 Podpora dalších DBMS

Aplikace momentálně podporuje databázové systémy Oracle a Microsoft SQL Server. Do budoucna by bylo vhodné aplikaci rozšířit o podporu jiných DMBS.

Postup pro přidání podpory nového DMBS je následující:

1. Do výčtového typu `DatabaseType` doplnit hodnotu pro nový DBMS.
2. Přidat třídu, která dědí od `AbstractSqlGenerator` a implementuje generování SQL kódu přizpůsobeného danému DBMS.
3. Do třídy `SqlGeneratorFactory` v konstruktoru přidat záznam do mapy `sqlGenerators`, který výčtový typ namapuje na konkrétní instanci SQL generátoru.

4. Do mapy `databaseNameToType` ve třídě `DatabaseConnectionDialog` přidat záznam pro namapování jména databázového systému na hodnotu typu `DatabaseType`. Tím se tento DMBS zobrazí jako možnost při konfiguraci.

4.4.4 Podpora použití více dekorátorů

Současná implementace neumožňuje uživateli pro jeden sloupec zvolit více než jeden dekorátor. Pro přidání této funkcionality bude ve třídách `ColumnModel` a `ColumnConfiguration` nutno referenci na jednu instanci dekorátoru nahradit kolekcí. V GUI bude nutno ve třídě `ColumnPanel` použít pro vybírání dekorátorů Swing komponentu, která podporuje zvolení více položek. Dále bude nutno upravit třídu `DecoratorSettingsPanel` tak, aby pracovala s více dekorátory. Nakonec bude nutno v metodě `getGeneratedDataAsStrings` třídy `ColumnModel` upravit využití dekorátoru tak, aby byly postupně aplikovány všechny zvolené dekorátory.

4.4.5 Import dat

Aplikace umožňuje uživateli ke generovaným datům pomocí GUI přidat vlastní data. Zadávání většího množství těchto dat je ale zdlouhavé. Pokud má uživatel tato data předem připravená, pro urychlení procesu jejich vyplnění se nabízí přidání možnosti data importovat. Importování by mohlo být umožněno např. z databáze nebo CSV souboru.

Pro přidaní této funkcionality by zasažení do existující kódu bylo minimální. Stačilo by pouze do GUI přidat nějaký mechanismus, jak import spustit. Například ve třídě `MainWindow` v metodě `show` při vytváření hlavní lišty přidat položku, která spustí proces importu.

Vlastní proces importování by tedy mohl být implementovaný kdekoliv v projektu. Při jeho vyvolání by mu stačilo předat instanci třídy `TableModel` právě zvolené tabulky. Ta obsahuje instanci třídy `UserDataModel`, která umožňuje metodou `addRow` přidávat nové záznamy.

4.4.6 Využití více informací o struktuře databáze

Nástroj GEM Winch poskytuje z EA repozitáře informace o struktuře databáze, které současná implementace aplikace nevyužívá. Těmito informacemi je například maximální velikost textových sloupců a tabulka referencovaná cizím klíčem. Tyto informace je možné v budoucnu využít pro automatické nastavování částí konfigurace. Momentálně je nutno, aby uživatel pro sloupce s cizím klíčem ručně zvolil generátor cizích klíčů a nastavil referencovanou tabulku jako parametr. Automatické nastavení této konfigurace by uživateli ušetřilo čas a zamezilo by potenciálním chybám vzniklých špatnou konfigurací.

Kapitola 5

Testování

Kromě samotné implementace v průběhu práce vznikly i automatizované testy. Tyto testy nepokrývají části aplikace realizují její grafické rozhraní. Jeho funkčnost byla testována manuálně v průběhu jeho vývoje. K realizaci automatického testování byla využita knihovna JUnit. Tuto knihovnu již využívá nástroj GEM Winch pro svoje testy.

Testovací data (vstupy a očekávané výstupy) pro testování jednotlivých generátorů jsou převážně definovány přímo ve zdrojovém kódu testovacích metod a tříd. Tyto třídy svým názvem odpovídají třídám testovaným: např. pro třídu `FirstNameGenerator` se testovací třída nazývá `FirstNameGeneratorTest`.

Testování funkčnosti ukládání a načítaní konfigurací ze souboru je realizováno třídami `JSONFileConfigurationReaderTest` a `JSONFileConfigurationWriterTest`. Data, která pro testování používají, jsou uložena v souboru `WinchTest.json`.

Třída `GenerationConfigurationTest` realizuje test převedení datového modelu aplikace na pomocné třídy určené k serializaci. Test opačného procesu je realizován ve třídě `GenerationModelTest`.

Třída `GenerationModelTest` dále realizuje testování vytvoření datového modelu aplikace podle databázové struktury, uložené v repozitáři nástroje EA (soubor `WinchTest.eap`). Třída je také zodpovědná za otestování funkcionality generování SQL skriptu pro vložení dat do databáze.

Míru, kterou tyto automatické testy pokrývají kód aplikace, je vidět v tabulce 5.1.

Balíček	Pokrytí		
	Třída	Metoda	Řádek
datagen.configuration	84.6%	84.6%	92.6%
datagen.configuration.persistence	100%	66.7%	75%
datagen.database	100%	86.4%	91.7%
datagen.decorator	100%	50%	50%
datagen.generation	100%	83.3%	80.7%
datagen.generator	92.9%	88.9%	97.2%
datagen.model	93.3%	82.4%	81.9%

Tabulka 5.1: Tabulka pokrytí kódu testy

Závěr

Cílů, vytyčených v rámci bakalářské práce, bylo dosaženo. V průběhu práce byla provedena rešerže a srovnání existujících nástrojů sloužících ke generování dat. Na základě srovnání pak byly sestaveny konkrétní požadavky na aplikaci. K usnadnění realizace těchto požadavků přispělo využití nástroje GEM Winch.

Podařilo se navrhnout a implementovat funkční prototyp generátoru dat, který byl navržen tak, aby bylo možné ho v budoucnu dále rozšiřovat. V současném stavu je aplikace schopna vygenerovat několik základních typů údajů a podporuje komunikaci s databázemi Oracle a MSSQL Server. V práci byla popsána možná rozšíření a stručně nastíněn postup při jejich implementaci.

Bibliografie

1. *Redgate SQL Data Generator* [online]. 2021 [cit. 2021-06-16]. Dostupné z: https://www.red-gate.com/products/sql-development/sql-data-generator/.
2. *Database Tools for Development and Management* [online]. 2021 [cit. 2021-06-16]. Dostupné z: https://www.devart.com/dbforge/.
3. *EMS SQL Manager* [online]. 2021 [cit. 2021-06-16]. Dostupné z: https://www.sqlmanager.net/.
4. *SB Data Generator* [online]. 2021 [cit. 2021-06-16]. Dostupné z: https://soft-builder.com/sb-data-generator/.
5. KREIDL, Martin; MLEJNEK, Jiří; DAVIDÍK, V. *Uživatelská příručka aplikace GEM Winch* [online]. 2018 [cit. 2021-06-18]. Dostupné z: https://www.gemsystem.cz/wp-content/uploads/WINCH-3.1.3.-U%C5%BEivatelsk%C3%A1-p%C5%99%C3%ADru%C4%8Dka.pdf.
6. *JavaFX* [online]. 2021 [cit. 2021-06-17]. Dostupné z: https://openjfx.io/.
7. BEN-KIKI, Oren; EVANS, Clark; NET, Ingy döt. *YAML Ain't Markup Language (YAML™) Version 1.2* [online]. 2009 [cit. 2021-06-17]. Dostupné z: https://yaml.org/spec/1.2/spec.html.
8. MALER, Eve; SPERBERG-MCQUEEN, Michael; YERGEAU, François; PAOLI, Jean; BRAY, Tim. *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. 2008-11 [cit. 2021-06-19]. W3C Recommendation. W3C. Dostupné z: https://www.w3.org/TR/2008/REC-xml-20081126/.
9. *JSON* [online]. 2021 [cit. 2021-06-17]. Dostupné z: https://www.json.org/json-en.html.
10. *Groovy Language Documentation* [online]. 2015 [cit. 2021-06-19]. Dostupné z: https://docs.groovy-lang.org/docs/groovy-2.3.10/html/documentation/.

11. KAHN, A. B. Topological Sorting of Large Networks. *Commun. ACM* [online]. 1962, roč. 5, č. 11, s. 558–562 [cit. 2021-06-21]. ISSN 0001-0782. Dostupné z DOI: 10.1145/368996.369025.

Příloha **A**

Seznam použitých zkratek

CI Continuous integration

CSV Comma-separated values

DBMS Database management system

EA Enterprise Architect

FK Foreign key (cizí klíč)

GUI Graphical user interface (grafické uživatelské rozhraní)

JSON JavaScript Object Notation

PK Primary key (primární klíč)

SQL Structured query language

XML Extensible markup language

YAML YAML Ain't Markup Language

Příloha B

Uživatelská příručka

V této příručce je krátce popsán postup pro vytvoření konfigurace a spuštění procesu generování dat.

B.1 Před spuštěním

Před použitím aplikace je nutné mít k dispozici soubor s repozitářem nástroje Enterprise Architect, do kterého byla importována struktura databáze. Postup pro tento import popisuje kapitola 5 uživatelské příručky nástroje Winch [5].

Dále je nutno mít nastavenou systémovou proměnnou `JAVA_HOME` s cestou k instalaci 32 bitové verze Javy.

B.2 Tvorba nové konfigurace

Po spuštění aplikace se objeví úvodní okno. Pro tvorbu nové konfigurace postupujte následovně:

1. Klikněte na levé tlačítko („Vytvořit konfiguraci z EA modelu“).

2. V dialogu, který se objeví, zvolte soubor repozitáře EA.

   - Pokud již máte nástroj Enterprise Architect spuštěný a v něm tento repozitář otevřený dialog se nezobrazí. Aplikace repozitář načte ze spuštěného nástroje.

3. Ve zobrazeném dialogu zadejte GUID Aplikace vytvořené pomocí Winch Add-In.

- GUID naleznete v nástroji EA zvolením vytvořené Aplikace v *Project Browseru*. Vpravo v *Element Properties* rozbalte položku *Project*. Odtud GUID zkopírujte i se složenými závorkami.

4. V dialogu zvolte schéma vytvořené pomocí Winch Add-In, pro které chcete vytvořit konfiguraci.

5. Po zvolení schématu se zobrazí hlavní okno s výchozím nastavením.

6. Konfiguraci nyní můžete uložit použitím hlavní lišty: *Konfigurace-Uložit*.

B.3 Načtení existující konfigurace

Pro načtení existující konfigurace po spuštění aplikace v úvodním okně klikněte na tlačítko „Otevřít existující konfiguraci“. V dialogu, který se zobrazí, zvolte soubor s uloženou konfigurací.

B.4 Úprava konfigurace

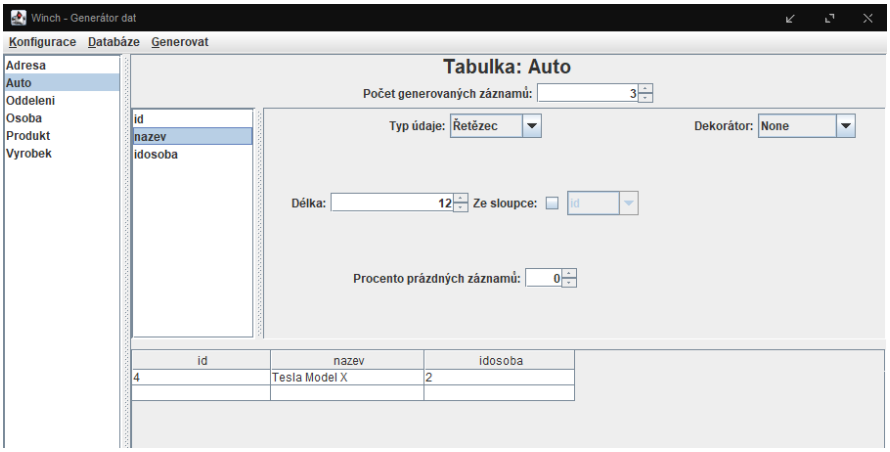
B.4.1 Konfigurace tabulky

Pro nakonfigurování počtu záznamů generovaných pro danou tabulku a ručnímu zadávání dat do tabulky postupujte následovně:

1. V levém seznamu zvolte danou tabulku.

2. Ve prostředku obrazovky pod názvem tabulky zadejte počet záznamů k vygenerování.

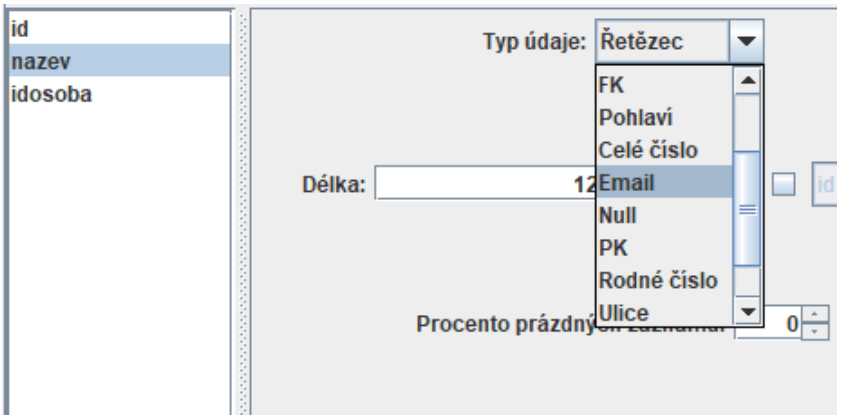
3. Do tabulky pod seznamem sloupců vyplňte data, která chcete k těm vygenerovaným přidat.



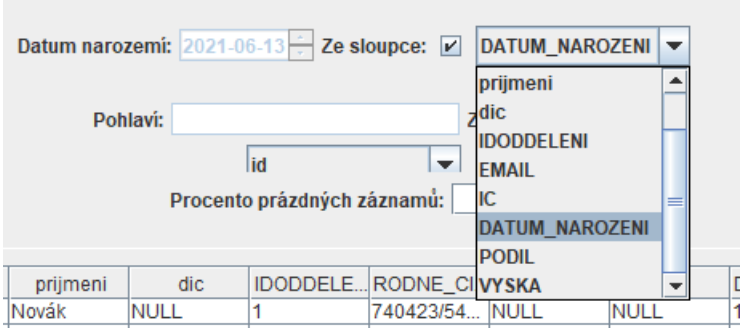
B.4.2 Konfigurace sloupce

Pro nakonfigurování dat generovaným pro dáne sloupec postupujte takto:

1. V levém seznamu zvolte tabulku, ve které se daný sloupec nachází.
2. V vpravo od seznamu tabulek zvolte daný sloupec.
3. Pro sloupec zvolte typ údaje, který chcete generovat.



4. Po zvolení typu údaje vyplňte parametry pro jeho generování.
   - Pokud má být parametr společný pro všechny záznamy, stačí vyplnit jeho hodnotu.
   - Pokud chcete, aby se hodnota parametru získala z jiného generovaného sloupce, zaškrtněte pole „Ze sloupce“ a zvolte sloupec, ze které se hodnota načte.

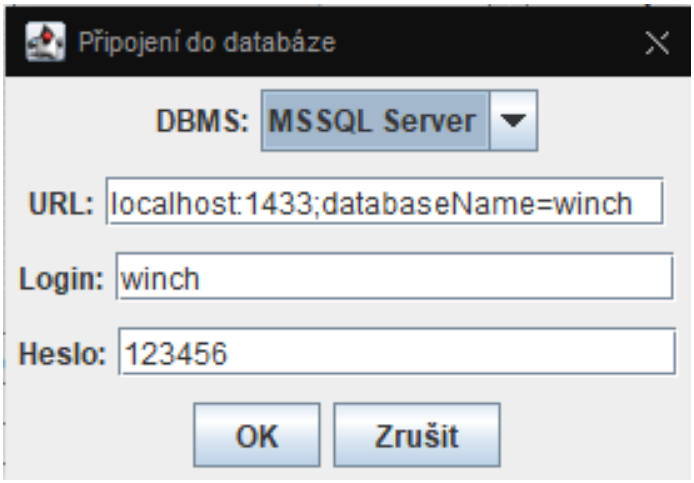


5. Po vyplnění všech parametrů zadejte kolik procent záznamů má mít hodnotu *null*.
6. Pokud potřebujete data nějak po vygenerování upravit zvolte příslušný dekorátor a vyplňte jeho parametry.

B.4.3 Konfigurace připojení k databázi

Pro nastavení připojení k databázi a zvolení typu databáze postupujte následujícím způsobem:

1. Na liště v hlavním okně klikněte na *Databáze-Nastavit*
2. Ve zobrazeném dialogu zvolte typ databázového systému, URL pro připojení, uživatelské jméno a heslo pro přihlášeni.



   - Pro SQL Server musí URL odpovídat vzoru: [ip adresa/doména serveru]:[port];databaseName=[název databáze].
     - Například: localhost:1433;databaseName=myTestDB.
   - Pro Oracle musí URL odpovídat vzoru: [ip adresa/doména serveru]:[port]:[SID].
     - Například: localhost:1521:orcl.

3. Poté klikněte na tlačítko „OK“.
4. Pro otestování nastavení databázové připojení na liště v hlavním okně klikněte na *Databáze-Otestovat*. Zobrazí se dialog se zprávou o stavu připojení.

B.5 Generování dat

Pro spuštění procesu generování postupujte takto:

1. Nejprve vytvořte nebo otevřete konfiguraci.
2. Pokud chcete vygenerovaná data rovnou vložit do databáze, klikněte na *Generovat-Do databáze*.

- Zobrazí se dialog s informací zda generování a ukládaní proběhlo v pořádku.

3. Pokud chete vygenerovaná data exportovat jako SQL skript, v liště klikněte na *Generovat-Do souboru*. V dialogu vyberte soubor, do kterého chcete SQL skript uložit.

Příloha C

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP-Sevcik-Karel-2021.pdf` ..... text práce ve formátu PDF