VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GENERÁTOR TESTOVACÍCH DAT PRO DATABÁZE FINANČNÍCH TECHNOLOGIÍ

TEST DATA GENERATOR FOR DATABASES OF FINANCIAL TECHNOLOGY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **EVA MORESOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Studentka: **Moresová Eva**
Program: Informační technologie
Název: **Generátor testovacích dat pro databáze finančních technologií**
Test Data Generator for Databases of Financial Technology
Kategorie: Analýza a testování softwaru

Zadání:

1. Nastudujte techniky testování softwaru založené na datech. Seznamte se s projektem Dataster vyvíjeným na FIT VUT pro generování testovacích dat.
2. Ve spolupráci s firmou Roger analyzujte požadavky pro generování testovacích dat reálných aplikací z domény finančních technologií (tzv. fintech).
3. Navrhněte automatický generátor testovacích dat pro relační databáze fintech zohledňující různá kritéria závislostí testovacích dat (např. strukturální závislosti odpovídající schematu databáze, datové závislosti hodnot, kombinace hodnot).
4. Implementujte generátor jako rozšíření nástroje Dataster.
5. Ověřte základní funkcionalitu generátoru automatickými testy. Demonstrujte funkcionalitu nástroje na reálných případech použití.

Literatura:

- KOTYZ, Jan. *Nástroj pro tvorbu obsahu databáze pro účely testování software*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. 2018-06-21. Vedoucí práce Smrčka Aleš. Dostupné z: https://www.fit.vut.cz/study/thesis/18066/

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Táto bakalárska práca sa zaoberá vytvorením generátora testovacích dát pre databázy aplikácií z oblasti finančných technológií. Riešenie tohto problému bolo realizované ako rozšírenie a upravenie funkcionality existujúceho nástroja dbgenx, ktorý je súčasťou platformy Testos. Vytvorený nástroj umožňuje generovať dáta s ohľadom na ich štrukturálne a sémantické závislosti, definovať vlastné externé moduly na generovanie a poskytuje efektívny zápis predpisu pre generované dáta.

Abstract

This bachelor thesis deals with the creation of a test-data generator for databases of applications in the field of financial technology. The solution was implemented as an extension and modification of the functionality of existing dbgenx tool, which is a part of the Testos platform. The created tool enables data generation with regard to its structural and semantic dependencies, definition of custom external modules for data generation and provides an efficient way to define specification of the generated data.

Kľúčové slová

generovanie testovacích dát, testovanie databázových aplikácií, testovanie softvéru, SMT riešiteľ, aplikácie finančných technológií

Keywords

test-data generation, database applications testing, software testing, SMT solver, financial technology applications

Citácia

MORESOVÁ, Eva. *Generátor testovacích dat pro databáze finančních technologií*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Generátor testovacích dat pro databáze finančních technologií

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Aleša Smrčku, Ph.D. Uviedla som všetky literárne pramene a publikácie, z ktorých som pri písaní práce čerpala.

.....
Eva Moresová
11. mája 2022

Poďakovanie

Týmto by som chcela poďakovať vedúcemu Ing. Alešovi Smrčkovi, Ph.D, za jeho pripomienky, ochotu a trpezlivosť pri vypracovaní tejto bakalárskej práce. Taktiež ďakujem pánovi Ing. Janovi Kotýzovi, na ktorého podnet vznikla téma tejto baklárskej práce za to, že sa ujal role externého konzultanta a za čas, ktorý mi venoval.

Obsah

Kapitola 1

Úvod

Testovanie je dôležitou súčasťou vývojového cyklu softvéru. Podľa niektorých odhadov tvorí údržba systému 40 až 80 % (typicky 60 %) nákladov na vývoj softvéru. Je teda žiaduce tieto výdavky minimalizovať. Jedným z typov údržby je nápravná údržba, ktorá zahŕňa opravovanie chýb vytvoreného systému. Chyby objavené po nasadení systému predstavujú oveľa väčšiu záťaž ako chyby detegované pri vývoji a preto je potrebné ich počet minimalizovať. Tomu napomáha práve dôkladný proces testovania. Avšak aj so samotným testovaním sú spojené náklady ako cena písania a údržby testov alebo cena použitých testovacích nástrojov. Výška týchto nákladov by mala byť podľa možností čo najmenšia. Podstatnou súčasťou softvérových systémov sú dáta, s ktorými narábajú a preto zohrávajú dôležitú úlohu aj pri testovaní. Proces vytvárania a údržby testovacích dát je zložitý a časovo náročný problém. Značné vylepšenie tohto procesu a následný pozitívny dopad na celkové testovacie náklady predstavuje práve generovanie testovacích dát, ktorým sa táto práca zaoberá. V moderných aplikáciách je uchovávanie dát oddelené od ich manipulácie a aplikačnej logiky. Správa dát je typicky realizovaná databázovými systémami. V roku 1970 Ted Codd predstavil návrh implementácie relačného dátového modelu, ktorý je aj napriek vzrastajúcej popularite takzvaných *NoSQL* databáz v súčasnej dobe najpoužívanejší. Z tohto dôvodu sa táto práca zaoberá generovaním dát pre relačné databázy.

Finančné technológie sú jednou z oblastí, kde je testovanie obzvlášť významné. Okrem zvýšených nákladov na údržbu spôsobujú chyby v aplikáciách finančnej sféry priame peňažné straty. Príkladom je firma Knight Capital, ktorej v roku 2012 chyba v algoritme plne automatizovaného obchodného systému takmer spôsobila bankrot, alebo výpadok burzy New York Stock Exchange v roku 2015 spôsobený nesprávnou konfiguráciou systému pri aktualizácii.

V rámci výskumnej skupiny VeriFit vznikla testovacia platforma pre podporu automatizácie testovania Testos (Test Tool Set). Táto platforma zahŕňa nástroj Dataster. Jedná sa o jednoducho ovládateľný nástroj, ktorý umožňuje naplnenie akejkoľvek relačnej databázy potrebnými testovacími dátami [10]. Cieľom tejto bakalárskej práce je vytvoriť generátor testovacích dát pre databázy systémov z oblasti finančných technologií, implementovaný ako rozšírenie nástroja Dataster.

Kapitola 2 obsahuje priblíženie teórie týkajúcej sa testovania softvéru založeného na dátach, jeho procesu, typov a popis vlastností a tvorby testovacích dát. V kapitole 3 je analyzovaná reálna aplikácia z oblasti finančných technológií a špecifikované požiadavky na vyvíjaný nástroj, ktoré z tejto analýzy vyplývajú. Kapitola 4 popisuje nástroj Dataster, konkrétne jeho komponentu dbgenx, ktorú táto práca rozširuje. V kapitole 5 sú na základe definovaných požiadaviek identifikované nedostatky pôvodného nástroja a predsta-

vené rozšírenia, ktoré tieto nedostatky riešia. Kapitola 6 popisuje implementačné detaily navrhnutých rozšírení. Kapitola 7 obsahuje zhrnutie procesu overovania funkcionality pomocou automatických testov a kapitola 8 popis demonštrácie funkcionality na reálnom prípade použitia.

Kapitola 2

Testovanie softvéru založené na dátach

Testovanie softvéru je proces spúšťania programu za účelom hľadania chýb [9]. Program spúšťame so vstupnými hodnotami a po vykonaní testu porovnávame dosiahnuté výsledky s očakávanými. Je neoddeliteľnou a dôležitou súčasťou vývoja softvéru. Pomáha zaisťovať kvalitu vytváraných aplikácií a overovať, že testovaná funkcionalita zodpovedá špecifikovaným požiadavkám. V prvej podkapitole 2.1 je popis procesu testovania, podkapitola 2.2 obsahuje špecifikáciu pojmu kvalita softvéru. Popis základných typov testovania sa nachádza v podkapitole 2.3 a popis testovacích dát v podkapitole 2.4.

2.1 Proces testovania

Proces testovania je postupnosťou na seba nadväzujúcich krokov. Tieto kroky sa líšia v závislosti od firmy a taktiež aj od konkrétneho projektu. Nasleduje popis typicky používaných krokov, ktoré sú prevzaté z [11].

Prvým krokom je definovanie zásad testovania. Je vykonávané manažmentom na úrovni organizácie. Určuje cieľ, ktorý má byť testovaním dosiahnutý. Obsah testovacích zásad sa odvíja od vyspelosti spoločnosti, typu zákazníka, používanej vývojovej metodológie ako aj typu vyvíjaného softvéru. Nasleduje určenie testovacej stratégie, ktorá definuje všeobecný prístup k testovaniu, výber testovacích metód a procesov. Zahŕňa tiež stanovenie požadovaného pokrytia kódu a rozhodnutie o automatizácii. Testovacia stratégia je definovaná bez ohľadu na konkrétny projekt. Pre jednotlivé projekty je potom manažérom alebo vedúcim testovania vytvorený testovací plán. ISTQB (International Software Testing Qualifications Board) definuje testovací plán ako dokument popisujúci rozsah, prístup, zdroje a harmonogram plánovaných testovacích aktivít [3]. Jeho šablóna je definovaná aj IEEE normou pre softvérovú a systémovú dokumentáciu testov [5], ktorá obsahuje položky ako vlastnosti, ktoré sa majú testovať, netestované funkcie, kritériá úspechu alebo zlyhania a iné. Nasleduje stanovenie cieľov testovania, ktoré sa majú dosiahnuť. Ciele testovania tvorí priorizovaný list validačných a verifikačných cieľov. Testovacím cieľom pre každý projekt je napríklad predchádzať defektom, verifikovať, či boli splnené všetky špecifikované požiadavky, kontrolovať, či je testovaný objekt hotový, a validovať jeho funkcionalitu alebo budovať dôveru v úroveň kvality testovaného objektu [13]. Ďalším dôležitým krokom je navrhnutie a následné vytvorenie testovacích scenárov a testovacích prípadov. *Scenár* je vysokoúrovňový popis testovateľnej funkcionality. Slúži ako rámec pre definovanie testovacích prípadov. *Tes-*

tovací prípad je detailný popis krokov, možných vstupov a očakávaných výstupov. Je tiež potrebné zvoliť testovacie dáta. Tento proces bližšie rozoberá podkapitola 2.4.1. Pre testovanie je potrebné vytvoriť testovacie prostredie, ktoré môže byť reálne alebo simulované. Prostredie zahŕňa špecifický hardvér, softvér, operačný systém a testovaný produkt a musí zaručiť transparentné, opakovateľné testovanie. Nakoniec nasleduje samotné spustenie testovacích prípadov s vybranými testovacími dátami, zaznamenanie dosiahnutých výsledkov a ich analýza.

2.2 Kvalita softvéru

Testovanie pomáha zaisťovať a overovať kvalitu softvérového produktu. Kvalita všeobecne je miera, s akou súbor vnútorných charakteristík produktu spĺňa požiadavky. Pre oblasť vývoja softvéru boli hodnotené charakteristiky produktu prijaté Medzinárodnou organizáciou pre normalizáciu. Norma ISO/IEC 25010:2011 [4] kategorizuje kvalitu do nasledujúcich atribútov a ich subcharakteristík:

- **funkčnosť** – vhodnosť, presnosť, interoperabilita, bezpečnosť,
- **spoľahlivosť** – vyspelosť, odolnosť proti chybám, obnoviteľnosť,
- **použiteľnosť** – zrozumiteľnosť, naučiteľnosť, prevádzkyschopnosť,
- **efektívnosť** – časové správanie, správanie týkajúce sa zdrojov, dodržiavanie účinnosti,
- **udržateľnosť** – analyzovateľnosť, modifikovateľnosť, stabilita, testovateľnosť
- **prenostiteľnosť** – adaptabilnosť, inštalovateľnosť, spolužitie, súlad so štandardami, nahraditeľnosť.

Tieto aspekty sú priamo premietnuté do špecifikácie požiadaviek softvérového produktu. Pre jednotlivé atribúty a chrakteristiky sú definované hodnoty, ktoré má produkt dosahovať. Na základe miery ich dosiahnutia je vyhodnotená kvalita produktu.

2.3 Typy testovania

Špecifikácia požadovaných vlastností systému je formulovaná do požiadaviek rôznych typov. Funkčné požiadavky definujú konkrétnu funkcionalitu, ktorú má systém poskytovať. Funkcia systému je definovaná vstupmi, následným správaním systému a očakávanými výstupmi. Nefunkčné požiadavky špecifikujú akým spôsobom bude funkcionalita poskytovaná. Týkajú sa oblastí ako bezpečnosť, výkon, znovupoužiteľnosť, udržateľnosť, a iné. Keďže rôzne požiadavky definujú rôzne aspekty aplikácie, aj k ich testovaniu je potrebné využiť rozdielny prístup. V nasledujúcich podkapitolách sú predstavené niektoré zo základných typov testovania.

Medzi ďalšie spôsoby a zamerania testovania patrí napríklad testovanie prenositeľnosti, použiteľnosti, udržovateľnosti, testovanie GUI, API, odolnosti, a iné, ktorých popis je však nad rámec tejto bakalárskej práce.

2.3.1 Funkcionálne testovanie

Funkcionálne testovanie sa zameriava na validovanie funkčných požiadaviek [8]. Overuje funkčnosť a použiteľnosť softvéru. Medzi techniky funkcionálneho testovania patrí testovanie so znalosťami interných detailov, napríklad zdrojového kódu, predtým často označované pojmom testovanie bielej skrinky (angl. white box testing). Druhou možnosťou je testovanie bez ohľadu na implementáciu, predtým označované pojmom čierna skrinka (angl. black box). Vzhľadom na tlak poznať interné detaily a zároveň zaručiť nezávislosť verifikácie a validácie od implementácie sa od takejto striktnej kategorizácie ustupuje.

Testovanie so znalosťami interných detailov sa zameriava na testovanie štruktúry programu. Pre vytvorenie testovacích prípadov je potrebná hlboká znalosť kódu. Pomáha odhaliť veľké množstvo logických chýb avšak nezabezpečuje, že program funguje tak, ako bolo zamýšľané. Tento typ testovania je najčastejšie používaný pri písaní jednotkových testov (angl. unit tests), avšak je možné sa s ním stretnúť aj na úrovni integračných a systémových testov. Jednou z techník testovania so znalosťou interných detailov je analýza pokrytia kódu. Určuje aká časť kódu bola počas testov vykonávaná. Stupeň pokrytia je definovaný na základe použitého kritéria pokrytia, napríklad pokrytie príkazov, hrán, podmienok alebo ciest.

Testovanie bez ohľadu na implementáciu overuje funkcionalitu programu. Vnútorná štruktúra a konkrétna implementácia nie je pri vytváraní testov známa. Na základe zadaných vstupov sú kontrolované očakávané výstupy. Testovacie prípady sú tvorené na základe špecifikácie požiadaviek.

2.3.2 Testovanie výkonu

Testovanie výkonu skúma ako reaguje systém pod určitou záťažou. Nie je určený na hľadanie chýb ani overenie správnosti funkcionality. Testuje rýchlosť odozvy aplikácie, stabilitu, robustnosť a iné. Medzi základné druhy testov v tejto skupine patria záťažové testy, testy hraničnej záťaže, testy odolnosti, testy časti infraštruktúry, testy objemu dát a výkonnostné testy.

2.3.3 Testovanie bezpečnosti

Cieľom testovania bezpečnosti je nájsť zraniteľnosti systému, overiť, že aplikácia je chránená pred zlomyseľnými útokmi a identifikovať potenciálne bezpečnostné hrozby. Základnými aspektmi testovania bezpečnosti sú:

- autentizácia – overuje identitu užívateľa,
- autorizácia – určuje práva užívateľa,
- integrita – systém musí zabezpečiť, že dáta ktoré poskytuje sú správne, zachovávajú svoje väzby a závislosti,
- dôvernosť – s citlivými klientskými dátami je manipulované bezpečne,
- nepopierateľnosť – zahŕňa potvrdenie prijatia správy, overenie odosielateľa,
- dostupnosť – systém je dostupný.

2.3.4 Automatické a manuálne testovanie

Testovanie aplikácií môžeme podľa spôsobu vyhodnocovania a realizácie rozdeliť do dvoch skupín – na automatické a manuálne. Automatické testovanie vykonáva softvér. Jedná sa o spustenie testovacieho skriptu, ktorý realizuje jednotlivé testovacie prípady. Tento prístup je vhodné použiť v prípadoch, kedy sú testované opakované úkony alebo existuje potreba vykonať veľké množstvo testov. Úspešne ukončený test nemusí znamenať aj správnu funkčnosť aplikácie vzhľadom na definované požiadavky. Signalizuje len, že aplikácia sa správa podľa očakávaní zadaných vo vyhodnotení testu. Vďaka tomu je ale vhodné automatické testy použiť napríklad pre regresné testovanie. Vtedy je žiadúce overiť, že sa pridaním novej funkcionality alebo refaktorizáciou kódu nezmenilo už existujúce správanie aplikácie.

Manuálne testovanie vykonáva a vyhodnocuje ručne človek. Patrí sem jednoduché prvotné testovanie novovytvorenej funkcionality, ktoré zvyčajne uskutočňuje programátor ešte počas vývoja. Taktiež ale môže znamenať aj systematické testovanie podľa testovacích prípadov vytvorených na základe definovaných požiadaviek. Tejto aktivite sa spravidla venujú dedikovaní manuálni testeri.

2.4 Testovacie dáta

Pojem testovacie dáta predstavuje vstupné hodnoty pri testovaní. Tieto hodnoty ovplyvňujú priebeh programu a zároveň môžu byť týmto procesom upravované a transformované. Zadané testovacie dáta môžu byť z pohľadu testovaného softvéru validné a ich použitím je overené štandardné správanie. Zároveň je však pri testovaní hodnotné používať aj dáta, ktoré program považuje za chybné. Tak je možné overiť ošetrenie neočakávaných stavov. Typický príklad je zadávanie nesprávnych užívateľských dát.

V ideálnom prípade by bol softvér testovaný na všetkých hodnotách, ktoré môžu vstupy nadobúdať. Tak by mohli byť odhalené všetky potenciálne chyby. To však vzhľadom na veľký počet prvkov väčšiny množín vstupných dát nie je prakticky uskutočniteľné. Preto je na testovanie vyberaná určitá podmnožina možných hodnôt testovacích dát. Podla úspešnosti testov vykonanej na tejto podmnožine odhadujeme celkovú chybovosť softvéru. Z tohto dôvodu je dôležité, aby bola zvolená množina testovacích dát kvalitná, keďže od nej priamo závisí kvalita testovania a úroveň spoľahlivosti testov. Posudzovanie kvality dát závisí vo veľkej miere od zvoleného spôsobu testovania. V nasledujúcej podkapitole 2.4.1 sú popísané žiaduce vlastnosti testovacích dát a rôzne stratégie ich výberu. Ďalej v podkapitole 2.4.2 sú rozobrané spôsoby tvorenia testovacích dát.

2.4.1 Výber a vlastnosti testovacích dát

Testovacie dáta musia byť zvolené tak, aby spôsobili stav systému, ktorý chceme testovať. Spôsob výberu a žiaduce vlastnosti testovacích dát teda závisia aj od zvoleného typu testovania.

Pre funkcionálne testovanie je dôležité, aby boli dáta rôznorodé. Typicky totiž chceme overiť, že pre rôzne skupiny hodnôt funguje systém tak, ako je špecifikovaný. Hlavnými, už spomínanými skupinami, do ktorých je možné dáta rozdeľovať, sú validné a nevalidné dáta. Jednou z významných skupín ako validných, tak nevalidných dát sú hraničné hodnoty. V prípade, že je vstup obmedzený, alebo je pre rôzne intervaly hodnôt očakávané iné správanie, sú pre testovanie vyberané krajné hodnoty týchto intervalov. V systémoch, na ktoré sú kladené vysoké požiadavky spoľahlivosti, sa tiež môže jednať o výber hraničných hodnôt

použitých dátových typov na overenie ošetrenia pretečenia typu. Správanie aplikácie tiež môže závisieť od rôznych kombinácií vstupov, ktorých výber sa realizuje napríklad podľa tabuľky rozhodnutí. V prípade, že systém prechádza viacerými stavmi je žiaduce overiť aj správnosť prechodov a postupnosti medzi týmito stavmi. Špeciálnou skupinou dát sú prázdne hodnoty. V programoch je typicky potrebné prázdne hodnoty ošetrovať osobitne, čoho zanedbanie môže spôsobiť chyby. Ďalej je pri funkcionálnom testovaní, konkrétne technike testovania s ohľadom na znalosť interných detailov, často požadovaný určitý stupeň pokrytia kódu. Značí, aká časť kódu je otestovaná. Existujú rôzne kritéria pokrytia, podľa ktorých sa tento stupeň vyhodnocuje, napríklad pokrytie príkazov, hrán, podmienok alebo ciest. Množina vybraných dát teda musí zahŕňať také hodnoty, aby po vykonaní testovacích prípadov systém prešiel každou z ciest, splnil každú podmienku a pod.

Pre výkonnostné testovanie je dôležité aby vstupné testovacie dáta boli čo najviac podobné reálnym dátam. V ideálnom prípade sú pri testovaní použité už existujúce produkčné dáta, ktoré sú podľa potreby anonymizované. Anonymizácia dát slúži na odstránenie citlivých užívateľských dát, či už nahradením za dáta s rovnakou štruktúrou a inými hodnotami alebo šifrovaním. Okrem samotných hodnôt dát je tiež dôraz na ich dostatočný objem.

Pri testovaní bezpečnosti je rovnako dôležité pre testovanie používať validné aj nevalidné dáta. Na rozdiel od funkcionálneho testovania však validitu neurčujeme na základe samotných hodnôt, ale ich významu. Na testovanie správnosti autentizácie sú vyberané prihlasovacie údaje ako pre vytvoreného, tak pre neexistujúceho užívateľa. Pri testovaní autorizácie sú vyberané rôzne kombinácie rolí a akcií tak, aby bolo overené, že určitá rola má prístup len k funkcionalite, ktorá jej náleží.

2.4.2 Tvorba testovacích dát

Testovacie dáta je možno pripravovať manuálne a často vznikajú aj v procese testovania, kedy sú dáta vytvorené v jednom testovacom prípade použité pre nadväzujúce testovacie prípady. Ďalšou možnosťou je dáta generovať.

Generovanie náhodných testovacích dát môže byť významné pre oba spôsoby testovania. Pri automatickom testovaní je generovanie možné použiť pre záťažové testy, v situáciách, kedy treba na testy veľké objemy dát. Menej vhodný je tento prístup pre automatické testy použité na regresné testovanie. V tomto prípade je totiž dáta potrebné generovať iba raz na prvotné naplnenie databázy. Potom je žiaduce, aby sa tieto dáta nemenili, keďže kontrola ich hodnôt je súčasťou vyhodnocovania úspešnosti testu.

Pri manuálnom testovaní výsledky vyhodnocuje človek, čo znamená, že náhodná povaha dát nepredstavuje problém pre určenie správnosti výsledku. Tiež je prínosné, keď je testovanie vykonávané nad rozmanitými dátami. Pre testovanie niektorých funkcií, napríklad stránkovanie, je žiadúci väčší objem dát. Pre tieto dôvody má generovanie testovacích dát veľký potenciál využitia aj v oblasti manuálneho testovania.

Kapitola 3

Analýza požiadaviek reálnych aplikácií

Podstatnou časťou tejto práce je analýza potrieb reálnych aplikácií. Nástroj musí byť použiteľný pre reálne systémy, jeho funkcionalita dostatočná pre vytváranie testovacích dát. Taktiež je kľúčové preskúmať a sformulovať motivácie pre aplikáciu vyvíjaného nástroja, ako aj spôsob, akým bude používaný. Na všetky tieto faktory je potrebné prihliadať pri návrhu, aby bolo vytvorené riešenie dostačujúce, použiteľné a užitočné.

Táto bakalárska práca je vedená v spolupráci s platobnou inštitúciou Roger. Aplikácie tejto firmy predstavujú pre účely tejto práce reprezentanta aplikácií z oblasti finančných technológií. Predpokladá sa, že potreby, ktoré vyplývajú z rozboru týchto aplikácií, dátového modelu a procesu ich testovania sa približujú požiadavkám iných podobných aplikácií. Zároveň sú požiadavky v podkapitole 3.2 sformulované so zámerom pokryť viac, než len jeden špecifický prípad tak, aby vytvorené riešenie bolo aplikovateľné na čo najširšiu skupinu aplikácií.

3.1 Platobná inštitúcia Roger

Platobná inštitúcia Roger ponúka služby rozdelené do viacerých aplikácií. Aplikácia Platba pomáha prepojením spoločností s investormi firmám a podnikateľom skrátiť dlhé splatnosti faktúr a tým vylepšiť ich cash flow. Investori formou aukcie súťažia o spoločnosťami vystavené faktúry. Aplikácia Aukce slúži pre týchto investorov a ponúka im možnosť krátkodobých investícií do faktúr vystavených za bonitnými spoločnosťami. Ďalšou službou inštitúcie Roger je Invoice Financing, ktorá slúži na posilnenie dodávateľského reťazca a uvoľnenie pracovného kapitálu. Jeho hlavnou úlohou je financovanie obchodných tokov medzi odberateľom a dodávateľom. Služba Roger Ověření poskytuje report spoločností zahŕňajúci okrem základných informácií aj dôležité dáta o ich finančnom zdraví ako výpočetný rating popisujúci mieru rizika finančnej tiesne v nasledujúcich 12 mesiacoch, posledný zistený obrat, prevádzkový výsledok a iné.

Aplikácie firmy Roger a softvérové produkty všeobecne sú neustále vyvíjané. Pri vývoji často dochádza aj k zmenám databázy či už v dôsledku pridania novej funkcionality alebo refaktorizácie kódu. Každá takáto zmena musí byť reflektovaná aj v testoch a testovacích dátach. V takom prípade má ručné vytváranie testovacích dát viacero nevýhod. Pri zmene databázy je potrebné dáta upraviť tak, aby zostali konzistentné. Pre testera, ktorý nové dáta vytvára, nemusia byť jasné všetky väzby a závislosti medzi jednotlivými tabuľkami

a záznamami. Môže teda dojsť k vytvoreniu chybných testovacích dát, čo ovplyvní efektivitu a spoľahlivosť následného testovania. Ďalšou problémovou oblasťou je časová náročnosť vytvárania dát. Vytvoriť väčší objem dát je síce mechanická, ale zdĺhavá práca. Zložitá a časovo náročná údržba testovacích dát je teda hlavnou motiváciou pre využitie generátora, ktorý by tieto problémy odstránil alebo aspoň značne zmiernil. V nasledujúcich podkapitolách je popísaný dátový model a proces testovania, ktorý je vo firme Roger používaný.

3.1.1 Dátový model

Firma Roger pre databázu používa relačný databázový systém PostgreSQL. Tradične používaným indikátorom na klasifikáciu relačných databáz je normalizačná teória, ktorá však nemusí byť dostačujúca pre meranie komplexity databázy. Piattini v článku *Database Complexity Metrics* [14] navrhuje okrem normalizácie charakterizovať databázu nasledujúcimi metrikami:

- **Hĺbka referenčného stromu** – označuje najdlhšiu referenčnú cestu medzi tabuľkami, z tabuľky T do akejkoľvek inej tabuľky v schéme. Pri výpočte tejto metriky je databáza reprezentovaná ako graf, pričom tabuľky predstavujú uzly grafu a hrany značia referenčné vzťahy medzi tabuľkami (realizované cudzími kľúčmi). Hĺbka stromu označuje počet hrán cesty v grafe, pričom cykly sú uvažované iba raz.
- **Referenčný stupeň** – značí počet referencií (cudzích kľúčov) tabuľky.
- **Počet atribútov** – vyjadruje počet atribútov (stĺpcov) tabuľky.

Pre hlavnú databázu inštitúcie Roger nadobúdajú zmienené metriky hodnoty uvedené v tabuľke 3.1.

Metrika	Priemerná hodnota	Minimum	Maximum
Hĺbka referenčného stromu	2,83	0	7
Referenčný stupeň	1,72	0	6
Počet atribútov	7,63	2	35

Tabuľka 3.1: Metriky hlavnej databázy inštitúcie Roger

Hlavná databáza obsahuje približne 100 tabuliek. Obsahuje dáta ako osobné údaje, dáta špecifické pre aplikácie z odvetvia finančníctva a dáta špecifické pre firmu Roger a ich aplikáciu. Dátový model je možné významovo rozdeliť do nasledujúcich sekcií:

- systémové dáta,
- dáta aukcií,
- finančné dáta,
- profily subjektov,
- dáta mapujúce investorov a dlžníkov.

Nasleduje krátky popis jednotlivých sekcií, uvádzajúci účel a význam dát, približný počet tabuliek v sekcii, charakter a typ dát. Pre vybrané sekcie bol vytvorený diagram

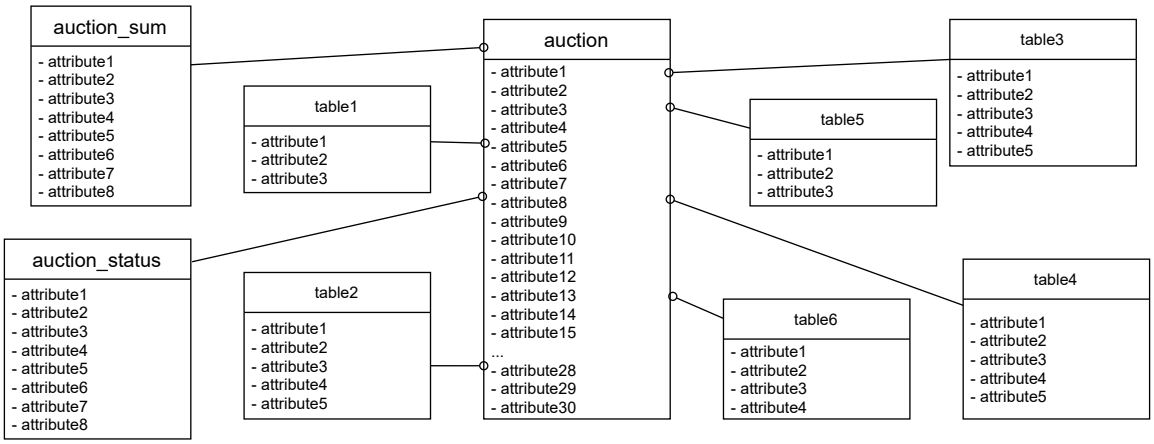
znázorňujúci databázové tabuľky a ich vzťahy. Názvy stĺpcov, prípadne celých tabuliek boli z dôvodu zachovania dôvernosti informácií týkajúcich konkrétnej štruktúry systému anonymizované. Hlavný účel uvedených diagramov je ilustrovať a vizuálne priblížiť rozsiahlosť a zložitosť charakterizovaného systému. Z tabuliek sú vynechané väzby na tabuľky mimo popisovanej sekcie.

Systémové dáta

Dáta v tejto sekcii slúžia na správu systému. Tiež slúžia na poskytovanie vedľajšej funkcionality systému, nejedná sa priamo o užívateľské dáta. Patria sem napríklad dáta súvisiace s plánovačom úloh, informácie potrebné pri zasielaní notifikácií, úložisko súborov alebo dáta súvisiace s autentizáciou a autorizáciou užívateľov. Do tejto sekcie patrí približne 20 tabuliek. Značná časť týchto dát je vytváraná v rámci použitých frameworkov a teda tieto dáta nie je potrebné generovať. Záznamy, ktoré nie sú daným frameworkom vytvorené automaticky, napríklad nastavenie systému, budú načítané z preddefinovaných dát.

Dáta aukcií

Táto sekcia zahŕňa dáta súvisiace s aukciami. Patria sem samotné aukcie, záznam ich stavu ako aj ponuky podané investormi. Táto sekcia obsahuje približne 10 tabuliek. Hlavnú časť dát tvoria číselné hodnoty, časové značky a odkazy na iné súvisiace záznamy. Obrázok 3.1 zobrazuje časť modelu týkajúci sa aukcií.



Obr. 3.1: Anonymizovaný diagram dátovného modelu aukcií

Finančné dáta

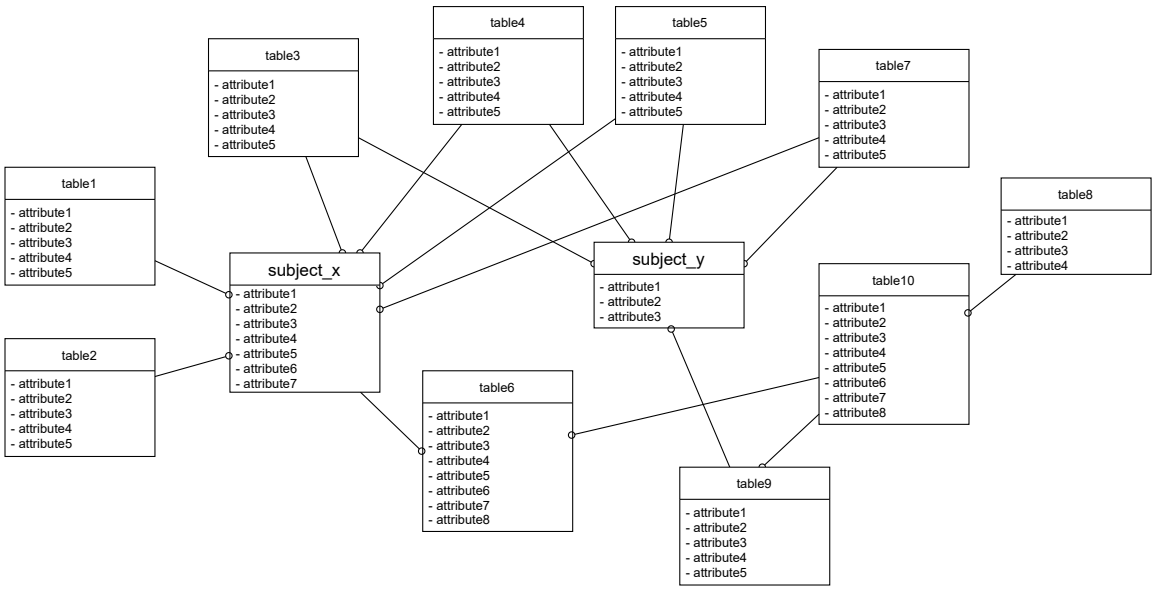
Ďalšou skupinou dát sú finančné dáta, teda faktúry a k nim zodpovedajúce súbory. Tieto záznamy obsahujú dáta zákonom požadované na fakturačných dokladoch, napríklad označenie účastníkov transakcie, označenie dokladu, peňažná suma a dátum vystavenia. Faktúry sú reprezentované menej ako 10 tabuľkami. Prevažná väčšina dát z tejto sekcie sú reťazce určitého formátu.

Profily subjektov

Subjekt predstavuje právnický subjekt, teda právnickú alebo fyzickú osobu. Subjekty sú viazané na konkrétny užívateľský účet aplikácie. Záznam o užívateľovi obsahuje osobné údaje ako meno, adresa, kontaktné údaje. Subjekt obsahuje dáta potrebné na vykonávanie transakcií dané legislatívou, napríklad identfikátory IČO a DIČ. V závislosti od typu subjektu k nemu patria aj popisné dáta ako napríklad ohodnotenie spoločnosti. Táto sekcia obsahuje približne 40 tabuliek. Vzhľadom na obsah a význam dát je hlavným ťažiskom v tejto sekcii generovanie reťazcov. Formát reťazcov je buď striktne definovaný v rámci právnych predpisov alebo v prípade osobných údajov sémanticky daný.

Dáta mapujúce investorov a odberateľov

Táto sekcia dát obsahuje reprezentáciu dodávateľov a investorov, či už ako spoločnosti alebo jednotlivca. Tiež obsahuje dáta, ktoré udávajú vzťahy medzi nimi, nastavujú limity investícií investorov do jednotlivých odberateľov. Patrí sem približne 10 tabuliek, ktoré obsahujú hlavne číselné údaje a zaznamenanie vzťahov tabuliek. Obrázok 3.2 zobrazuje zodpovedajúcu časť dátového modelu.



Obr. 3.2: Anonymizovaný diagram dátového modelu mapovania investorov a odberateľov

Pre systémy všetkých typov, systémy z odvetvia finančníctva obzvlášť, je dôležité uchovávať záznamy o auditovaní, teda záznamy o modifikácii dát, o tom kto zmeny vykonal, prípadne pôvodné hodnoty. V rámci aplikácií firmy Roger sú taktiež akékoľvek zmeny osobných údajov schvaľované ručne, aby bola zabezpečená správnosť uvedených informácií. Databáza teda obsahuje množstvo pomocných záznamov pre editované dáta.

3.1.2 Spôsob testovania

Okrem analýzy dátového modelu systému je dôležité preskúmať aj samotný proces testovania, keďže tiež ovplyvňuje požiadavky na nástroj na generovanie aj samotné dáta a ich vlastnosti. Firma Roger využíva automatické aj manuálne testovanie. Tieto pojmy sú bližšie

popísané v podkapitole 2.3.4. Testovacie dáta chcú však generovať práve pre potreby manuálneho testovania. Pôvodne bol pre naplnenie databázy pre manuálne testovanie použitý automatický skript. Pre každú databázovú tabuľku existoval súbor vo formáte csv s testovacími dátami. Skript po spustení naplnil tabuľky dátami uloženými v korešpondujúcich súboroch. Tento script a dáta zo súborov budú nahradené vytvoreným generátorom.

Nasledujúce sekcie obsahujú stručný popis základných testovacích prípadov vykonávaných manuálnym testovaním. Tieto príklady vyplývajú z primárnych prípadov použitia. Pre všetky aukcie a faktúry spomenuté v prerekvizitách jednotlivých testovacích prípadov sa predpokladá existencia všetkých závislostí, teda subjekt dodávateľa, odberateľa, nastavenie ich vzťahu, a iné.

Registrácia užívateľa

Prvým testovacím prípadom je registrácia užívateľa, investora resp. dodávateľa, do aplikácie. Po navigácii na stránku registrácie sú v jednotlivých krokoch vyplnené údaje registračného dotazníka. Po úspešnom dokončení registrácie je v databáze vytvorený záznam s podpísanými Všeobecnými Obchodnými Podmienkami a je vytvorený užívateľ. Pre tento testovací prípad nie je prerekvizitou prítomnosť žiadnych už existujúcich dát, keďže záznamy sú vytvárané vykonaním testu. Tento prípad teda reprezentuje skupinu testovacích prípadov, pre ktoré nie je potrebné testovacie dáta generovať.

Investícia do aukcií

Ďalším typickým testovacím prípadom je investovanie investorom do aukcií. Na obrázku 3.3 je zobrazený prehľad aukcií, ktorý je počiatočným bodom tohto prípadu. Prerekvizitami testu sú už existujú bežiace aukcie, teda aukcie v stave RUNNING. Nachádzajú sa medzi nimi aukcie, pre ktoré už boli investormi vytvorené ponuky, aj aukcie bez ponúk. Ďalej sú vykonané následujúce kroky:

Aukce	Odběratel	Rating	Pojištěno	Hodnota	Úrok (% p.a.) Max.	Úrok (% p.a.) Bid	Splatnost	Konec
190007	ČEZ a.s.	A	---	37 500 CZK	12,0	12,0	19. 07. 2022	20.04.2022 14:14
190006	ČEZ a.s.	A	---	37 500 CZK	12,0	---	19. 07. 2022	20.04.2022 15:13
190005	ČEZ a.s.	A	---	1 500 EUR	12,0	---	19. 07. 2022	20.05.2022 15:13
190004	ČEZ a.s.	A	---	37 500 CZK	12,0	3,3	19. 07. 2022	20.05.2022 15:13
190003	ČEZ a.s.	A	---	37 500 CZK	12,0	---	19. 07. 2022	20.05.2022 15:13
190002	ČEZ a.s.	A	---	1 125 000,23 CZK	12,0	11,8	18. 08. 2022	23.05.2022 14:58
190001	Alza.cz a.s	A	Ano	750 000 CZK	12,0	---	19. 07. 2022	23.05.2022 15:14

Obr. 3.3: Prehľad aukcií z aplikácie Roger Aukce

1. Investor1 vyberie aukciu, na ktorú ešte nebola podaná ponuka (bid).
2. Investor1 zadá svoju minimálnu ponuku o 3 % nižšiu (napr. 5 %) než je maximálny úrok (napr. 8 %) aukcie.
3. Aktuálny úrok aukcie by sa mal nastaviť na výšku maximálneho úroku (8 %).
4. Investor2 si vyberie rovnakú aukciu, ako Investor1 v bode 1.
5. Investor2 zadá svoju minimálnu ponuku o 1 % nižšiu, ako je aktuálna ponuka (7 %).
6. Aktuálny úrok aukcie sa zníži na hodnotu o 0.1 % nižšiu (6,9 %), než je minimálna ponuka Investora2 a pre aukciu je považovaný ako "momentálne vyhrávajúci" Investor1.

Financovanie faktúr

Hlavnou službou poskytovanou dodávateľom je prefinancovanie ich faktúr. Pred vykonaním testovacieho prípadu musí byť zaistená existencia viacerých nefinancovaných faktúr. Do aplikácie Invoice Financing je prihlásený užívateľ dodávateľa. Vykonávajú sa nasledovné kroky, stav aplikácie po vykonaní prvého kroku je zachytený na obrázku 3.4:

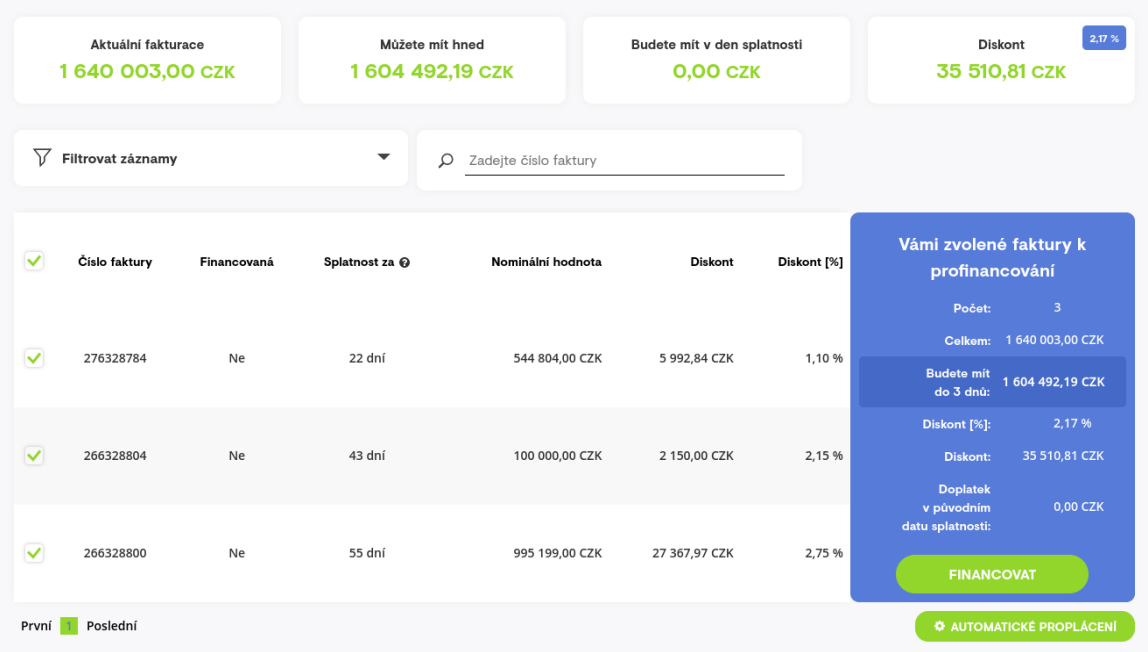
1. Užívateľ klikne na prvé zaškrtávacie políčko, ktoré vyberie všetky nefinancované faktúry zodpovedajúce aktuálnemu dodávateľovi.
2. Užívateľ klikne na tlačidlo prefinancovať.
3. Objaví sa okno s podpisom dohody, ktorá je podpísaná zadaním kódu prijatého cez SMS správu.
4. Užívateľ klikne na tlačidlo podpis zmluvy.
5. Užívateľ potvrdí výber faktúr na financovanie.

Očakávaným výsledkom je zaslanie faktúr na ďalšie spracovanie do interného systému. V tomto testovacom prípade je možné vidieť, že požiadavky na testovacie dáta sa líšia od tých na reálne dáta. Pri reálnom použití aplikácie je generovaný podpisový kód zaslaný pomocou SMS správy. Pre účely testovania je tento proces zjednodušený a je potrebné tento kód ukladať do databázy.

Export vybraných aukcií

Ďalšou funkcionalitou užívateľa investora je export ním vyhratých aukcií. Prekrekvizitou je prihlásený užívateľ, ku ktorému je vytvorených niekoľko vyhratých aukcií v rôznych stavoch. Potom sú vykonané nasledujúce kroky:

1. Užívateľ klikne na odkaz "Moje aukcie".
2. Užívateľ klikne na odkaz "Exportovať".



Obr. 3.4: Prehľad faktúr z aplikácie Invoice Financing

Očakávaným výsledkom je excel súbor (vo formáte ods) s 12 stĺpcami, ktorý pre vysporiadané aukcie (aukcie v stave CLEARED) obsahuje hodnoty pre stĺpce aukcia, koniec, odberateľ, financované, dátum financovania, dátum úhrady, splatnosť, bid, výnos, mena, výnos v CZK a menový kurz. Pre aukcie v inom stave sú vyplnené len hodnoty stĺpcov aukcia, koniec, odberateľ, financované, dátum financovania, splatnosť, bid a mena.

Veľkou skupinou testovacích prípadov je overenie zobrazovania dát. Patria sem prípady ako prehľad ukončených aukcií dodávateľa a ich informácií (stavy, čiastky, dátumy) alebo prehľad financovaných faktúr dodávateľa. Tieto testovacie prípady sú vysoko závislé na prítomnosti už existujúcich dát, keďže väčšinou testujú hlavne ich vizuálnu reprezentáciu v aplikácii. Práve pre takéto prípady predstavuje použitie generátora testovacích dát najväčšiu časovú úľavu. Ďalší testovací prípad náročný z pohľadu vytvárania testovacích dát je testovanie stránkovania. Napríklad na overenie stránkovania tabuľky aukcií musí systém obsahovať aspoň 25 aktuálne bežiacich aukcií.

3.2 Požiadavky

Táto podkapitola obsahuje špecifikáciu požiadaviek na výsledný nástroj a na testovacie dáta, ktoré má generovať. Boli sformulované na základe konzultácií s firmou Roger, analýzy ich systému, dátového modelu a s ohľadom na používaný proces testovania. Keďže táto práca rozvíja už vytvorený nástroj, niektoré z definovaných požiadaviek už sú zabezpečené. Aj tieto požiadavky sú pre úplnosť uvedené a v tabuľke odlíšené šedou farbou textu. Podkapitola 3.2.1 popisuje funkčné požiadavky a v podkapitole 3.2.2 sú rozobrané nefunkčné požiadavky.

3.2.1 Funkčné požiadavky

Funkčné požiadavky špecifikujú konkrétnu funkcionalitu, ktorú má nástroj poskytovať. Tabuľka 3.2 obsahuje požiadavky týkajúce sa samotného nástroja, jeho použitia, vlastností datasetov, spôsobu reprezentácie dát a vlastností generovaných testovacích dát.

Id	Popis
SREQ1	Dáta bude možné generovať priamo do pripojenej databázy.
SREQ2	Nástroj bude fungovať ako samostatná konzolová aplikácia.
SREQ3	Nástroj bude vedieť generovat dáta náhodným výberom z predom pripravených dát – datasetov.
SREQ4	Nástroj bude vedieť generovat dáta postupným výberom položiek z datasetov.
SREQ5	Bude možné vytvoriť si vlastný dataset.
REQ1	Dáta sú vkladané do databázy v poradí tak, aby bola zabezpečená konzistencia cudzích kľúčov.
REQ2	Nástroj bude vedieť generovat dáta náhodným výberom z predom pripravených dát tabuľkovej štruktúry.
REQ3	Generované dáta sú rôznorodé.
REQ4	Nástroj umožňuje reprezentovať reálne dáta.
REQ5	Nástroj umožňuje generovať dáta s agregačnými závislosťami medzi stĺpcami tabuliek.

Tabuľka 3.2: Funkčné požiadavky nástroja

Poradie vkladania dát – Reálne dáta obsahujú veľké množstvo väzieb, ktoré sú v databáze realizované cez cudzie kľúče, je preto nevyhnutné, aby bolo zabezpečené ich korektné vkladanie. Túto požiadavku zabezpečuje rozšírenie popísané v podkapitole 5.1.

Štruktúrovaný dataset – Okrem datasetu obsahujúceho preddefinované hodnoty pre jeden atribút typu, respektíve stĺpec tabuľky, bude možné použiť preddefinované dáta pre set atribútov. Požiadavku realizuje rozšírenie 5.4.

Rôznorodosť dát – V kontexte tejto práce znamená rôznorodosť dát zastúpenie hodnôt z rôznych skupín testovacích dát, tak ako sú popísané v podkapitole 2.4.1. Požiadavka je realizovaná rozšírením v podkapitole 5.2.

Reálne dáta – Nástroj musí poskytovať možnosť reprezentovať reálne dátové typy a väzby medzi objektmi. Požiadavku zabezpečujú rozšírenia 5.5 a 5.8.

Agregačné väzby – Nástroj musí poskytovať možnosť vytvárať aj hodnoty, ktoré vznikajú na základe výpočtu z iných hodnôt, konkrétne suma, počet a priemer. Túto požiadavku realizuje rozšírenie 5.3.

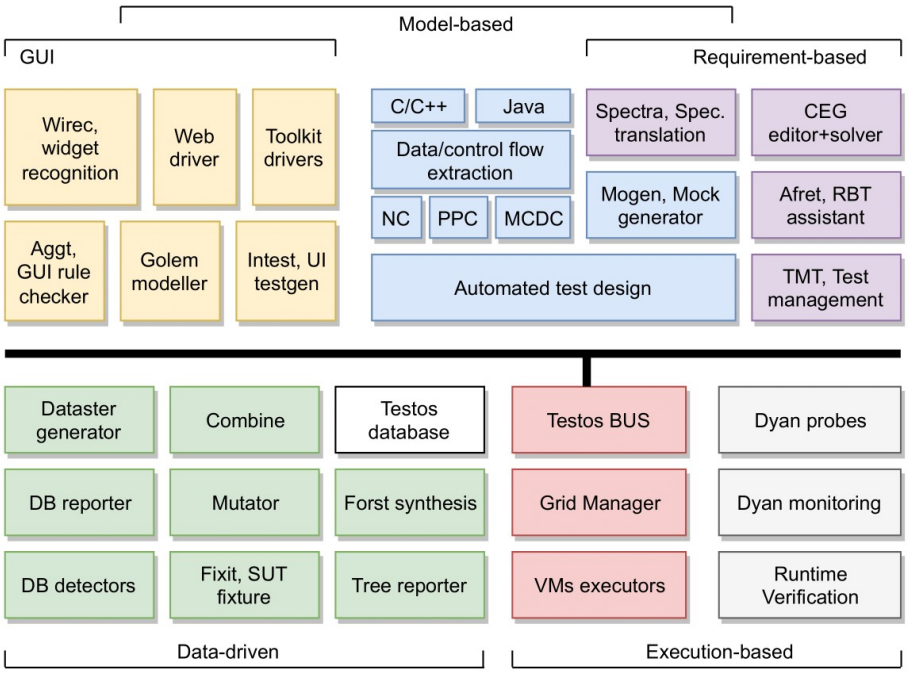
3.2.2 Nefunkčné požiadavky

Hlavnou motiváciou pre použitie generátora testovacích dát v praxi je časová úspora. Je teda potrebné zabezpečiť aby použitie nástroja túto potrebu spĺňalo. Najviac časovo náročnou časťou pri používaní, je vytvorenie predpisu pre generovanie testovacích dát. Tento proces tiež vyžaduje aktívnu účasť užívateľa nástroja, na rozdiel od samotného generovania. Preto sa jedná o oblasť, pre ktorú je efektivita najdôležitejšia. Urýchlenie vytvárania konfiguračného súboru realizuje rozšírenie popísané v podkapitole 5.6.

Kapitola 4

Nástroj dbgenx

Dataster je projekt vyvíjaný v rámci platformy Testos, viď obrázok 4.1, ktorá podporuje automatizáciu testovania softvéru a zahŕňa viaceré úrovne testovania. Jednou z nich je dátami riadené testovanie, kam patrí aj tento projekt.



Obr. 4.1: Platforma Testos [15]

Dataster sa skladá z viacerých aplikácií. Zahŕňa webové užívateľské rozhranie Dataster, ďalej samotný generátor dbgenx a jeho rozšírenie nástrojom Combine, ktorý umožňuje generovať dáta na základe požiadaviek kombinačného testovania. Táto bakalárska práca nadväzuje na nástroj dbgenx, vyvinutý v rámci inej diplomovej práce [10]. Cieľom je upraviť a rozšíriť tento nástroj pre potreby firmy Roger a iných aplikácií finančných technológií. V tejto kapitole je v krátkosti priblížená jeho pôvodná funkcionalita a bližšie popísané časti, ktoré sú pre túto prácu podstatné.

Dbgenx je nástroj na generovanie dát pre naplnenie databázy za účelom testovania. Ako vstup prijíma konfiguračný súbor, ktorý obsahuje popis štruktúry a obmedzení pre generované testovacie dáta a jeho formát je bližšie predstavený v podkapitole 4.1. Na základe konfiguračného súboru potom prebieha generovanie dát pomocou rôznych generátorov v závislosti od typu dát. Výsledok je uložený do databázy alebo vypísaný vo forme SQL príkazov na štandartný výstup, prípadne do súboru. Tento proces je priblížený v podkapitole 4.2.

4.1 Konfiguračný súbor

Vstupný súbor, nazývaný aj recept, obsahuje popis konfigurácií generátora. Špecifikuje štruktúru databázy, počet záznamov jednotlivých typov, závislosti a väzby medzi dátami, ako aj sémantické obmedzenia pre generované dáta. Využíva formát YAML a skladá sa z piatich sekcií — **include**, **types**, **schema**, **constraints**, **generate**, pričom položky **include** a **constraints** sú nepovinné.

- **include** – Sekcia **include** sa skladá zo zoznamu datasetov. Dataset je súbor, ktorý obsahuje list záznamov, z ktorých sú pri generovaní vyberané hodnoty. Slúži na zabezpečenie charakteru reálnych dát a v sekcii **types** je použitý ako jednoduchý typ. Dataset je taktiež zapisovaný vo formáte YAML.
- **types** – Sekcia **types** obsahuje definíciu typov vo forme asociatívneho poľa. Kľúčom je názov typu a hodnotou jeden z preddefinovaných typov, dataset alebo iný užívateľom vytvorený typ. V prípade, že typ obsahuje podtyp, je možné uviesť aj počet objektov podtypu. Dbgenx poskytuje nasledujúce preddefinovane typy: integer, big integer, float, interval, float interval, dátumový interval, boolean, reťazec, formátovaný reťazec, reťazec so špecifikovanou dĺžkou, reťazec odpovedajúci regulárnemu výrazu, sekvencia.
- **schema** – Sekcia **schema** obsahuje definíciu štruktúry databázových tabuliek a priradenie typov k jednotlivým stĺpcom. Toto priradenie je ďalej nazývané aj ako mapovanie typov na tabuľky. Ďalej obsahuje aj špecifikáciu cudzích kľúčov pomocou výrazu FROM, ktorý sa na rozdiel od ostatných stĺpcov vzťahuje nie na typ, ale na stĺpec inej tabuľky.
- **constraints** – Sekcia **constraints** obsahuje užívateľom zadané podmienky, ktoré sa vzťahujú k typom a podtypom. Umožňuje definovať závislosti medzi položkami v rámci typu a určiť sémantické obmedzenia. Zapisuje sa formou jednoduchého predikátu. Momentálne je podporované vytvárať podmienky len pre atribúty vybraných dátových typov: integer, big integer, float, interval, float interval, dátumový interval a boolean. Podporované operátory sú: =, !=, <, >, <=, >=.
- **generate** – V sekcii **generate** je uvedený list typov a čísiel, ktoré určujú aký počet objektov daného typu sa má generovať. Ak typ obsahuje podtypy, tie sú automaticky generované spolu s nadradeným typom v počte uvedenom pri jeho definícii.

Na obrázku 4.2 je uvedená ukážka jednoduchého konfiguračného súboru. Reprezentuje príklad rezervačného systému ubytovacieho zariadenia. V sekcii **include** obsahuje zahrnutie dátových setov použitých ako hodnoty mena užívateľa. Sekcia **types** obsahuje definíciu typov izba, rezervácia a užívateľ. V sekcii **schema** je definované mapovanie jednotlivých typov na zodpovedajúce tabuľky. Taktiež je farebne rozlíšené či sa hodnota priradená jednotlivým stĺpcom vzťahuje k sekcii **types** alebo **schema**. Na riadkoch 34 a 35 je realizované

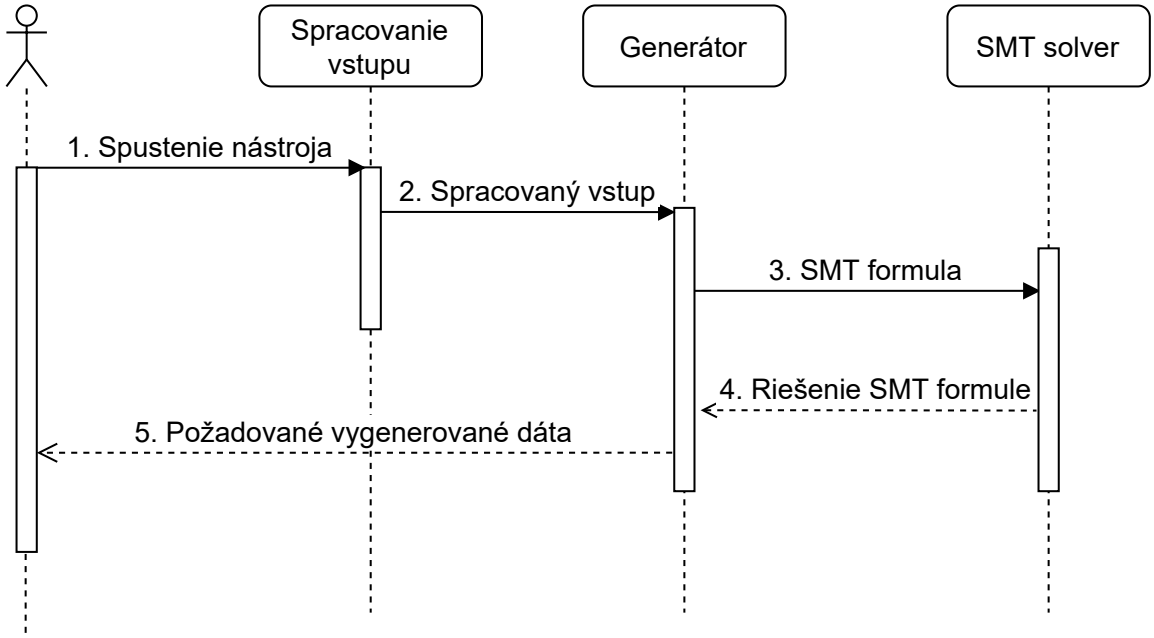
prepojenie rezervácie so záznamom užívateľa a izby pomocou cudzieho kľúča. Sekcia **constraints** obsahuje sémantické obmedzenie dátumov rezervácie tak, aby začiatočný dátum vždy predchádzal koncový dátum. V sekcii **generate** je uvedený počet genovaných záznamov jednotlivých typov.

```
include:
  - first_names
  - last_names

types:
  room:
    number: seq(0, 100,1)
    capacity: interval(1,5)
    en_suite_bathroom: bool
  reservation:
    from: dinterval(2022-06-01, 2022-08-01)
    to: dinterval(2022-06-01, 2022-08-01)
  user:
    id: seq(0, 10000, 1)
    first_name: first_names
    last_name: last_names

schema:
  room_t:
    number: room.number
    capacity: room.capacity
    en_suite_bathroom: room.en_suite_bathroom
  user_t:
    id: user.id
    first_name: user.first_name
    last_name: user.last_name
  reservation_t:
    from: reservation.from
    to: reservation.to
    room_number: "FROM room_t.number"
    user_id: "FROM user_t.id"

constraints:
  - reservation.from < reservation.to

generate:
  - room = 10
  - user = 5
  - reservation = 7
```

Obr. 4.2: Konfiguračný súbor pre rezervačný systém. Zelená farba predstavuje identifikátory spojené s dátovými typmi, modrá indikuje vzťah ku schéme databázy

4.2 Priebeh generovania dát

Po načítaní vstupného súboru prebieha transformácia jeho jednotlivých sekcií do vnútornej reprezentácie. Nasleduje samotné generovanie dát, na ktoré sú použité rôzne spôsoby, v závislosti od typu generovaných dát. Výsledok je poslaný na výstup. Celý proces možno vidieť zakreslený na obrázku 4.3, ktorý bol prevzatý z diplomovej práce, na ktorú táto práca nadväzuje [10].



Obr. 4.3: Sekvenčný diagram priebehu generovania dát

4.2.1 Spracovanie konfiguračného súboru

Položka **constraints** je prevedená na list trojíc, skladajúcich sa z ľavého operátora, operandu a pravého operátora. Tieto podmienky sú neskôr prevedené do podoby SMT formulí a použité pri generovaní pomocou SMT riešiteľa, ktorý je bližšie popísaný v podkapitole 4.3. Z reťazcov v sekcii **generate** sú vytvorené dvojice názvu typu a hodnoty. Dáta načítané zo sekcií **include** a **schema** zachovávajú rovnakú štruktúru ako v konfiguračnom súbore. Pre každý z typov **types** je vytvorený objekt jednej z preddefinovaných tried s unikátnym názvom. Triedy typov generovaných SMT riešiteľom implementujú funkcie pre vytvorenie SMT formulí.

4.2.2 Generovanie dát

Číselné, booleovské dáta a dátumy sú generované pomocou SMT riešiteľa. Sekvencie sú generované na základe sekvenčného čítača triedy, ale taktiež do riešiteľa vstupujú, a to v podobe formule `i = hodnota`. Na základe tried objektov vytvorených z položky **types**, ich unikátnych mien a podmienok zo sekcie **constraints** sú vytvorené SMT formule, ktoré sú vstupom pre riešiteľa. Po nájdení riešenia sú výsledné hodnoty uložené do príslušných objektov.

Reťazcové hodnoty sú generované samostatne. Prvým spôsobom je vytváranie náhodných alfanumerických reťazcov, ktoré spĺňajú obmedzenia vyplývajúce z použitého typu reťazca. Ďalej môže byť hodnota vybraná z datasetu alebo vytvorená spojením viacerých častí vygenerovaných predošlými spôsobmi, kedy sa jedná o takzvaný štrukturovaný reťazec.

4.2.3 Vkladanie dát do databázy

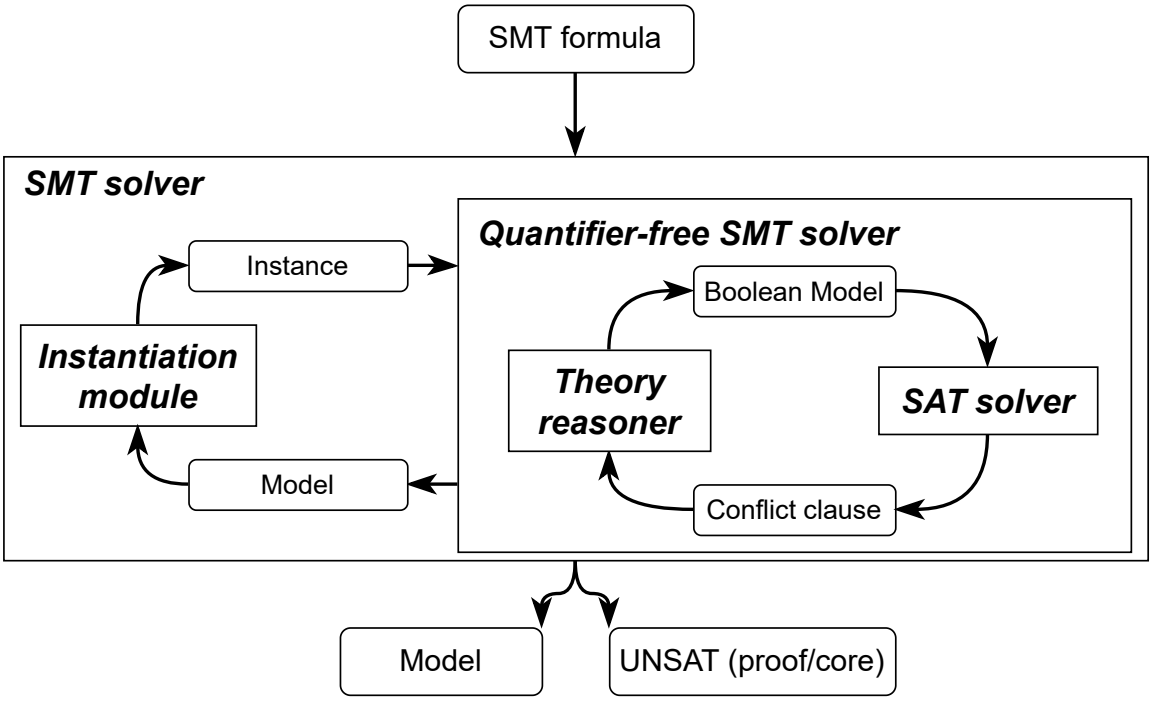
Po vygenerovaní hodnôt sa jednotlivé typy mapujú na stĺpce tabuliek definovaných v sekcii **schema**. Spôsoby, akými je zabezpečená konzistencia cudzích kľúčov, sú detailnejšie analyzované v podkapitole 5.1. Ak je možné namapovať všetky stĺpce tabuľky, je vytvorený SQL insert príkaz, v opačnom prípade je daný riadok vynechaný. Po vytvorení príkazov pre všetky tabuľky je uskutočnená databázová transakcia, prípadne sú SQL príkazy vypísané na štandardný výstup alebo vložené do súboru.

4.3 SMT riešiteľ

Dôležitou súčasťou nástroja dbgenx je komponent SMT riešiteľ (angl. SMT solver). Splnitelnosť modulo teórií (angl. satisfiability modulo theories), ďalej len SMT, je nástroj, ktorý rozhoduje splniteľnosť alebo validitu formulí v rámci vybraných teórií, napríklad operácie nad celými číslami alebo reťazcami [12]. Je generalizáciou problému splniteľnosti booleovskej formule, označovaného ako SAT. Na rozdiel od SAT okrem booleovského výrazu s premennými umožňuje SMT definovať formule obsahujúce predikátovú logiku. Dáva možnosť vyjadriť porovnávanie, aritmetiku, kvantifikátory, celé a reálne čísla, štruktúry ako bitové vektory, polia a iné. Formula je splniteľná, keď je možné nájsť riešenie pre hodnoty premenných tak, aby formula platila. Formula tiež môže byť nesplniteľná v prípade, že obsahuje kontradikčné podmienky. SMT riešitele sú využívané v oblasti statickej kontroly, verifikačných systémoch, pre kontrolu modelov, abstrakciu predikátov či generovanie testovacích prípadov. Obrázok 4.4, zobrazuje schématický pohľad na SMT riešiteľa. Vstupom riešiteľa sú SMT formule a výstupom je v prípade úspechu model, v opačnom prípade dôkaz alebo jadro nesplniteľnosti. Formule obsahujúce kvantifikátory sú redukované vytvorením inštancií, čo zabezpečuje modul pre vytváranie inštancií a riešené riešiteľom bez spracovania kvantifikátorov.

Nástroj dbgenx poskytuje možnosť výberu konkrétneho SMT riešiteľa, čo je realizované pomocou knižnice PySMT [2]. Táto knižnica poskytuje prístup cez jednotné rozhranie k riešiteľom MathSAT, Z3, CVC4, Yices 2, CUDD, PicoSAT a Boolector.

SMT riešiteľ v rámci nástroja slúži na generovanie testovacích dát. Hodnoty typov sú reprezentované premennými. Spolu s podmienkami a obmedzeniami, ktoré definujú jednotlivé dátové typy, sú prevedené do podoby SMT formulí. Riešiteľ následne nájde hodnoty jednotlivých premenných, ktoré prestavujú riešenie zadanej formule. V prípade, že je požiadavka generovať viacero objektov zodpovedajúcim rovnakej formule, teda viacero objektov toho istého typu, je nutné zaistiť, že riešiteľ vždy nájde iné riešenie. Z toho dôvodu je po nájdení riešenia do formule pripojená pre každý súbor premenných podmienka udávajúca, že nesmie byť použitá rovnaká kombinácia hodnôt ako v už nájdenom riešení. V prípade, že je formula nesplniteľná, teda nie je možné generovať hodnoty, končí generovanie neúspešne. Táto situácia môže nastať aj v prípade, že neexistuje taký počet riešení aký je požadovaný. Spôsob akým je riešiteľ používaný spôsobuje, že sa pravidlá a obmedzenia aplikujú vždy iba pre daný typ, ktorý sa generuje. Nie je teda možné definovať vzájomné obmedzenia medzi nevnorenými typmi (vnorený typ je generovaný spolu s jeho nadradeným typom).



Obr. 4.4: Schematický pohľad na SMT riešiteľ, prevzaté z [7]

Pri použití konfiguračného súboru z obrázka 4.2 sú pre riešiteľ generované nasledujúce SMT formule. Formula pre izbu generovanú tretiu v poradí:

```
(room.number = 2) &
((1 <= room.capacity) & (room.capacity <= 5)) &
((0 <= room.en_suite_bathroom) & (room.en_suite_bathroom <= 1)) &
(! ((room.capacity = 1) & (room.en_suite_bathroom = 0))) &
(! ((room.capacity = 3) & (room.en_suite_bathroom = 0)))
```

Prvý riadok obsahuje obmedzenie atribútu `number`, ktorý je typu sekvencia a teda jeho hodnota je v SMT formule určená sekvenčným čítačom triedy, ako je uvedené v podkapitole 4.2.2. Druhý a tretí riadok definujú obmedzenia vyplývajúce zo zvolených typov atribútov `capacity` a `en_suite_bathroom`. Riadky 4 a 5 obsahujú vyššie spomínanú podmienku zaručujúcu rozdielnosť nájdených riešení. Keďže je generovaná tretia izba, formula obsahuje dve tieto podmienky pre nájdené prvé a druhé riešenie.

Formula pre generovanie prvého užívateľa:

```
(user.id = 0)
```

Keďže typ `user` obsahuje okrem sekvencie atribúty, ktorých typom je dataset a teda nie sú generované SMT riešiteľom, obsahuje formula užívateľa len podmienku pre sekvenciu. Konkrétna hodnota `id` je opäť určovaná sekvenčným čítačom. Formula pre generovanie druhej rezervácie v poradí má nasledovnú podobu:

```
((19144 <= reservation.from) & (reservation.from <= 19205)) &
(reservation.from < reservation.to) &
((19144 <= reservation.to) & (reservation.to <= 19205)) &
(! ((reservation.from = 19144) & (reservation.to = 19145)))
```

Prvý a tretí riadok formule vyjadrujú obmedzenie intervalu dátumu určeného ako typ atribútov `from` a `to`. Na druhom riadku je predikát, ktorý vznikol z podmienky uvedenej v sekcii **constraints**. Posledný riadok opäť zaručuje nerovnosť predošlému riešeniu.

Kapitola 5

Návrh rozšírení nástroja dbgenx

Zo zistených požiadaviek reálnych systémov na navrhovaný nástroj a funkcionality momentálne poskytovanej generátorom dbgenx, vyplýva niekoľko oblastí, v ktorých je súčasné riešenie nedostačujúce, prípadne poskytuje priestor pre zlepšenie. V tejto kapitole sú predstavené návrhy rozšírení, ktoré tieto problémy riešia.

5.1 Poradie vkladania záznamov do tabuliek

V pôvodnej implementácii je možné vyjadriť vzťahy medzi objektmi dvoma spôsobmi. Prvou možnosťou je definovanie kompozičného vzťahu, kedy je podriadený objekt definovaný ako atribút rodičovského objektu. V tomto prípade je poradie vkladania záznamov do tabuliek určené podľa zanorenia objektu, teda najskôr je vložený záznam rodičovského objektu a až následne záznam objektu. Teda nemôže dôjsť k situácii, kedy by bol vložený záznam odkazujúci na iný, ešte neexistujúci záznam.

Druhou možnosťou na vyjadrenie vzťahu medzi dvoma objektami je použitie kľúčového slova FROM, ktoré umožnuje vkladať do stĺpcov tabulky hodnoty zo stĺpcov inej tabuľky. Týmto spôsobom je možné definovať väzbu (cudzí kľúč) aj pre tabuľky, ktoré nie sú navzájom hierarchicky závislé. Tento spôsob však neurčuje poradie vkladania záznamov objektov do databázy. Toto poradie je určené výlučne postupnosťou typov definovaných v **generate** sekcii konfiguračného súboru a je teda na jeho autorovi, aby bol zoznam zoradený správne. Pri malom počte tabuliek a jednoduchých vzťahoch je takéto riešenie uskutočniteľné, avšak s narastajúcim objemom typov a zložitosti vzťahov predstavuje pre človeka časovo náročný problém.

Na to, aby bol odstránený tento problém pôvodného riešenia, bude potrebné priradzovať konkrétne hodnoty cudzích kľúčov až po vygenerovaní všetkých dát. Dáta budú teda počas generovania len ukladané a samotné SQL príkazy budú vytvorené až po ukončení tohto procesu. Toto však spôsobuje problém pri určovaní správneho poradia v prípade kompozičných vzťahov, kde bolo správne poradie zabezpečované názvami premenných, ktoré po dokončení generovania zanikajú. Je teda potrebné pre určenie poradia použiť iný prístup.

Na určovanie poradia tabuliek bude použitý algoritmus topologického triedenia. Tento algoritmus je bežne používaný napríklad v nástrojoch na kompiláciu na určenie poradia úloh v makefile, serializáciu dát a riešenie závislostí symbolov v linkeroch [6]. Princíp algoritmu spočíva v zoradení uzlov orientovaného grafu tak, aby každý uzol nasledoval vždy pred všetkými uzlami, na ktoré ukazuje. Algoritmus topologického triedenia je možné použiť len pre acyklické grafy. V databáze sa cyklická väzba vyskytnúť môže, avšak ich výskyt

je v porovnaní s výskytom necyklických väzieb v rámci cudzích kľúčov značne nižší. Preto je aj napriek tomuto nedostatku toto riešenie výhodnejšie než pôvodný prístup. Problém reprezentovania cyklických závislostí teda nebude vo vytvorenej verzii zohľadňovaný a riešený.

5.2 Viaceré skupiny obmedzení v sekcii constraints

V sekcii **constraints** konfiguračného súboru sú definované užívateľom zadané obmedzenia na vybrané typy a podtypy. Dáta sú potom generované tak, aby spĺňali zároveň všetky z týchto podmienok. Uvažujme situáciu, kedy hodnota jedného stĺpca tabuľky ovplyvňuje inú, napríklad podľa typu faktúry je určený interval percent, ktoré môžu byť z hodnoty faktúry prefinancované. V takomto prípade by bolo možné v sekcii **constraints** zapísať podmienky len pre jeden zo stavov faktúry a tomu odpovedajúcu hodnotu percent prefinancovania, čo by v dôsledku znamenalo, že všetky generované faktúry by boli rovnakého typu.

Reálne teda uvedený príklad nie je možné definovať použitím **constraints** v súčasnej podobe, keďže sa nejedná o podmienky, ktoré by mali platiť zároveň všetky naraz, ale o skupiny obmedzení, z ktorých má pre objekt platiť vždy jedna vybraná. Taktiež je žiaduce, aby dáta obsahovali objekty z viacerých týchto skupín, čo vyplýva z požiadavky na rôznorodé generované dáta.

```
types:
    coordinate:
        x: int
        y: int
schema:
    coordinate_t:
        x: coordinate.x
        y: coordinate.y
constraints:
    common:
        - coordinate.x < 100
    x_axis:
        - coordinate.y = 0
    first_quadrant:
        - coordinate.x > 0
        - coordinate.y > 0
generate:
    - x_axis.coordinate = 2
    - first_quadrant.coordinate = 1
```

Obr. 5.1: Konfiguračný súbor pre generovanie bodov v rovine obsahujúci viaceré skupiny obmedzení.

Rôzne podmienky na testovacie dáta obvykle zopovedajú rôznym testovacím prípadom, respektíve skupinám testovacích prípadov. Toto vylepšenie má teda umožniť generovanie

testovacích dát pre rozličné skupiny prípadov bez toho, aby bolo nutné definovať viacero konfiguračných súborov.

Vzhľadom na uvedené nedostatky boli navrhnuté nasledujúce zmeny. Sekcia **constraints** bude ďalej rozčlenená na pomenované skupiny podmienok, pričom pre definovanie tých, ktoré majú platiť vo všetkých prípadoch bude slúžiť špeciálna skupina *common*. Ďalej je nutné v súlade s týmito zmenami upraviť aj sekciu **generate**, kde bude pridaná informácia o tom, ktorá skupina podmienok sa má pri generovaní daného typu použiť.

Na obrázku 5.1 môžeme vidieť jednoduchý príklad konfiguračného súboru s viacerými skupinami obmedzení, ktoré sú zvýraznené. Slúži na generovanie bodov v rovine. Pre všetky z nich je aplikované globálne obmedzenie pre hodnotu súradnice x zapísané na riadkoch 10 až 11. V sekcii **generate** je potom uvedené, že požadujeme generovať dva body na osi x (riadok 18) a jeden bod, ktorý leží v prvom kvadrante (riadok 19).

5.3 Generovanie agregovaných závislostí

Pôvodná verzia dbgenx neumožňuje vyjadriť agregované závislosti, teda že stĺpec v jednej tabuľke je agregačnou funkciou stĺpca v inej tabuľke. Táto možnosť je potrebná napríklad v prípade, keď je celková cena aukcie sumou cien jej položiek.

Pri návrhu tejto funkcionality je možné uvažovať v dvoch smeroch. Je možné sa na informáciu o tom, že atribút typu obsahuje výsledok agregácie pozerať ako na nový druh podmienky pre daný typ a teda ju definovať v sekcii **constraints**. To by znamenalo, že bude spracovaná pri generovaní dát pomocou SMT solvera. Bolo by teda potrebné, aby objekty pre jednotlivé typy neboli generované osobitne, ale všetky naraz a generovací proces by musel byť značne pozmenený.

V druhom prístupe je táto hodnota dopočítaná po vygenerovaní príslušných objektov, z ktorých sa má určiť a teda ide o dodatočné spracovanie generovaných dát. V tomto prípade by bolo možné túto informáciu uviesť až na úrovni stĺpca tabuľky, avšak znamenalo by to, že pre daný stĺpec nie je možné definovať obmedzenia. Vzhľadom na zložitosť prvého riešenia bol tento prístup vybraný aj napriek tomuto nedostatku.

5.4 Štrukturované datasety

V minulej verzii nástroja dbgenx je možné definovať dataset pre jednotlivé stĺpce. Toto rozšírenie pridá možnosť vytvárať štruktúrované datasety, v ktorých budú zapísané hodnoty pre viacero stĺpcov, prípadne celú tabuľku. Pôvodný dataset súbor je formátu yaml. Vzhľadom na tabuľkovú povahu dát, ktoré obsahuje štruktúrovaný dataset, však nie je preň tento formát vhodný. Namiesto toho bol vybraný csv formát, keďže sa doň dajú ľahko exportovať už existujúce dáta z databázy alebo excel súboru a tento formát je pre tabuľkové dáta ľahšie upravovateľný a viac prehľadný. Prvý riadok štruktúrovaného datasetu musí obsahovať názvy stĺpcov. Na tie sa potom bude odkazovať v definícii typu vo formáte `<dataset_súbor>.<stĺpec>`.

5.5 Typ `text`

Pri analýze databázy reálneho systému vyvstala potreba generovať dlhší blok textu, napríklad pre popis firmy alebo zhodnotenie aukcie. Pre testovacie účely nie je potrebné, aby bol tento text zmysluplný, ale je žiaduce aby mal vizuálnu formu súvislého textu, deleného do

slov a viet. Súčasné reťazcové typy, ktoré dbgenx ponúka, nie sú vhodné pre toto použitie a preto bol navrhnutý nový typ TextBlock. V konfiguračnom súbore môže byť použitý ako `text(N)`, pričom `N` je maximálny počet znakov generovaného textu.

5.6 Skrátenie zápisu schémy databázy

V prípade, kedy sa atribúty objektu zhodujú s názvami zodpovedajúcich stĺpcov, dochádza pri definovaní štruktúry tabuľky v sekcii **schema** v konfiguračnom súbore k zbytočnej duplikácii informácií, ktoré je možné získať aj z definície typu. Aby sa predišlo tomuto opakovaniu, bude možné využiť skrátený zápis. Pri definícii tabuľky bude namiesto celej jej štruktúry uvedený špeciálny stĺpec `"*"`, ktorého hodnota definuje typ a jeho atribúty, ktoré sa majú do tabuľky skopírovať. Na obrázku 5.2, je uvedený príklad časti konfiguračného súboru s navrhnutým zápisom:

```
1  types:
2    house:
3      id: int
4      rooms: int
5    owner:
6      id: int
7      name: string
8      age: int
9
10 schema:
11   house_t:
12     "*": house.*
13     owner: "FROM owner_t.id"
14   owner_t:
15     id[PK]: owner.id
16     "*": owner.[name, age]
```

Obr. 5.2: Konfiguračný súbor so skráteným zápisom schémy databázy

Riadok 13 vyjadruje kopírovanie všetkých atribútov typu obj. Taktiež na riadku 14 je možné vidieť, že v sekcii **schema** je možné pridať aj ďalšie stĺpce. Zápis na riadku 16 vyjadruje kopírovanie len vybraných atribútov daného typu. Po spracovaní bude štruktúra sekcie **schema** vyzerať nasledovne:

```
schema:
  house_t:
    id: house.id
    rooms: house.count
    owner: "FROM owner_t.id"
  owner_t:
    id: owner.id
    name: owner.name
    age: owner.age
```

5.7 Externý generátor

Rôzne aplikácie môžu mať potrebu generovať špecifické dáta, ktoré nie je možné vyjadriť pomocou poskytnutých typov. To by však v pôvodnej verzii nástroja `dbgenx` znamenalo zásah do zdrojového kódu. Proces pridania nového typu by vyžadoval znalosť implementácie generátora. Riešením tohto problému bude pridanie možnosti definovať vlastný modul. Užívateľom vytvorený modul bude slúžiť ako generátor pre atribút špecifikovaného typu.

Keďže generátor je implementovaný v jazyku python, bol tento jazyk zvolený aj pre užívateľské moduly. Je vyžadované, aby každý modul obsahoval minimálne nasledujúce funkcie:

- `reset` -– funkcia, ktorá uvedie generátor do počiatočného stavu,
- `get` -– funkcia, ktorá vracia vygenerované dáta.

Cesta k priečinku obsahujúcemu súbory s užívateľskými modulmi je špecifikovaná parametrom `--generator PATH`. Definícia vlastného generátora v konfiguračnom súbore je uskutočnená podobným spôsobom ako pre datasety. Do konfiguračného súboru bude pridaná ďalšia nepovinná sekcia **plugins**, ktorá bude obsahovať zoznam názvov súborov s užívateľskými generátormi. Hodnota atribútu typu v sekcii **types** bude meno uvedené v danom zozname. Zápis pre použitie užívateľského modulu pre generovanie IBAN identifikátora vyzerá nasledovne, pričom zvýraznené názvy špecifikujú názov súboru s modulom bez prípony `.py`:

```
plugins:
  - iban

types:
  bank_account:
    id: seq(0, 10000, 1)
    iban: iban
```

5.8 Dátové typy pre aplikácie finančných technológií

Aplikácie finančných technológií sú práve jednou z oblastí, ktorá obsahuje veľkú mieru dát špecifického formátu. Príkladom je medzinárodný formát čísla účtu IBAN. Toto číslo sa skladá z troch častí:

- kód krajiny -– dvojčíselný kód definovaný štandardom ISO 3166-1 alpha-2,
- dve kontrolné číslice – poskytujú ochranu proti chybe čísla účtu,
- BBAN (Basic Bank Account Number) – až 30 znakov identifikujúcich banku a bankový účet, formát aj dĺžka sa líšia podľa štandardu krajiny.

Prísne daná štruktúra a význam dát v spojení s veľkým počtom možností ich formátu činia IBAN náročnou položkou na generovanie. Takýto typ dát nie je možné zapísať pomocou typov, ktoré nástroj `dbgenx` už poskytuje, avšak nie je vhodné ani tento typ pridávať. Aj keď cieľom je umožniť nástroj používať v oblasti finančných technológií, je žiaduce, aby jeho funkcionalita bola profilovaná viac všeobecne. Preto je generovanie dát typických pre oblasť

finančníctva, na ktoré nie je možné použiť existujúce typy, uskutočnené pomocou rozšírenia externého generátora. Vlastné moduly sú vytvorené pre generovanie českého formátu identifikátorov IBAN, IČO a DIČ.

Kapitola 6

Implementačné detaily

V tejto kapitole sú popísané implementačné detaily jednotlivých rozšírení. V prípade, že k danému problému bolo možné pristupovať rôznymi spôsobmi, je odôvodnený výber určitého riešenia.

6.1 Poradie vkladania záznamov do tabuliek

V tejto podkapitole sú predstavené detaily implementácie automatického určovania poradia vkladania tabuliek do databázy z podkapitoly 5.1. V pôvodnej verzii nástroja dbgenx bol generovaný požadovaný počet objektov pre každý **generate** záznam z konfiguračného súboru v poradí, ako boli uvedené. Po vytvorení všetkých objektov náležiacich k danému **generate** záznamu, boli pre tieto objekty generované SQL INSERT príkazy a ich hodnoty boli zároveň uložené do pomocnej premennej pre neskoršie použitie. V prípade, že tabuľka `tab_x` obsahovala referenciu pomocou FROM na tabuľku `tab_y` a typy, ktoré boli na zodpovedajúce tabuľky mapované sú v **generate** sekcii uvedené v poradí `type_x`, `type_y`, boli najskôr generované dáta tabuľky `tab_x`. Pri vytváraní INSERT príkazov bola hodnota cudzieho kľúča vyhľadávaná v pomocnej premennej uchovávajúcej generované dáta, ktorá dáta pre tabuľku `tab_y` ešte neobsahovala a generovanie skončilo výnimkou.

Nová verzia nahrádza funkciu SQL triedy `variables_to_sql` používanú na priebežné generovanie SQL príkazov funkciou `variables_to_data`, ktorá dáta len ukladá. Po dokončení generovania je SQL generované funkciou `generate_sql`, keď sú už dostupné všetky závislosti. Mapa, do ktorej sú dáta počas generovania ukladané má štruktúru zhodnú so sekciou **schema**. Hodnoty jednotlivých riadkov sú ukladané do listu k zodpovedajúcemu kľúču stĺpca. Aj keď je hodnota cudzích kľúčov určovaná na konci generovania, je potrebné uchovávať informáciu o tom, že daný stĺpec obsahuje hodnoty z inej tabuľky. Pre tento účel bola vytvorená trieda `Reference`. Uchováva názov tabuľky a stĺpca, z ktorého bude nadobúdať hodnotu. Objekty tejto triedy sú ukladané spolu s vygenerovanými dátami do zodpovedajúcich stĺpcov.

Pre určenie poradia bol použitý algoritmus topologického triedenia. Pre jeho použitie je však najskôr potrebné vytvoriť orientovaný graf závislostí tabuliek. Závislosť môže byť v sekcii **schema** reprezentovaná buď použitím odkazu FROM, alebo v prípade kompozičného vzťahu obsahuje podriadený záznam primárny kľúč nadriadeného. Pre detekciu druhého typu závislosti je potrebné rozlišovať pri tabuľkách stĺpec s primárnym kľúčom. Označenie stĺpcov reprezentujúcich primárny kľúč je realizované nasledujúcim zápisom:

```
schema:
    user:
        id[PK]: user.id
        name: user.name
```

Zostavenie grafu závislostí prebieha v dvoch cykloch. V prvom prechode sekcie **schema** sú zaznamenané primárne kľúče tabuliek (cieľové uzly hrán grafu). V druhom prechode je zaznamenaná závislosť (hrana grafu) pre tabuľky, ktoré obsahujú primárny kľúč inej tabuľky, alebo záznam FROM. Tento graf je uchovávaný triedou Graph, ktorá tiež obsahuje funkciu na vytvorenie listu uzlov (tabuliek) v topologickom usporiadaní. Usporiadanie bolo implementované pomocou algoritmu založeného na prehľadávaní do hĺbky implementovaného funkciami `get_order` a `top_util` triedy `Graph`.

6.2 Viaceré skupiny obmedzení v sekcii constraints

Táto podkapitola popisuje implementáciu rozšírenia z podkapitoly 5.2. Zadané obmedzenia sú z konfiguračného súboru načítané ako asociatívne pole, ktoré obsahuje skupiny so zoznamom obmedzení. Taktiež sekcia **generate** je načítaná ako asociatívne pole, kde kľúč definuje názov skupiny obmedzení a hodnota je list záznamov typu a počtu objektov, ktoré sa majú pre danú skupinu generovať. Toto pole je ďalej transformované na pole trojíc obsahujúcich typ, počet objektov a názov skupiny obmedzení.

Vo funkcii `generate_data`, bol pozmenený vstup SMT generátora. V pôvodnej verzii bol vstupom list všetkých obmedzení. V upravenej implementácii je to skupina zodpovedajúca skupine daného **generate** záznamu. K tomuto zoznamu sú vždy pripojené obmedzenia zo skupiny *common*, keďže tie platia pre všetky generované objekty. Výraz *common* je teda v rámci názvov skupín obmedzení rezervovaným slovom.

6.3 Generovanie agregovaných závislostí

Táto podkapitola obsahuje implementačné detaily rozšírenia predstaveného v 5.3. Jedným z prvých krokov pri implementácii bolo zváženie formátu zápisu, aký sa použije v konfiguračnom súbore. Definícia musí obsahovať typ agregačnej funkcie, názov tabuľky a stĺpca z ktorého sa bude počítať hodnota a tiež podmienku pre výber riadkov, ktorá sa pri výpočte použije. Tiež bolo potrebné určiť formát zápisu výberovej podmienky. Pôvodný návrh zahŕňal len jednoduchý predikát, napríklad na porovnanie cudzieho kľúča s hodnotou, avšak tento spôsob nepokrýva zložitejšie použitia. V prípade, že by boli možnosti definície podmienky rozšírené tak aby boli dostačujúce, spracovanie tohto reťazca by nadobudlo značnú zložitosť.

Na základe týchto faktorov bolo vybraté iné riešenie. Výsledok funkcie nepočíta samotný generátor, namiesto toho je vytvorený zodpovedajúci SQL príkaz, ktorý slúži na vypočítanie a následné vloženie hodnoty. Užívateľom zadaná podmienka priamo zodpovedá WHERE časti príkazu. Takýmto spôsobom bol generátor odbremenený od spracovania zložitých podmienok a užívateľ má zároveň možnosť utilizovať plnú silu SQL jazyka. Zvolený formát zápisu agregovaných závislostí je `FUNC(table, column, condition)`, kde:

- **FUNC** – jedna z agregačných funkcií SUM, COUNT, AVG, MIN, MAX, ktoré majú rovnakú funkcionalitu ako ich SQL alternatívy,

- **table** – vybraná tabuľka obsahujúca stĺpec,
- **column** – stĺpec, z ktorého bude počítaná hodnota,
- **condition** – podmienka pre výber riadkov pre výpočet, vo formáte SQL WHERE podmienky.

Agregačný vzťah je reprezentovaný objektami triedy `Aggregation`, ktorá uchováva parametre agregácie špecifikované v konfiguračnom súbore a poskytuje funkciu na generovanie zodpovedajúceho SQL update príkazu. Pri vytváraní INSERT príkazov s generovanými dátami je do stĺpcov obsahujúcich agregáciu vložená počiatočná hodnota 0. Príkazy pre agregačné závislosti sú generované nakoniec a pripojené za ostatné INSERT príkazy.

```
types:
  bill:
    id: int
  item:
    price: int

schema:
  bill_t:
    id: bill.id
    sum: SUM(item_t, price, bill_t.id = item_t.bill_id)
  item_t:
    price: item.price
    bill_id: "FROM bill_t.id"

generate:
  - item = 4
  - bill = 1
```

Obr. 6.1: Konfiguračný súbor s agregovanou závislosťou

Na obrázku 6.1 je zobrazené použitie navrhnutého zápisu pre výpočet celkovej sumy účtu na základe jeho položiek. V nasledujúcom výstupe generátora pri použití uvedeného konfiguračného súboru sú farebne vyznačené sekcie zodpovedajúce zápisu:

```
INSERT INTO bill_t (id, sum) VALUES ('0', '0');
INSERT INTO item_t (id, price, bill_id) VALUES ('0', '1', '0');
INSERT INTO item_t (id, price, bill_id) VALUES ('1', '2', '0');
INSERT INTO item_t (id, price, bill_id) VALUES ('2', '3', '0');
INSERT INTO item_t (id, price, bill_id) VALUES ('3', '4', '0');
UPDATE bill_t SET sum = (SELECT SUM(price)
    FROM item_t WHERE bill_t.id = item_t.bill_id);
```

6.4 Štrukturované datasety

Táto podkapitola obsahuje popis implementácie štruktúrovaných datasetov predstavených v 5.4. Dáta zo štrukturovaného datasetu sú odkazované pri jednotlivých atribútoch typu

cez názov stĺpca. Zároveň je potrebné uchovať riadky datasetu. Vzhľadom na tento spôsob prístupu k dátam je teda dataset načítaný ako list asociačných polí, kde prvok listu reprezentuje riadok, kľúčmi asociačného poľa sú názvy stĺpcov a hodnotami samotné dáta. Pre reprezentáciu datasetu bola vytvorená nová trieda `StructuredDataset`. Obsahuje okrem listu s hodnotami aj index, ktorý určuje práve aktívny riadok. Funkcia get potom vracia hodnotu špecifikovaného stĺpca aktuálneho riadku. K zmene riadku má dôjsť po použití záznamu. Keďže je záznam používaný pre viaceré stĺpce a zároveň nemusí byť použitý každý stĺpec, je zmena riadku určená explicitne pomocou funkcie `increment_index`. Táto funkcia ešte obsahuje kontrolu, či bol z daného datasetu načítaný aspoň jeden stĺpec a teda je možné aktuálny záznam považovať za použitý. V opačnom prípade zostáva číslo riadku nezmenené.

Ďalej bol vytvorený nový interný typ `DatasetColumn`. Obsahuje referenciu na štruktúrovaný dataset a názov stĺpca. Táto trieda je používaná ako typ atribútov v objektoch zo sekcie **types**.

Pôvodne boli datasety načítavané počas spracovania sekcie **types**. Keď sa typ atribútu zhodoval s menom datasetu, bol načítaný yaml súbor s daným názvom a podľa jeho typu (jednoduchý, zoradený, obsahujúci bloky) bol ďalej spracovávaný. V prípade, že bol dataset používaný viackrát, musel sa načítať zakaždým. Keďže pri štruktúrovanom datasete je očakávané, že bude použitý viackrát – vždy pre iný stĺpec, bolo by podobné riešenie veľmi neefektívne. Zároveň je však žiaduce zachovať, aby sa dataset načítal až v prípade keď je naozaj použitý ako typ.

Na riešenie problému načítavania a uchovávania datasetov bola vytvorená trieda Datasets. Jej hlavnou funkciou je `get_dataset`, ktorá nahradila predtým používanú funkciu na načítanie datasetov. Táto funkcia buď len vráti už existujúci dataset alebo v prípade, že sa jedná o prvé použitie ho načíta. Samotné načítanie bolo rozšírené, aby podporovalo okrem yaml aj csv súbor. V prípade, že sa jedná o csv, je dataset ukladaný ako StructuredDataset s vyššie spomínanou štruktúrou. Ďalej obsahuje trieda Datasets funkciu `increment_datasets`, ktorá zavolá funkciu na zvýšenie indexu riadku pre každý štrukturovaný dataset. Ostatné datasety nie sú ovplyvnené. K zvýšeniu indexov dochádza v hlavnom cykle generovania dát vždy po vygenerovaní jedného setu objektov, teda jedného záznamu pre každú z generovaných tabuliek.

6.5 Typ text

Táto podkapitola obsahuje implementačné detaily pre rozšírenie 5.5. Pre generovanie náhodného textu bol použitý lorem ipsum generátor [1]. Textový blok je vygenerovaný použitím jeho funkcie `paragraph`. Následne sa kontroluje, či nebola presiahnutá zadaná maximálna dĺžka. V prípade, že k tomu došlo, sú prebytočné znaky useknuté. Táto funkcionalita je zapuzdrená v triede `Paragraph`, ktorá je používaná pre vnútornú reprezentáciu textového typu.

6.6 Skrátenie zápisu schémy databázy

Táto podkapitola obsahuje implementačné detaily pre rozšírenie popísané v podkapitole 5.6. Trieda SQL bola upravená tak, aby prijímala pri inicializácii aj argument obsahujúci typy. Z definície schémy a typov je zostavená výsledná štruktúra tabuliek pomocou funkcie `_get_schema_from_type`. Táto funkcia nahradí všetky stĺpce “*” zodpovedajúcimi stĺp-

cami prevzatými zo špecifikovaného typu. Hodnota stĺpca, teda odkaz na atribút typu, je vytváraná vo formáte `type.attribute`. Podporuje teda len jednu úroveň zanorenia typu a skrátený zápis nie je možné použiť pre vnorené typy ani stĺpce obsahujúce označenie primárneho kľúča.

6.7 Externý generátor

Táto podkapitola obsahuje implementačné detaily rozšírenia predstaveného v podkapitole 5.7. Načítanie externého modulu je uskutočnené pomocou modulu `importlib.util`. Obsahuje objekty umožňujúce konštrukciu importéra, tzn. objektu, ktorý nájde a načíta modul. Konkrétne sú použité funkcie `spec_from_file_location` a `module_from_spec`, ktorými je modul načítaný zo súboru určeného absolútnou cestou a následne pridaný do objektu s načítanými modulmi `sys.modules`.

Pred prvým použitím užívateľského modulu na získanie generovaných dát je vykonaná jeho funkcia `reset`. Funkcia `get` musí zabezpečiť, aby mohla byť správne používaná viackrát za sebou, teda po jej vykonaní musí byť modul v korektnom stave pre ďalšie použitie. Toto umožňuje generovať aj hodnoty, ktoré sú závislé na predošlých vytvorených dátach užívateľského modulu. Po ukončení generovania všetkých dát je modul odstránený.

6.8 Dátové typy pre aplikácie finančných technológií

Táto kapitola obsahuje implementačné detaily rozšírenia z podkapitoly 5.8, teda popis implementácie užívateľských generátorov pre identifikátory IBAN, IČO a DIČ. Každý z modulov obsahuje funkcie `get` a `reset`, tak ako to vyžaduje návrh popísany v podkapitole 5.7. Keďže nie je potrebné moduly nijak explicitne inicializovať, vo všetkých prípadoch funkcia `reset` obsahuje len kľúčové slovo `pass`, ktorého výsledkom nie je žiadna operácia. Funkcia `get` obsahuje logiku generovania popísanú pre jednotlivé identifikátory v nasledujúcich odsekoch.

IBAN

Elektronická forma identifikátoru IBAN Českej Republiky má 24 znakov a sa skladá z kódu krajiny (CZ), kontrolného dvojčíslia, kódu banky a čísla účtu doplneného nulami na 16 číslic. V užívateľskom generátore identifikátoru IBAN implementovanom v rámci tejto práce je kód banky náhodne vybraný zo zoznamu bánk pôsobiacich v ČR. Číslo účtu je generované ako reťazec náhodných číslic dĺžky 16 znakov. Pre verifikáciu ibanu a výpočet kontrolných číslic je použitý algoritmus MOD 97-10. Následne sú všetky časti spojené v správnom poradí a je pridaný kód krajiny.

IČO

IČO je osemmiestné číslo, pričom posledná číslica je kontrolná. Prvá časť je generovaná ako reťazec siedmich náhodných číslic. Kontrolná číslica je definovaná ako zvyšok po delení váženého súčtu jedenástimi. Pre IČO $n_8n_7n_6n_5n_4n_3n_2x$, je hodnota kontrolnej číslice x počítaná ako:

$$x = (11 - (8n_8 + 7n_7 + 6n_6 + 5n_5 + 4n_4 + 3n_3 + 2n_2) \bmod 11) \bmod 10$$

Obsah súboru s implementáciou vlastného modulu slúžiaceho na generovanie českého formátu identifikátora IČO je teda nasledovná:

```
# Module for generating ICO identifier of Czech format
import random

def get():
    ico = [0] * 8
    sum = 0
    for idx in range(0, 6):
        ico[idx] = random.randint(0, 9)
        digit_weight = (8 - idx)
        sum += digit_weight * ico[idx]

    ico[7] = (11 - (sum % 11)) % 10
    return ''.join(map(str, ico))

def reset():
    pass
```

DIČ

Formát DIČ závisí od typu daňového subjektu, teda či sa jedná o fyzickú alebo právnickú osobu. V rámci tejto práce bol implementovaný generátor DIČ pre právnickú osobu. Skladá z kódu krajiny a identifikátora IČO, ktoré je utvorené rovnakým spôsobom ako je popísané pri generátore IČO.

Kapitola 7

Overenie funkcionality automatickými testami

Nástroj už obsahuje automatické jednotkové a integračné testy, vytvorené v rámci diplomovej práce, na ktorú táto práca nadväzuje. Prvým dôležitým krokom realizovaným pri testovaní bola aktualizácia týchto existujúcich testov. Bolo potrebné pôvodné testy upraviť tak, aby zodpovedali novej štruktúre kódu. Tieto testy boli následne použité na regresné testovanie – umožnili overiť, že pridané rozšírenia a refaktorizácia kódu nespôsobili nefunkčnosť pôvodných funkcií. Následne boli pridané testy pre novovzniknuté scenáre.

7.1 Jednotkové testy

Jednotkové testy sú testy najnižšej úrovne. Overujú logiku a funkcionalitu izolovaných softvérových komponentov ako sú funkcie a metódy. Na implementáciu jednotkových testov bol použitý vstavaný python framework unittest. Tento framework umožňuje testy združovať, podporuje automatizáciu testovania, zdieľané nastavenie stavu pred testami a vyčistenie po vykonaní testov. V rámci tejto práce boli pridané testy do existujúcich testovacích prípadov overujúce pozmenenú funkcionalitu. Tiež boli vytvorené nové testovacie prípady, konkrétne pre triedu `Datasets`, ktorá slúži na spracovanie a manipuláciu s datasetmi a `SQL` triedu, keďže v novej verzii obsahuje väčšie množstvo aplikačnej logiky, ktorú je potrebné testovať. Testy sú spúšťané pomocou skriptu `unit_tests_runner.py`, ktorý sa nachádza v priečinku `tests/unit`. V tabuľke 7.1 sú vymenované všetky zmenené a pridané jednotkové testy. Ku každému je uvedený názov testu (názov testovacej funkcie), trieda v ktorej sa nachádza implementácia, rozlíšenie novovytvoreného alebo upraveného testu a odkaz na podkapitolu popisujúcu rozšírenie, ktoré spôsobilo zmeny alebo je testované daným testom. Ostatné testovacie triedy zostali nezmenené.

7.2 Integračné testy

Integračné testy slúžia na testovanie kombinácií viacerých softvérových komponent a ich spolupráce. Testujú rozhrania modulov a ich správanie v prepojení s ďalšími modulmi, prípadne iným systémom, hardvérom alebo operačným systémom. Na integračné testovanie je v nástroji Dataster použitý správaním riadený testovací framework behave. Tento framework používa na definovanie testovacích prípadov jazyk Gherkin a jeho Given-When-Then notáciu pre špecifikáciu požadovaného správania. Integračné testy nástroja zahŕňajú ok-

Test	Trieda	Stav	Rozšírenie
test_parse_input_returns_expected_values	TestCLI	zmena	5.2
test_get_generate_parse_valid_input _into_list_of_2tuples	TestCLI	zmena	5.2
test_get_constraints_parse_input _into_list_3_tuples	TestCLI	zmena	5.2
test_get_constraint_raises_valueerror _on_not_complete_constraint	TestCLI	zmena	5.2
test_get_constraint_raises_valueeror _on_too_long_constraint	TestCLI	zmena	5.2
test_create_schema_from_types_all_attrs	TestSQL	nové	5.6
test_create_schema_from_types _selected_attrs	TestSQL	nové	5.6
test_create_schema_from_types_extra_cols	TestSQL	nové	5.6
test_create_schema_from_types _type_not_exists	TestSQL	nové	5.6
test_get_tables_dependency_graph	TestSQL	nové	5.1
test_get_variables_int	TestVarParser	zmena	5.4, 5.7
test_get_variables_two_int	TestVarParser	zmena	5.4, 5.7
test_get_variables_float	TestVarParser	zmena	5.4, 5.7
test_get_variables_bigint	TestVarParser	zmena	5.4, 5.7
test_get_variables_bool	TestVarParser	zmena	5.4, 5.7
test_get_variables_string	TestVarParser	zmena	5.4, 5.7
test_get_variables_text	TestVarParser	zmena	5.4, 5.7
test_get_variables_interval	TestVarParser	zmena	5.4, 5.7
test_get_variables_sequence	TestVarParser	zmena	5.4, 5.7
test_get_variables_dataset_name	TestVarParser	zmena	5.4, 5.7
test_get_variables_ordered_dataset	TestVarParser	zmena	5.4, 5.7
test_get_variables_dataset	TestVarParser	zmena	5.4, 5.7
test_get_variables_formatted_string	TestVarParser	zmena	5.4, 5.7
test_get_variables_text	TestVarParser	nové	5.5, 5.7
test_get_variables_custom_generator	TestVarParser	nové	5.7
test_parse_input_returns_expected_values	TestVarParser	zmena	5.7
test_paragraph_maximum	TestParagraph	nové	5.5

Tabuľka 7.1: Pridané a modifikované jednotkové testy

rem modulu dbgenx aj nástroj combine. Testy sú spúšťané príkazom python3 -m behave ./tests/integration/features v priečinku comdbgenx.

Integračné testy sú testy vyššej úrovne a majú menšiu závislosť na konkrétnej štruktúre kódu, narozdiel od jednotkových testov. Prispôsobovanie starých testov pre novú rozšírenú verziu nástroja teda spočívalo hlavne v upravení vstupov použitých pre testovanie, tak aby zodpovedali zmenám. Tabuľka 7.2 obsahuje zoznam zmien integračných testov.

Feature	Step	Stav	Rozšírenie
generate_using _combinatioral_extension	start generation required testing data	zmena	5.2, 5.6, 5.7
	start generation required testing data expecting exception	zmena	5.2, 5.6, 5.7
configuration_file	Config file with missing database structure	zmena	5.2
	User generate requirements are missing in config file	zmena	5.2
	There are not specified custom data types in configuration file	zmena	5.2
	There are not specified datasets in valid configuration file	zmena	5.2
	There are all items specified	zmena	5.2
generate_from_dataset	Module run data generation	zmena	5.2, 5.6, 5.7
	Simple dataset specified in include in configuration file	zmena	5.4
	Simple ordered dataset specified in include in configuration file	zmena	5.4
	Structured dataset specified in include and types in configuration file	nové	5.4
	Two rows are selected from structured dataset and generated as SQL command	nové	5.4

Tabuľka 7.2: Pridané a upravené integračné testy

Kapitola 8

Aplikácia riešenia pre reálny systém

Z databázy inštitúcie Roger bola vybraná podmnožina tabuliek tak, aby na nich bola demonštrovaná vytvorená funkcionalita. Z dôvodu zachovania senzitívnych údajov ohľadom presnej štruktúry a povahy dát uchovávaných v databáze, nie je možné uviesť konkrétny príklad konfiguračného súboru. Nasledujúce sekcie preto aspoň približujú oblasti, pre ktoré boli jednotlivé rozšírenia aplikované.

Štruktúrované datasety — vzhľadom na vysoký výskyt väzieb medzi tabuľkami nebolo možné v databáze vybrať vhodnú izolovanú podmnožinu tabuliek, ktorá by mohla byť naplnená pomocou generátora. Tento problém je riešený použitím štruktúrovaných datasetov. Pre jednotlivé rozšírenia bola vybraná skupina tabuliek, na ktorej je najlepšie demonštrované ich použitie a ostatné tabuľky sú naplnené už existujúcimi dátami pochádzajúcimi z pôvodného testovacieho procesu.

Poradie vkladania záznamov do tabuliek — Vďaka tomuto rozšíreniu nebolo potrebné brať do úvahy závislosti medzi tabuľkami a určovať **generate** záznamy v špecifickom poradí.

Viaceré skupiny obmedzení v sekcii constraints – Toto rozšírenie bolo použité pri generovaní testovacích dát pre tabuľku aukcia. Aukcia obsahuje atribút state, ktorý udáva ďalšie hodnoty jej atribútov a tiež atribútov súvisiacich tabuliek. Pre každý stav bola vytvorená osobitná skupina podmienok a následne špecifikovaný žiadaný počet aukcií rôznych typov, ktorý sa má generovať.

Generovanie agregovaných závislostí — Rozšírenie bolo použité pre štatistické záznamy o aukciách. Funkcia SUM bola využitá pre súčet celkovej hodnoty aukcií za určité obdobie. Funkcia AVG bola použitá na definovanie priemerného výnosu.

Skrátenie zápisu schémy databázy — Typy v konfiguračnom súbore boli tvorené na základe databázovej schémy. Štruktúra typov zo sekcie **types** a tabuliek zo sekcie **schema** bola teda identická okrem primárnych kľúčov, cudzích kľúčov a špecifikácie agregačných vzťahov pridaných do sekcie **schema**. Toto rozšírenie bolo teda využité pre každú položku

v sekcii **schema**.

Typ text — Tento typ bol využitý napríklad pri generovaní popisu aukcie, užívateľa, subjektu, a tiež text notifikácie.

Externý generátor – Toto rozšírenie bolo využité pre implementáciu modulov na generovanie dátových typov pre aplikácie finančných technológií.

Dátové typy pre aplikácie finančných technológií – Implementované generovanie identifikátorov IBAN, IČO a DIČ bolo aplikované pre atribúty faktúry, bankového účtu a subjektu.

Kapitola 9

Záver

V tejto práci bol vytvorený nástroj na generovanie testovacích dát pre relačné databázy aplikácií z oblasti finančných technológií. Generátor je implementovaný ako rozšírenie nástroja Dataster. Umožňuje generovať dáta na základe užívateľom definovanej špecifikácie. Nástroj umožňuje generovanie realistických testovacích dát zohľadnením štrukturálnych a dátových závislostí. Umožňuje efektívne zapisovať špecifikáciu generovaných dát a poskytuje možnosť definovať vlastný externý modul pre generovanie špecifických hodnôt. Nástroj je možné použiť aj pre iné oblasti ako je finančníctvo.

Externé užívateľom vytvorené moduly sú oblasťou s vysokým potenciálom pre budúce rozšírenia. Užitočné rozšírenia zahŕňajú napríklad pridanie parametrov do generujúcej funkcie, umožnenie viacerých generujúcich funkcií v jednom rozširujúcom module, ktoré by generovali navzájom súvisiace hodnoty alebo implementáciu kontextu, ktorý by umožnil spoluprácu naprieč externými modulmi. Ďalším možným rozšírením by bola možnosť špecifikovania kardinality vzťahov špecifikovaných výrazom FROM.

Literatúra

[1] *Lorem Text: Dummy lorem ipsum text generator* [online]. [cit. 2022-03-22]. Dostupné z: https://pypi.org/project/lorem-text/.

[2] *PySMT: A library for SMT formulae manipulation and solving* [online]. [cit. 2022-05-01]. Dostupné z: https://github.com/pysmt/pysmt.

[3] *Standard Glossary of Terms used in Software Testing* [online]. 3.1. International Software Testing Qualifications Board [cit. 2022-04-30]. Dostupné z: http://castb.org/wp-content/uploads/2018/09/10-Glossary-All-Terms.pdf.

[4] Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. In: *International Standard ISO: ISO/IEC 25010–2011*. International Organization for Standardization ISO, 2011.

[5] Software and systems engineering – Software testing – Part 3: Test documentation. In: *ISO/IEC/IEEE 29119-3:2021*. 2021.

[6] AfterAcademy Team. *Topological Sorting* [online]. AfterAcademy, máj 2020 [cit. 2022-03-25]. Dostupné z: https://afteracademy.com/blog/topological-sorting#:~:text=Scheduling%20jobs%20from%20given%20dependencies,we%20can%20use%20topological%20sorting.&text=Determining%20the%20order%20of%20compilation,resolving%20symbol%20dependencies%20in%20linkers.

[7] Déharbe, D., Fontaine, P., Guyot, Y. a Voisin, L. Integrating SMT solvers in Rodin. *Science of Computer Programming*. 2014, zv. 94, s. 130–143. DOI: https://doi.org/10.1016/j.scico.2014.04.012. ISSN 0167-6423. Dostupné z: https://www.sciencedirect.com/science/article/pii/S016764231400183X.

[8] Hamilton, T. *What is Functional Testing?* [online]. Guru99, 2022 [cit. 2022-04-06]. Dostupné z: https://www.guru99.com/functional-testing.html.

[9] Hetzel, W. C. *The Complete Guide to Software Testing*. 2. vyd. QED Information Sciences, 1988. ISBN 0894352423.

[10] Kotyz, J. *Nástroj pro tvorbu obsahu databáze pro účely testování software*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/18066/.

[11] Limaye, M. G. *Software Testing*. 2. vyd. Tata McGraw-Hill Education, 2009. 105-106 s. ISBN 978-0-07-013990-9.

[12] Moura, L. de a Bjørner, N. Z3: An Efficient SMT Solver. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 337–340. ISBN 978-3-540-78800-3.

[13] Olsen, K., Posthuma, M. a Ulrich, S. *Certifikovaný tester základnej úrovne* [online]. International Software Testing Qualifications Board, 2019 [cit. 2022-04-30]. Dostupné z: https://castb.org/wp-content/uploads/2020/05/ISTQB_CTFL_Syllabus_SK_2018_3.1-1.pdf.

[14] Piattini, M., Calero, C. a Genero, M. Table Oriented Metrics for Relational Databases. *Software Quality Journal*. Jún 2001, zv. 9, s. 79–97. DOI: 10.1023/A:1016670717863.

[15] Skupina Testos. *Domovská stránka projektu Testos* [online]. FIT VUT v Brně, 2018 [cit. 2022-03-23]. Dostupné z: http://testos.org/.