VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ NA VIZUALIZACI DATABÁZOVÉHO MODELU EXISTUJÍCÍ DATABÁZE

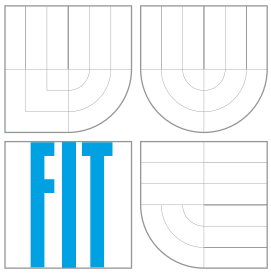
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

IVAN ŠIŠÁK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ NA VIZUALIZACI DATABÁZOVÉHO MODELU EXISTUJÍCÍ DATABÁZE

TOOL FOR DATABASE MODEL VISUALIZATION OF EXISTING DATABASE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

IVAN ŠIŠÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MAROŠ BARABAS

BRNO 2010

Abstrakt

Snaha zobrazovat informace jednodušším způsobem vedla ke vzniku tohoto projektu. V databázových systémech existuje několik informací, které popisují databázový model a jejich textová reprezentace je velmi rozsáhlá a nepřehledná. Vizualizací těchto informací se má dosáhnout názornějšího pohledu na databázový model a jednodušší práce s ním. Nástroj vytvářený v tomhle projektu slouží ke získání informací z databázového systému a následnému zobrazení těchto informácí ve zvoleném editoře anebo prohlížeči. Nástroj je schopný udělat konverzi jednoho souborového formátu na jiný.

Abstract

This project tries to give information to the user in a simpler way. In the database system a lot of information exists which describes the database model. Text representation of this information is bulky and not well-arranged. You can get a much a more vivid view at the database model by using visualization and you can work with it easier. You get information from a database system by using a tool developed in this project and after that you can visualize them in a editor or viewer of your choice. This tool can convert one file format to another.

Klíčová slova

databázový model, vizualizace, Python, Dia diagram editor, ANSI-SPARC architektura, SQL

Keywords

database model, visualization, Python, Dia diagram editor, ANSI-SPARC architecture, SQL

Citace

Ivan Šišák: Nástroj na vizualizaci databázového
modelu existující databáze, bakalářská práce, Brno, FIT VUT v Brně, 2010

Nástroj na vizualizaci databázového modelu existující databáze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Maroša Barabasa

.....
Ivan Šišák
19. května 2010

Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce, Marošovi Barabasovi, za jeho pomoc a podporu. Stejně bych chtěl poděkovat Martinovi Bačovskému z firmy Red Hat za jeho neocenitelné rady při vytváření programové části bakalářské práci. Poděkování patří Alexandře Gálové, která mi pomohla s korekcí a stylizací textu.

Obsah

Kapitola 1

Úvod

V úvode dokumentu sa hovorí o pojme vizualizácie a o využití vizualizácií v dnešnom svete. V ďalších častiach sú načrtnuté ciele, ktoré si kladie tento projekt a v záverečnej časti sú predstavené podobné nástroje ako tento projekt, pri ktorých sú stručne popísané ich výhody a nevýhody.

1.1 Vizualizácia

Vizualizácia nie je žiadnym novým prevratom v živote človeka, pretože jej začiatky pochádzajú už z obdobia, kedy boli pravekí ľudia schopní nakresliť vo svojich jaskyniach alebo v miestach, kde žili, nejaký obraz. V tej dobe zobrazovali činnosti, ktoré vykonávali alebo bohov, ktorých uctievali. Neskôr si ľudia vytvorili obrazy, ktoré reprezentovali jednotlivé časti reálneho sveta a vzniklo tak hieroglyfické písmo[5]. Postupne vznikali nové formy písma vrátane tej šúčasnej, v ktorej jeden znak zastupuje jednu hlásku. Ďalším príkladom vizualizácie sú mapy, ktoré graficky popisujú, ktorým smerom sa vydať, aby sme dosiahli vytúžený cieľ.

Z odborného hľadiska by sa vizualizácia dala definovať ako proces, ktorého cieľom je vytvorenie obrazu, diagramu alebo animácie za účelom podania nejakej informácie alebo súboru informácií[10]. Napríklad takýmto procesom vizualizácie je vytvorenie grafu. Predstavte si, že máte hodnoty teplôt zmeraných každú hodinu za nejaké obdobie. Tieto hodnoty teplôt v definícií predstavujú informácie a ich zobrazením do súradnicového systému vytvoríme graf.

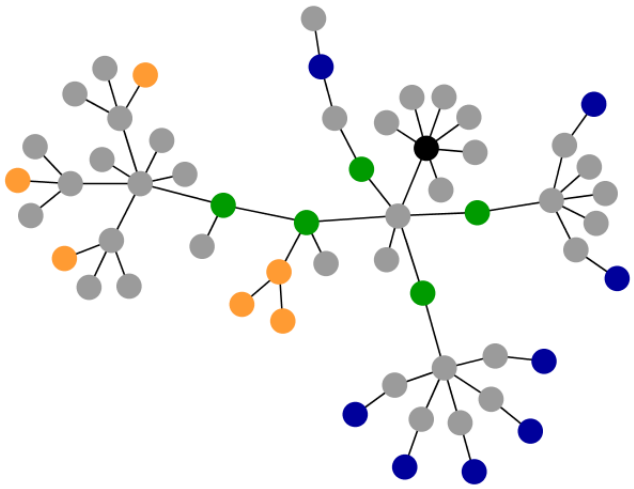
Využitie vizualizácie je dnes veľmi populárne vo všetkých odvetviach vedy a techniky, kde sa rôznymi výpočtami, simuláciami a testami získava veľké množstvo dát. Vizualizácia napomáha tieto dáta analyzovať tým, že ich zobrazí vhodnou formou. Príkladom môžu byť crash testy áut, kde sa najprv simuláciou alebo testovaním zmerajú hodnoty sily, ktoré na auto alebo na osoby v ňom pri zrážke pôsobili a potom sa vytvorí animácia, ktorá zobrazí miesta a intenzitu pôsobiacich síl[7]. Iným využitím vizualizácie je spôsob, pri ktorom sú informácie definované ručne, teda nie sú výsledkom žiadnych testov a simulácií. Tento spôsob sa využíva napríklad pri navrhovaní dizajnu predmetov, kde sa vytvorí technický výkres, na základe ktorého sa potom vytvorí 3D model.

Tým sa dostávame k tomu, prečo sa dnes informácie vizualizujú. Dôvodom je najmä to, že vizualizácia nám poskytuje názornejšie zobrazenie informácie. Predstavme si napríklad, že máme sto hodnôt o teplote nameraných za posledný týždeň. Zobrazenie teplôt v grafe nám pomáha lepšie porozumieť, ako sa teplota menila, aké boli teplotné rozdiely medzi

jednotlivými dňami. Pri sledovaní teplotných zmien nám nevadí ani fakt, že grafická reprezentácia nie je tak presná ako číselná, pretože nepotrebujeme sledovať konkrétnu teplotu, ale len to, či teplota stúpala alebo klesala. Ak by sme však chceli vedieť presnú hodnotu o 7:00 ráno, tak by sme to z grafickej reprezentácie nevedeli povedať presne a v takom prípade nie je táto reprezentácia vhodná. V takom prípade by mohla byť lepšia tabuľková reprezentácia, v ktorej je možné zapísanie presnej hodnoty teploty v požadovanom čase.

Spomenuté boli klady a dôvody, prečo vizualizácie využiť a teraz je namieste hovoriť aj o jej záporoch. Jeden bol v skutočnosti už spomenutý, a tým je nepresnosť. Za ďalší zápor sa považuje fakt, že je nutné mať isté znalosti o zobrazovanej skutočnosti, v opačnom prípade môže dôjsť k nevhodnému pochopeniu informácie.

Pre vysvetlenie by mohol slúžiť nasledujúci obrázok 1.1.



Obrázek 1.1: Príklad vizualizácie

Pokiaľ sa človek pozrie na obrázok bez podania akejkoľvek informácie, tak si môže informáciu reprezentovanú obrázkom domyslieť, ale je veľmi málo pravdepodobné, že bude hneď vedieť, akú informáciu obrázok reprezentuje. Môže si myslieť, že je to napríklad chemická mriežka. Pokiaľ človek obdrží informáciu, že ide o dokument HTML, jeho pohľad na obrázok sa zmení. Pokiaľ sa mu povie, že čierny uzol v grafe znamená značku *BODY*, modré uzly sú odkazy, oranžové sú odstavce, zelené sú oddiely a sivé sú ostatné značky v HTML dokumente, tak užívateľ získa veľmi presnú infomáciu o vzťahoch medzi jednotlivými uzlami v dokumente.

1.2 Ciele projektu

Táto časť je krátkym pohľadom na to, čo sa od projektu očakáva a aj odpoveďou na otázku, prečo projekt vzniká.

Hlavným cieľom projektu je umožniť užívateľovi jednoduchší pohľad na databázový model. Konkrétne by mal byť schopný z databázy získať informácie o reláciách, ich vlastnostiach a vzťahoch, ktoré sú medzi jednotlivými reláciami. Ďalšou jeho schopnosťou by

malo byť vizualizovanie získaných informácií a to napríklad uložením informácií do súboru v takej podobe, ktorej zvolený editor alebo prehliadač rozumie a je schopný zobraziť.

Jednou veľmi dôležitou vlastnosťou je rozširovanie nástroja o podporu pre ďalšie databázové systémy a tiež o podporu pre ďalšie súborové formáty.

1.3 Podobné nástroje

Existuje zopár nástrojov, ktoré sa venujú rovnakej problematike ako tento projekt. Jednu skupinu tvoria komerčné nástroje, akými sú napríklad SQL Studio od Microsoftu alebo Database Visual Architect od Visual Paradigm. Tieto nástroje sú komplexným riešením, čo sa týka návrhu a vizualizácie databázového modelu, avšak sú schopné generovať výstup len sami pre seba. Medzi sebou sú tieto nástroje nekompatibilné a nie je ich možné používať na vzájomnú spoluprácu.

Druhú skupinu tvoria nekomerčné nástroje, ktorých najväčšou chybou alebo skôr mínusom je to, že pracujú iba s konkrétnym databázovým systémom. Ide o nástroje MySQL Workbench, pgDesigner a Postgresql autodoc.

1.4 Štruktúra práce

- Úvod – Kapitola hovorí o dôvodoch, prečo projekt vzniká a aké sú jeho ciele. Stručne sú v nej popísané výhody a nevýhody vizualizácie.
- Databáza – Kapitola poskytuje základný pohľad na databázu a databázový systém, ktorý je nutný pre lepšie pochopenie ostatných častí dokumentu. Popisuje ANSI-SPARC architektúru a vysvetľuje, s akými databázovými modelmi sa možno stretnúť a aké vlastnosti jednotlivé modely popisujú.
- Python – V kapitole je popísaná krátka história programovacieho jazyka Python. Diskutuje sa v tejto kapitole o jednotlivých vlastnostiach, ktorými Python disponuje. Na konci kapitoly je uvedené využitie jazyka.
- Požiadavky – Kapitola zahŕňa informácie o požiadavkách, ktoré boli definované na tento projekt. Hovorí sa v nej o funkčných a nefunkčných požiadavkách a je uvedená analýza týchto požiadaviek a ako výsledok tejto analýzy sú definované prípady užitia.
- Návrh – V sebe zahŕňa informácie o návrhových triedach. Definuje ich názvy, atribúty a metódy, ktoré sa s objektami týchto tried dajú vykonávať. Popísané sú pluginy a rozhranie, ktoré pluginy využívajú pri práci s vnútorným modelom knižnice. Uvedená je v návrhu aj architektúra, ktorá reprezentuje jednotlivé bloky nástroja a ich závislosti medzi sebou.
- Implementácia – V implementácií je uvedený popis jednotlivých častí nástroja z pohľadu „ako“ tieto časti fungujú. Popísaný je URI parser, vstupné a výstupné pluginy.
- Záver – V kapitole sa hovorí o rozšíreniach, ktoré by mohli byť v budúcnosti implementované. Zhrnuté sú vlastnosti, ktorými nástroj v aktuálnom stave disponuje.

Kapitola 2

Databáza

Dáta a práca s nimi je dnes neodmysliteľnou súčasťou každého výpočtového systému. Každú chvíľu dochádza k spracovaniu veľkého množstva dát. Je nutné aby tieto dáta boli efektívne uložené, aby k nim bolo možné jednoducho pristupovať a aby bola práca s nimi rýchla. Tieto vlastnosti sa považujú za tie najdôležitejšie a má ich každý dobrý systém pre riadenie bázy dát. Dnes sa často hovorí o jednej vlastnosti a tou je bezpečnosť a zabezpečenie dát. Táto vlastnosť sa dosahuje mechanizmami, ktoré sa dovoľujú pripojiť len tým užívateľom, ktorí majú na to právo. Otázku zabezpečenia dát riešia nástroje, ktoré sú schopné šifrovať a dešifrovať dáta.

V tejto kapitole sú objasnené pojmy ako dáta, databáza a tiež ANSI - SPARC architektúra, ktorá nám poskytuje rôzne uhly pohľadu na databázu.

2.1 Dáta, databáza a databázový model

Dáta sú reprezentáciou akejkoľvek skutočnosti. Je ich možné prenášať, uchovávať, interpretovať alebo spracovávať[11]. Ak hovoríme o dátach v súvislosti s počítačmi, tak hovoríme o hodnotách, ktoré sú určitého dátového typu. Databázy zabezpečujú trvalé uloženie dát.

Databáza je organizovaná kolekcia súvisiacich dát. To akým spôsobom sú dáta organizované nám podmieňuje spôsob a rýchlosť neskoršieho získavania týchto dát z databázy. Akým spôsobom sú dáta uložené nám popisuje tzv. dátový model.

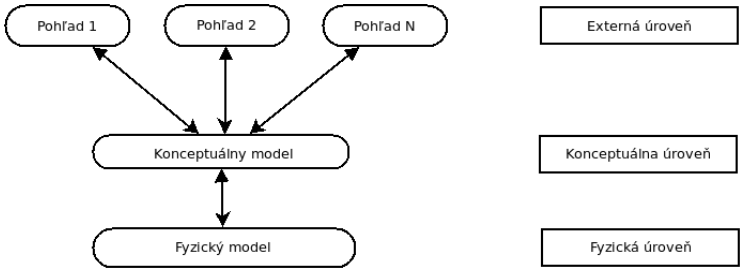
Databázový model (databázová schéma) je štruktúra alebo formát v akom sú jednotlivé dáta usporiadané. Model sa väčšinou popisuje formálnym jazykom, ktorému systém riadenia bázy dát rozumie. Najčastejšie je to jazyk SQL, ktorý je štandardom, ale je bežnou záležitosťou, že databázové systémy si tento jazyk prispôsobia podľa potrieb. Napríklad MySQL používa pre pomenovanie číselného dátového typu, ktorý má veľkosť štyri bajty názov *INT* a podľa štandardu by mal byť tento názov *INTEGER*. Pre každú databázu sa definuje špecifický model, ktorý je vytvorený tak, aby čo najviac vyhovoval účelu, na ktorý bude použitý.

V nasledujúcej časti bude predstavená ANSI - SPARC[1] architektúra databáz, ktorá nám objasní rôzne uhly pohľadu na databázu a pomôže nám aj k detailnejšiemu objasneniu pojmu databázový model.

[1] American National Standards Institute, Standards Planning And Requirements Committee

2.2 ANSI - SPARC Architektúra

ANSI - SPARC architektúra[8] je štandard pre návrh systému riadenia bázy dát. Prvý krát bola predstavená v roku 1975 a odvtedy sa stala štandardom pri návrhu databázových systémov. Napriek svojej dlhej minulosti nebol tento štandard nikdy formálne popísaný.



Obrázek 2.1: Úrovne ANSI - SPARC architektúry

Táto architektúra sa snaží vytvoriť tri úrovne abstrakcie nad dátami. Snaží sa oddeliť skutočné uloženie dát v hardvéri od práce s dátami (v zmysle pridávania, mazania a editácie dát). Rozloženie jednotlivých vrstiev ANSI - SPARC architektúry je vidieť na obrázku 2.1. Jej hlavné črty sú zhrnuté v nasledujúcich bodoch:

- Vytváranie pohľadov – pohľad slúži na sprístupnenie len tých dát, ktoré sú pre istú skupinu užívateľov podstatné a prípustné.
- Schovať detaily fyzického uloženia dát – užívateľ nepotrebuje vedieť, akým spôsobom budú dáta na fyzickom médiu uložené.
- Zmena fyzického uloženia nemá vplyv na zobrazenie dát užívateľovi – ak by administrátor zmenil spôsob fyzického uloženia dát (pridal by napríklad index nad nejakou skupinou dát kvôli zrýchleniu prístupu), nemalo by to ovplyvniť zobrazenie dát užívateľovi
- Zmena hardvéru nemá vplyv na štruktúru uloženia dát – príkladom môže byť výmena pevného disku alebo počítača, na ktorom beží databázový server.
- Zmena logickej reprezentácie dát nemá vplyv na zobrazenie dát užívateľovi – pridanie, zmena alebo odstránenie dátových štruktúr sa nesmie vážne dotknúť zobrazenia dát užívateľovi.

Architektúra sa skladá z troch úrovní, ktoré sú popísané pomocou tzv. schém alebo modelov. Tie popisujú štruktúru alebo formát dát na každej úrovni architektúry.

2.2.1 Fyzická úroveň

Fyzická úroveň zahrňuje skutočnú reprezentáciu dát v systéme, štruktúru uloženia dát v databáze a v počítači, definície pre uloženie jednotlivých záznamov, metódy reprezentácie dát, dátových polí a indexov. Pre každý databázový systém je fyzická schéma len jedna a vytvárajú ju programátori databázových systémov. Medzi základné modely fyzickej úrovne ANSI - SPARC architektúry patria:

- „Flat file“ model – dáta sú ukladané do dvojrozmerného poľa (tabuľky). Predpokladá sa, že hodnoty v stĺpcoch tabuľky sú rovnakého typu a jednotlivé hodnoty na riadku majú medzi sebou nejaký vzťah.
- Hierarchický model – dáta sú organizované v stromovej štruktúre. Pri tejto štruktúre je možné modelevanie vzťahov 1:N. Nevýhodou je redundancia jednotlivých hodnôt.
- Sieťový model – vychádza z hierarchického modelu, ktorý rozširuje o možnosť, že každá položka nachádzajúca sa v stromovej štruktúre môže mať viacerých predkov a viacerých nasledníkov. Výsledkom je odstránenie redundancie z databázy.
- Relačný model – vychádza z teórie množín, pre ktorú sú zakladné termíny relácia, atribút a doména. Reprezentáciou relácie v implementácii databázového systému sú tabuľky, ktoré zastupujú istú entitu v skutočnom svete. Jednotlivé stĺpce tabuľky sú v teórii množín atribútmi a sú vyjadrením istej vlastnosti alebo stavu entity v reálnom svete. Množina hodnôt, ktorá sa v stĺpci môže vyskytovať sa nazýva doména.
- Objektovo - orientovaný model – snaží sa o to aby reprezentácia dát v databáze bola čo najpríbuznejšia s reprezentáciou dát v aplikačnom programe.

2.2.2 Konceptuálna úroveň

Na tejto úrovni dochádza k popisu dát, ktoré budú v databáze uložené. Úroveň zahŕňa aj vzťahy medzi definovanými dátami. Nešpecifikuje, akým spôsobom budú dáta fyzicky uložené, ale zahŕňa definíciu, aký dátový typ bude priradený k dátam, aké vťahy budú medzi týmito dátami a tiež aké integritné obmedzenia sú kladené na jednotlivé dáta. Nad týmto modelom má administrátor databázy najväčšiu kontrolu. Definuje dáta a vzťahy, taktiež definuje užívateľské pohľady. Každá databáza má jednu takúto schému. Táto práca sa venuje najmä tejto úrovni ANSI - SPARC architektúry.

Pri vytváraní konceptuálnej schémy sa používa E-R modelovanie[2]. Pod pojmom E-R modelovanie sa rozumie proces, ktorý prebieha najmä na začiatku vytvárania informačných systémov a jeho výsledkom je popis informácií, ktoré sa majú v databáze ukladať. Grafickým výstupom je tzv. E-R diagram, v ktorom sú popísané jednotlivé entity, ich atribúty a vzťahy medzi jednotlivými entitami.

2.2.3 Externá úroveň

Sem patria užívateľské pohľady, ktoré istej skupine užívateľov sprístupňujú tie informácie, ktoré sú pre nich podstatné. Pomocou pohľadov sa zamedzuje aj tomu, aby pred užívateľom, ktorý nemá oprávnenie k prezeraniu istých dát, boli tieto dáta skryté. Schém popisujúcich užívateľské pohľady môže byť niekoľko a väčšinou sú ich definície obrazom toho, akej skupine užívateľov sú dáta sprístupňované.

2.3 Jazyk SQL

Jazyk SQL[3] vznikol v 70. rokoch 20. storočia. Pôvodne sa jazyk volal SEQUEL a bol určený pre vytváranie a získavanie dát. Za vznikom jazyka SQL[6] stoja dvaja muži: Donald D.

[2] Entity - relationship modeling
[3] Structured query language

Chamberlin a Raymond F. Boyce. Bol určený pre databázový systém od spoločnosti IBM, nazývaný *System R*. Neskôr v 70. rokoch prišla na trh firma, ktorá si osvojila koncept relačných databáz a tiež koncept jazyka SQL. Táto firma, dnešný Oracle, vytvorila prvý komerčný systém pre riadenie bázy dát. Prvými zákazníkmi boli úrady americkej vlády, CIA[4] a aj americká armáda. Tento jazyk je štandardizovaný a nachádza sa vo väčšine databázových systémov.

Jazyk má tri podmnožiny a každá podmnožina plní inú funkciu pri modelovaní databázového modelu. Tieto funkcie a s nimi spojené konštrukcie jazyka sú rozobrané v nasledujúcich častiach dokumentu.

2.3.1 Jazyk pre definíciu dát – DDL

Táto podmnožina jazyka slúži na vytvorenie konceptuálneho modelu databázového systému. Pomocou nej definujeme, aké relácie a aké vťahy medzi reláciami sa budú vyskytovať v databáze. Prípadne, môžeme definovať vlastnosti, ktoré ovplyvňujú fyzické uloženie dát. Jednou možnosťou je vytvorenie indexu nad istou množinou hodnôt. Tu sa využívajú nasledovné konštrukcie jazyka:

- CREATE – slúži na vytvorenie databázových objektov, akými sú tabuľky, pohľady, funkcie, menné priestory, indexy a iné.
- ALTER – slúži na vykonávanie zmien v jednotlivých objektoch. Typickými sú zmeny názvu, dátového typu, obmedzení a podobne.
- DROP – slúži na zmazanie databázových objektov.

2.3.2 Jazyk pre manipuláciu dát – DML

Táto podmnožina jazyka je oveľa častejšie používaná, keďže definícia objektov v databázovom systéme a ich zmeny sú vykonávané väčšinou na začiatku a veľmi zriedkavo počas prevádzky databázy. Prístup k dátam, pridávanie dát, editácia a prípadné odstraňovanie sú častejšími operáciami, ktoré sa vykonávajú v databáze. Tento jazyk využíva nasledujúce konštrukcie jazyka:

- SELECT – slúži na výber dát z tabuliek, pohľadov, rôznych spojení jednotlivých objektov. Pomocou klauzuly WHERE je možné vybrať tie riadky tabuľky, ktoré spĺňajú určitú podmienku. Tá sa špecifikuje za spomínanou klauzulou. Uvedením klauzuly ORDER BY dôjde k zoradeniu výsledku podľa kľúča, ktorý je špecifikovaný bezprostredne za klauzulou.
- INSERT – slúži na vkladanie nových dát do tabuliek.
- UPDATE – slúži na zmenu hodnôt konkrétnych atribútov v tabuľke.
- DELETE – slúži na zmazanie dát z tabuľky.

[4] Central intelligence agency

2.3.3 Jazyk pre riadenie prístupu k dátam – DCL

Množina slúži na vytváranie alebo rušenie autorizačných pravidiel pre užívateľov. Využíva konštrukcie:

- GRANT – slúži k povoleniu vykonávania istej operácie a
- REVOKE – slúži k zakázaniu vykonávania istej operácie,

pomocou ktorých sa zabraňuje v prístupe užívateľov k dátam, ktoré nie sú pre nich určené.

Kapitola 3

Python

Výber programovacieho jazyka má veľký vplyv na neskorší úspech softvéru, pretože nie každý jazyk sa hodí pre každý typ úloh. Ak chcem vyvíjať webovú aplikáciu, tak vhodným použitím je napríklad PHP. V prípade vytvárania desktopových aplikácií je vhodné použiť Javu, C++ a iné. Ďalšie faktory ovplyvňujúce výber jazyka môžu byť čas vývoja, požadovaná rýchlosť aplikácie alebo použitá platforma a iné.

V tomto projekte je použitý programovací jazyk Python, pretože to je jednou z požiadaviek, ktoré sú kladené na projekt a vďaka istej úrovni abstrakcie nad dátovými typmi a štruktúrami je v ňom vývoj softvéru rýchlejší. Dokonca podľa autora Pythona Guida van Rossuma je vývoj rýchlejší 5-krát[13] .

Nasledujúce časti tejto kapitoly popisujú tento jazyk, jeho históriu a vlastnosti, ktorými disponuje.

3.1 História

Tento jazyk sa začal formulovať v neskorých 80. rokoch 20. storočia a jeho vývoj začal v decembri 1989[12]. Jeho autorom je Holanďan Guido van Rossum. Hlavnou myšlienkou bolo vytvorenie mechanizmu pre správu výnimiek a vytvorenie rozhrania pre komunikáciu s vtedajšími operačnými systémami nazývanými Amoeba. Len pre zaujímavosť, názov tohto jazyka je odvodený od seriálu, ktorý bol vtedy vysielaný, s názvom *Lietajúci cirkus Montyho Pythona*.

V roku 1991 uzrela svetlo sveta jeho prvá verzia pod označením 0.9.0 . Verzia obsahovala základné rysy ako triedy, dedičnosti, systém na odchytávanie výnimiek, funkcie a základné dátové štruktúry *list, dict, str*. Jazyk bol už od jeho začiatku modulárny a v tejto verzii bol správca modulov požičaný z jazyka Modula-3.

V januári roku 1994 vychádza nová verzia Python 1.0, ktorá bola obohatená o nové prvky z oblasti funkcionálneho programovania (pridané funkcie *map, reduce, lambda* a *filter*). V ďalšej verzii s označením Python 1.4 boli pridané nové vlastnosti a to *keyword argumenty*[1] vo funkciách alebo počítanie s komplexnými číslami.

Vo verzii Python 2.0 prichádza s veľmi významnou novinkou nazvanou *garbage collector*, ktorú však vývojári Pythona nepoužili ako prví. Autorom *garbage collectoru* je John McCarthy, ktorý ho použil v roku 1959 pre jazyk Lisp. Ide o správcu pamäte, ktorý uvoľnuje

[1] argument funkcie, ktorému sa hodnota nepredá na základe pozície v definícii funkcie, ale pomocou priradenia hodnoty pri volaní funkcie

zabrané pamäťové miesta, ktoré už nie sú pre program potrebné. V prvom kroku vyhľadá miesta, ku ktorým sa už nedá z programu pristúpiť a v druhom kroku tieto miesta uvoľní.

Počas vývoja jazyka dochádzalo k vytváraniu nových a občas aj duplicitných spôsobov pre vykonanie rovnakej úlohy. Keďže hlavnou myšlienkou Pythona je poskytnutie len jedného spôsobu ako vyriešiť istú úlohu, tak verzia Python 3.0 bola navrhnutá tak, aby sa tieto duplicity odstránili z jazyka. Pri vytváraní verzie 3.0 neexistovala požiadavka, ktorá by hovorila o spätnej kompatibilite s verziou 2.x. Základnými zmenami v Pythone 3.0 sú napríklad príkaz print, ktorý sa zmenil na funkciu alebo využitie štandardu unicode pre všetky textové reťazce.

3.2 Programovacie paradigma

Existuje niekoľko programovacích paradigiem, no väčšinou programovacie jazyky využívajú len jednu. Napríklad jazyk C využíva štrukturálne programovanie, jazyk LISP zase funkcionálne. Python je jeden z mála jazykov, v ktorom sa dá využívať viacero spôsobov programovania. Tým základným je objektovo - orientované programovanie, v ktorom sú dáta a funkcie, ktoré s nimi pracujú, združené do jedného celku - objektu. Takémuto združeniu sa hovorí zapúzdrenie a poskytuje nám vytvorenie istej úrovne abstrakcie nad reprezentovanou skutočnosťou. Objekty sa tvária ako čierne skrinky, pri ktorých programátor nemusí vidieť do vnútra a vedieť všetky detaily ale stačí mu vedieť, čo sa s daným objektom dá robiť.

Narozdiel od objektovo-orietovaného programovania sú v štrukturálnom programovaní dáta samostatne uložené a pracujú s nimi samostatné funkcie, ktoré majú jeden vstup a jeden výstup. V Pythone je možné využívať aj tento spôsob programovania.

V každom programovacom jazyku sa líši implementácia objektovo-orientovaného programovania. Python podporuje napríklad viacnásobnú dedičnosť alebo preťaženie operátorov, avšak nepodporuje riadenie prístupu k jednotlivým členom objektu. To znamená, že každý člen objektu je verejný a len pomocou *mangling metody*[2] je ho možné skryť.

Použitie objektovo-orientovaného programovania je vhodné až pri väčších projektoch. Pri malých projektoch je vytváranie nových tried a ich metód skôr záležitosťou, ktorá zdržuje programátora, preto je v malých projektoch vhodnejšie využiť štrukturálne programovanie[9].

Len pre úplnosť je nutné dodať, že Python obsahuje aj niektoré konštrukcie jazyka, ktoré sú známe z funkcionálneho programovania.

3.3 Interpretovaný jazyk

Interpretované jazyky sú často považované za jazyky, ktoré sú pomalšie oproti tým kompilovaným. Pri kompilovaných sa zdrojový kód preloží do inštrukcií procesora.

Interpretované jazyky prekladajú zdrojový kód do tzv. bajtového kódu a tento kód je potom vykonávaný pomocou interpreta. Interpret je väčšinou kompilovaný program, ktorý inštrukcie v bajtovom kóde prevádza na inštrukcie procesora. Pri každom spustení takéhoto kódu dochádza k prekladu do bajtovej podoby a to spôsobuje spomalenie celého programu.

Python pre zvýšenie výkonu v tomto smere robí to, že automaticky prekladá zdrojový kód do bajtového kódu a tento bajtový kód si uloží do súboru. Pri ďalšom spustení programu

[2] názov člena objektu začína podtržníkom

už nedochádza k prekladu do bajtového kódu (pokiaľ nebol zdrojový kód zmenený), ale je spúšťaný bajtový kód zo súboru, ktorý bol predtým vytvorený.

3.4 Typovanie

Python využíva silné dynamické typovanie. Dynamické typovanie znamená, že dátový typ premennej je odvodený od priradenej hodnoty. Výhodou tohto príncípu je jednoduchšie programovanie, pretože premenná môže nadobúdať počas behu programu rôzne dátové typy. Nevýhodou je nemožnosť optimalizovať program na výkon.

Silné typovanie znamená, že je možné kontrolovať dátový typ premenných pri behu programu a je možné kontrolovanie ich chybného použitia.

3.5 Syntax

Tvorcovia Pythona považujú túto časť jazyka za veľmi dôležitú a venujú jej veľké úsilie. Snažia sa o vytvorenie jazyka s čo najjednoduchšou a priamočiarou syntaxou. Tiež sa snažili do syntaxe vložiť, čo najmenej konštrukcií, ktoré by zvádzali programátora k chybám. Jeden z veľmi dobrých znakov syntaxe je odsadenie. Telo funkcie je odsadené od definície funkcie, rovnako aj blokové príkazy v cykloch a podmienkach sú odsadené od výrazu porovnania. To opäť napomáha k zamedzeniu tvorby chýb. Napríklad v jazyku C je niekedy hľadanie zloženej zátvorky veľmi zdĺhavé a programátora stojí veľa nervov. Ďalším znakom syntaxe je minimalizmus, ktorý predstavuje malé množstvo konštrukcií pre podporu riadenia toku programu. V Pythone existuje jedna konštrukcia pre podmienené vykonávanie kódu, jedna pre konečný počet cyklov, jedna pre podmienený počet cyklov.

3.6 Prenositeľnosť

Python má veľmi dobrú prenositeľnosť programov a je možné ich spúšťať na rôznych hardvérových architektúrach a pod rôznymi operačnými systémami. Podporované operačné systémy sú Microsoft Windows, Unixové systémy vrátane Linuxu, Mac OS X a mobilné platformy (WindowsCE, Symbian60, Maemo).

Väčšina programov nie je napísaná čisto v Pythone a často používajú rôzne doinštalované knižnice, ktoré nie sú úplne prenositeľné. Teda je možné, že program bude fungovať v systéme Microsoft Windows, ale pri použití v Linuxe už nebude funkčný alebo bude jeho funkcionalita obmedzená. Napríklad štandardná knižnica obsahuje niektoré moduly, ktoré sú použiteľné len pre istý operačný systém. Ide najmä o systémovo špecifické veci, napríklad modul pre prácu s registrami v Microsoft Windows je nemožné použiť pre Linux. Dostupnosť modulov v jednotlivých operačných systémoch je popísaná v dokumentácií Pythonu[3].

3.7 Využitie

Python sa stal neoddeliteľnou súčasťou veľkého množstva operačných systémov. Často sa využíva na definovanie komunikačného rozhrania medzi systémovými volaniami a knižnicami. Tiež sa využíva ako doplnok pri programovaní aplikácií, ktoré majú mať programovateľné rozhranie.

Python v rámci štandardnej knižnice podporuje prácu s reťazcami a súbormi. Podporuje prvky kryptografie a niekoľko algoritmov v tejto oblasti, internetové protokoly (HTTP, IMAP, POP, FTP, SMTP atď.), oblasti softvérového inžinierstva (unit testing, logging, profiling a ďalšie), prácu s multimédiami a vytváranie užívateľských rozhraní.

Kapitola 4

Požiadavky

Vytvorenie požiadaviek je prvým krokom pri vývoji projektu. Bez definície požiadaviek nie je možné vyvíjať akýkoľvek projekt, pretože v skutočnosti neexistuje žiadna predstava o tom, čo by mal projekt robiť, kto by ho mal využívať a podobne.

Požiadavky možno rozdeliť do dvoch skupín:

- Funkčné požiadavky – definujú funkcionalitu systému. Predstavte si, že máte ako projekt správcu užívateľov, tak jeho funkčnými požiadavkami môže byť pridávanie, zrušenie a autorizovanie užívateľov.
- Nefunkčné požiadavky – definujú vlastnosti alebo obmedzujúce podmienky systému. V prípade správcu užívateľov môže byť nefunkčnými požiadavkami obmedzenie počtu autorizovaných užívateľov v istom časovom období, správca má byť multiplatformový a podobne.

Do týchto dvoch skupín budú rozdelené aj požiadavky pre tento projekt.

V tejto súvislosti stojí za zmienku, že zadávateľom tohto projektu bola firma Red Hat zastupovaná Martinom Bačovským, ktorý definoval všetky požiadavky na projekt.

V nasledujúcich častiach tejto kapitoly budú definované funkčné a nefunkčné požiadavky.

4.1 Funkčné požiadavky

Funkčné požiadavky popisujú správanie budúcej aplikácie. Neskôr sú využité pri testovaní aj na spätnú kontrolu, či sa aplikácia správa tak, ako bolo požadované. Na tento projekt sú kladené nasledujúce požiadavky.

4.1.1 Načítanie informácií z databázy

Prvou požiadavkou na tento projekt je schopnosť načítať informácie o tabuľkách a to názov, popis, názov schémy, do ktorej tabuľka patrí. Každá tabuľka obsahuje stĺpce a o nich by sa mali získať informácie akými sú názov, dátový typ, popis, implicitná hodnota. Ďalšie informácie, ktoré sa majú z databázy získať sú integritné obmedzenia pre jednotlivé tabuľky a ich atribúty. Medzi tieto integritné obmedzenia patria nutnosť uvedenia hodnoty atribútu tabuľky, primárny kľúč, unikátna hodnota a obmedzenie hodnoty podľa zadaného výrazu. Poslednou informáciou, ktorá sa má z databázy získavať je tzv. cudzí kľúč, ktorým sa odkazuje do inej tabuľky a na základe týchto kľúčov dochádza ku spájaniu tabuliek.

4.1.2 Vizualizácia informácií

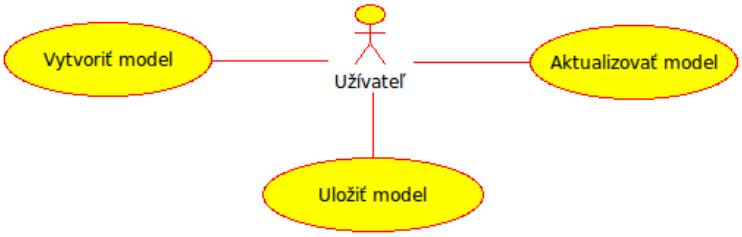
Načítané informácie z databázy je nutné zobraziť vo vhodnej forme, tak aby užívateľ získal čo najlepší pohľad na databázu, využitím niektorého z dostupných editorov alebo prehliadačov.

4.1.3 Aktualizovanie modelu

Pri aktualizácii databázového modelu je nutné zabezpečiť, aby formátovacie informácie, akými sú napríklad pozícia tabuľky v obraze, farba a podobne zostávali zachované a boli doplnené o aktuálne informácie z databázy. Modelová situácia by mohla vyzerať nasledovne. Tabuľka je zobrazená pri vizualizácii vľavo dole a má červenú farbu a obsahuje päť stĺpcov. V databáze sa do tabuľky vložil nový stĺpec. Po aktualizácii databázového modelu musí mať tabuľka červenú farbu, musí byť vľavo dole a musí obsahovať šesť stĺpcov.

4.2 Prípady užitia

Tento projekt nezahrňuje rozličnú funkcionalitu pre viacero osôb a preto diagram prípadov užitia na obrázku 4.1 definuje len jedného aktéra, ktorý je pomenovaný ako *užívateľ*.



Obrázek 4.1: Diagram prípadov užitia

Prvým prípadom použitia projektu užívateľom je vytvorenie modelu, ktorý je v diagrame zaznačený ako prípad *Vytvoriť model*. Tento prípad užitia zastupuje požiadavku načítania informácií z databázy.

Ďalšiu funkcionalitu, ktorú projekt poskytuje je uloženie informácií z databázy do súboru, pomocou ktorého sa môže databázový model zobraziť vo zvolenom prehliadači. Tento prípad je v diagrame označený ako *Uložiť model*. Prípad reprezentuje požiadavku vizualizácie databázového modelu a uloženia do súboru vo zvolenom formáte, ktorému prehliadač rozumie.

Treťou možnosťou použitia projektu užívateľom je aktualizovanie už existujúceho modelu, ktorý sa nachádza vo zvolenom súbore. Tento prípad je v diagrame označený ako *Aktualizovať model*. Tento prípad môže slúžiť aj na konverziu medzi jednotlivými súborovými formátmi, pri ktorej dôjde k aktualizácii informácií o tabuľkách a obmedzeniach.

4.3 Nefunkčné požiadavky

Nefunkčnými požiadavkami sa rozumie špecifikácia pre vlastnosti systému a obmedzenia, ktoré systém musí spĺňať. V tomto projekte bolo špecifikovaných niekoľko nefunkčných požiadaviek, ktoré budú prediskutované v nasledujúcom texte.

4.3.1 Databázové systémy

Už viackrát bolo spomenuté, že by projekt mal byť rozšíriteľný. Jedným z možných rozšírení by mali byť databázové systémy. Pre tento projekt bol požadovaný hlavne databázový systém s názvom PostgreSQL.

V prípade neskoršieho vývoja projektu by sa mohli pridať systémy SQLite, MySQL, Oracle database, Microsoft SQL server.

4.3.2 Programovací jazyk

Programovací jazyk Python 2.5 bol požiadavkou na implementačný jazyk najmä kvôli podpore v operačných systémoch Fedora a Red Hat Enterprise Linux, ale aj kvôli podpore API so spomínaným databázovým systémom PostgreSQL a podpore pre prácu s XML dokumentami.

4.3.3 Formáty súborov

Výstupné formáty súborov slúžia na uloženie informácií získaných z databázy a ich následnú vizualizáciu vo zvolenom editore. Jedným z požadovaných výstupných formátov je formát vhodný pre zobrazenie v *Dia – Diagram Editor*. Ďalším formátom má byť vlastným spôsobom štruktúrovaný XML súbor.

Pri rozširovaní knižnice by sa mohli doplniť ďalšie formáty súborov ako napríklad SVG, PDF, DocBook a iné.

4.3.4 Identifikácia zdrojov pomocou URI

Pre prístup do databázového systému je nutné, identifikovať umiestnenie a zadanie napríklad mena, hesla alebo čísla portu, na ktorom požadovaný databázový systém očakáva spojenia. Pre túto identifikáciu bolo požadované použitie URI[1].

V prípade databáz by mal mať tento identifikátor tvar:

```
<protokol>://<užívateľ>:<heslo>@<umiestnenie>:<port>/<databáza>/<schéma>
```

Taktiež je nutné identifikovať výstupný súbor alebo v prípade aktualizácie aj súbor vstupný. V tomto prípade by mal mať URI tvar:

```
file://<cesta k súboru>
```

[1] Unified resource identifier - jednotný identifikátor zdroja

Kapitola 5

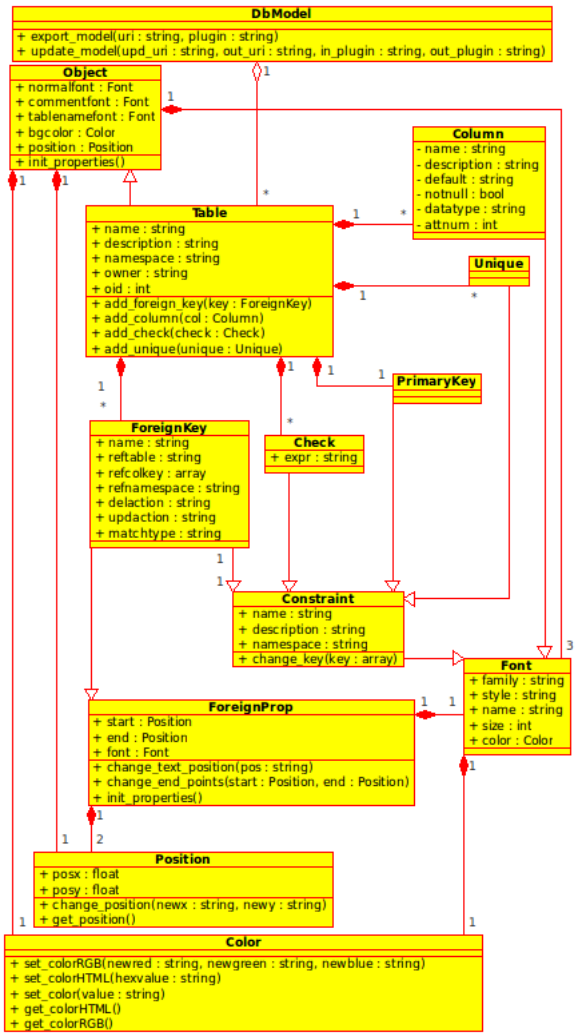
Návrh

Táto časť sa venuje ďalšej etape pri vývoji softvérového produktu a tou je etapa návrhu, ktorej hlavným cieľom je vytvorenie diagramu tried a definícia rozhrania. Kapitola popisuje práve tieto spomínané ciele. V úvode je predstavený diagram a popis jednotlivých tried. V druhej časti je predstavené rozhranie, pomocou ktorého jednotlivé časti nástroja spolu komunikujú.

5.1 Návrh tried

V tejto časti budú rozoberané jednotlivé triedy, ktoré budú použité pri implementácii. Pri každej triede bude uvedený krátky popis toho, ktorú entitu v reálnom svete reprezentuje, popis atribútov, ktoré sa v nej nachádzajú a metódy, ktoré je možné s objektom danej triedy vykonávať. Diagram týchto tried a vzťahy medzi nimi sú na obrázku 5.1. Jednotlivé triedy, ktoré sa vyskytujú v diagrame:

- **DbModel** – trieda je vnútornou reprezentáciou databázového modelu. Zahrňuje zoznam odkazov na tabuľky, ktoré sa v databáze vyskytujú. Obsahuje dve metódy:
  - *export_model* – podľa vstupného argumentu s názvom *plugin* sa zvolí, do akého formátu sa budú transformovať informácie obsiahnuté vo vnútornom modeli. Po vytvorení zvoleného formátu obsahu sa tento obsah zapíše do súboru identifikovaného argumentom s názvom uri, ktorý je URI identifikátorom.
  - *update_model* – táto metóda slúži na aktualizáciu už existujúceho súboru. Podľa *in_plugin* a *out_plugin* sa identifikuje o aký vstupný a výstupný formát má užívateľ záujem. Ďalšie dva argumenty slúžia na identifikovanie súboru, z ktorého sa budú načítavať formátovacie informácie a na identifikáciu súboru, do ktorého sa zapíše výsledok.
- **Table** – trieda zastupuje z databázového modelu tabuľku a jej vlastnosti:
  - *name* – názov tabuľky,
  - *description* – popis tabuľky,
  - *owner* – vlastník tabuľky, najčastejšie osoba, ktorá tabuľku vytvorila,
  - *namespace* – schéma, do ktorej tabuľka patrí a
  - *oid* – jednoznačný identifikátor tabuľky.



Obrázek 5.1: Diagram tried

Udržuje si zoznamy o ukazateľoch na objekty, ktoré reprezentujú stĺpce a o integritných obmedzeniach, ktoré tabuľka musí spĺňať. K priradeniu stĺpca k tabuľke slúži metóda *add_column*, ktorej parametrom je objekt triedy Column. Pre pridávanie integritných obmedzení slúžia metódy *add_primary_key*, *add_check*, *add_unique* a *add_foreign_key*, ktorých argumentami sú objekty tried v poradí PrimaryKey, Check, Unique, ForeignKey.

- **Column** – reprezentuje jednotlivé stĺpce v tabuľkách a udržuje si nasledujúce vlastnosti o stĺpcoch:
  - *name* – názov stĺpca, ktorý slúži ako jednoznačný identifikátor,
  - *datatype* – dátový typ,
  - *description* – popis stĺpca,
  - *default* – implicitná hodnota, ktorá je nastavená pri vkladaní nového riadku tabuľky, pokiaľ nie je hodnota zadaná explicitne,

  - *notnull* – integritné obmedzenie, ktoré hovorí, že v tomto stĺpci musí alebo nemusí byť hodnota uvedená a
  - *attnum* – poradové číslo stĺpca v tabuľke.

- **Constraint** – trieda je nadradenou triedou pre triedy ForeignKey, Check, Unique, PrimaryKey. Pre tieto triedy sú nasledujúce vlastnosti spoločné:
  - *name* – názov integritného obmedzenia,
  - *key* – zoznam stĺpcov, ktorých sa obmedzenie týka,
  - *description* – popis obmedzenia a
  - *namespace* – názov schémy, do ktorej obmedzenie patrí.

- **ForeignKey** – udržiava informácie o cudzích kľúčoch existujúcich v tabuľke. Trieda dedí metódy a vlastnosti od triedy Constraint. Obsahuje ešte ďalšie vlastnosti:
  - *reftable* – názov tabuľky, na ktorú sa kľúč odkazuje,
  - *refcolkey* – zoznam stĺpcov, ktoré slúžia v odkazovanej tabuľke ako jednoznačná identifikácia,
  - *refnamespace* – názov schémy, do ktorej odkazovaná tabuľka patrí,
  - *updaction* – v prípade zmeny hodnôt v odkazovanej tabuľke sa vykonajú zmeny v odkazujúcej tabuľke podľa akcie uvedenej v tejto vlastnosti,
  - *delaction* – v prípade zmazania hodnôt v odkazovanej tabuľke sa vykonajú zmeny v odkazujúcej tabuľke podľa akcie uvedenej v tejto vlastnosti a
  - *matchtype* – nastavenie presnosti zhody cudzích kľúčov pri porovnávaní.

- **Check** – udržiava informáciu o integritnom obmedzení, ktoré kontroluje či hodnoty v danom stĺpci alebo stĺpcoch vyhovujú zadanému výrazu. Okrem vlastností a metód, ktoré získa pomocou dedičnosti od triedy Constraint, je trieda rozšírená o výraz, ktorému majú hodnoty vyhovovať.

- **Unique** – táto trieda je identická s triedou Constraint a v diagrame je vytvorená ako zvláštna trieda kvôli naznačeniu, že tabuľka môže obsahovať niekoľko integritných obmedzení unikátnosti stĺpca alebo skupiny stĺpcov.

- **PrimaryKey** – trieda je identická s triedou Constraint a slúži len k znázorneniu faktu, že tabuľka môže mať najviac jeden primárny kľúč.

Uvedené triedy sú triedami, ktoré slúžia najmä pre uloženie získaných informácií z databázy. Pre popis formátovacích informácií slúžia nasledujúce triedy:

- **Font** – trieda popisuje typ písma, ktorý sa použije pre akýkoľvek element, ktorý má textový charakter. Príkladom môže byť písmo jednotlivých stĺpcov, písmo pre názov tabuľky a pod. Vlastnosti písmá sú reprezentované nasledujúcimi atribútmi triedy:
  - *family* – skupina, do ktorej písmo patrí (*serif, sans-serif, cursive, fantasy, monospace*),
  - *style* – rez písma (tučné, kurzíva, normálne),
  - *size* – veľkosť písma,

  - *name* – názov písma (*Times New Roman, Arial, Verdana* a iné) a
  - *color* – farba písma (reprezentovaná v RGB formáte alebo v hexadecimálnom formáte používanom v HTML).

- **Color** – trieda popisuje farbu napríklad pri písme alebo pri farbe pozadia v tabuľke a podobne. Farba je reprezentovaná buď v RGB formáte alebo v hexadecimálnom formáte, ktorá sa používa v jazyku HTML (♯000000 znamená čiernu farbu). K farbe je možné pristupovať pomocou metódy *get_colorHTML* alebo *get_colorRGB*. Pre nastavenie novej farby slúži metóda *set_color*.

- **Position** – trieda popisuje pozíciu v dvojrozmernom priestore. Obsahuje atribút súradnice na osi X a na osi Y.

Ďalšie triedy slúžia na zoskupenie vlastností, ktoré patria pri zobrazení objektu. Ide o triedy:

- **Object** – Trieda zoskupuje formátovacie vlastnosti pre Tabuľku. Ak by sa projekt rozširoval o zobrazenie informácie o pohľadoch, tak z tejto triedy by mohla práve trieda reprezentujúca pohľad dediť vlastnosti. Trieda zahrňuje tieto vlastnosti:
  - *normalfont* – implicitná hodnota pre typ písma používaného pri zobrazení,
  - *commentfont* – typ písma pre informácie, ktoré sú komentármi istého objektu,
  - *tablenamefont* – typ písma použitého pre zobrazenie názvu tabuľky,
  - *bgcolor* – farba pozadia tabuľky využívaná pre odlíšenie členstva tabuľky v schémach a
  - *position* – pozícia tabuľky vo výslednom obraze.

- **ForeignProp** – Trieda zoskupuje formátovacie vlastnosti pre cudzie kľúče. Od tejto triedy dedí trieda ForeignKey vlastnosti, ktorými sú začiatočný a koncový bod šípky značiacej vzťah medzi dvomi tabuľkami a typ písma použitého pre zobrazenie textu.

5.2 Architektúra nástroja

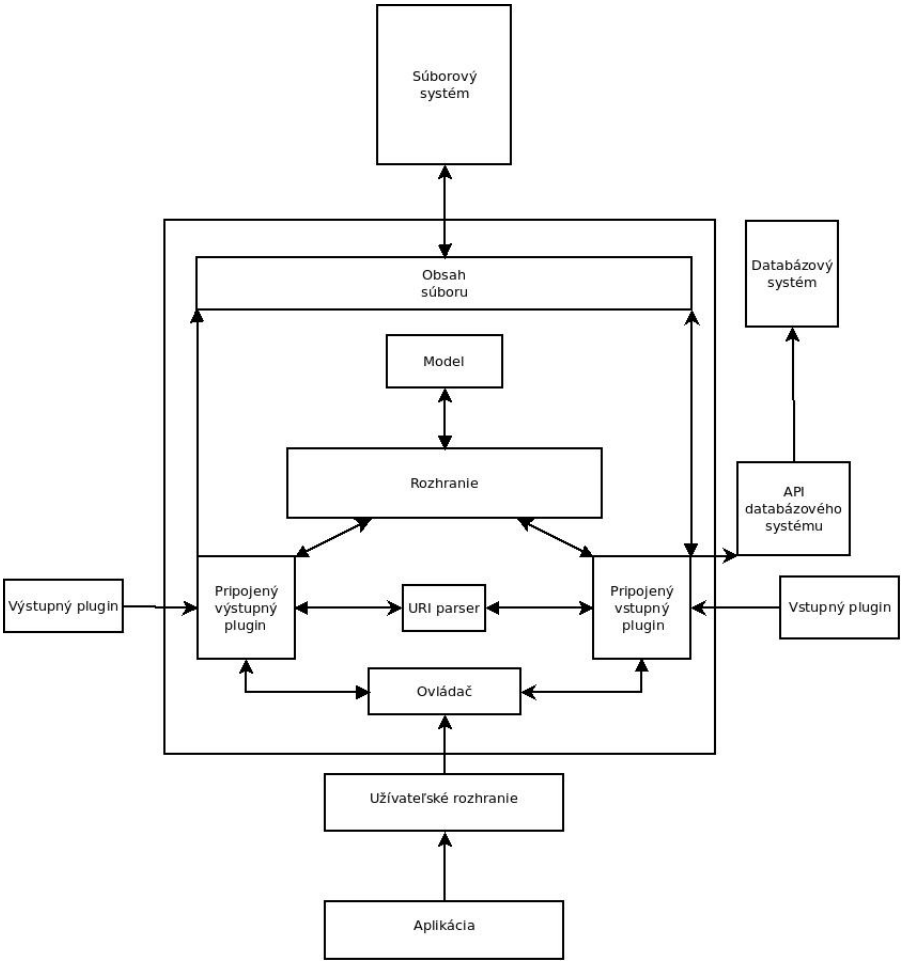
V nasledujúcej časti dokumentu budú popísané jednotlivé časti nástroja spolu s rozhraním, cez ktoré komunikujú. Grafický popis architektúry je zobrazený na obrázku 5.2.

Aplikácia pomocou užívateľského rozhrania komunikuje s knižnicou, presnejšie s ovládačom. Ten zabezpečuje riadenie programu na základe toho, o akú funkcionalitu má užívateľ záujem.

Jenotlivé výstupné alebo vstupné pluginy slúžia na rozširovanie knižnice, napríklad rozšírením podpory pre databázové systémy alebo pre rozšírenie podpory nových výstupných formátov. Viac o tom, akým spôsobom majú byť jednotlivé pluginy navrhnuté si povieme v ďalšej časti dokumentu.

Časť architektúry s názvom *model* je vnútornou reprezentáciou databázového modelu. Udržuje informácie získané z databázového systému a formátovacie informácie. Vytváranie tohto modelu riadi vstupný plugin pre databázu. Vstupný plugin využíva služby rozhrania, ktoré sú v systéme definované. Sú to napríklad funkcie pre pridanie tabuľky, pridanie stĺpca, pridanie integritných obmedzení a podobne.

Ďalšou časťou systému je URI parser, ktorý príjma URI identifikátor a z neho je schopný získať požadované informácie. URI identifikátor sa využíva na identifikáciu umiestnenia databázy, špecifikovanie autentizačných údajov a špecifikovanie mena databázy alebo schémy,



Obrázek 5.2: Architektúra knižnice

o ktorú má užívateľ záujem. Ďalšie využitie časti URI parser je pri zistení, kam sa má výstupný súbor uložiť a aký má mať názov alebo identifikovanie súboru, z ktorého sa majú informácie čítať v prípade aktualizácie.

Obsah súboru je výsledkom vykonania výstupného pluginu. Je to reprezentácia informácií, ktoré sa budú zapisovať do súboru a bude ich potom možné pomocou prehliadača zobraziť.

5.3 Popis pluginov

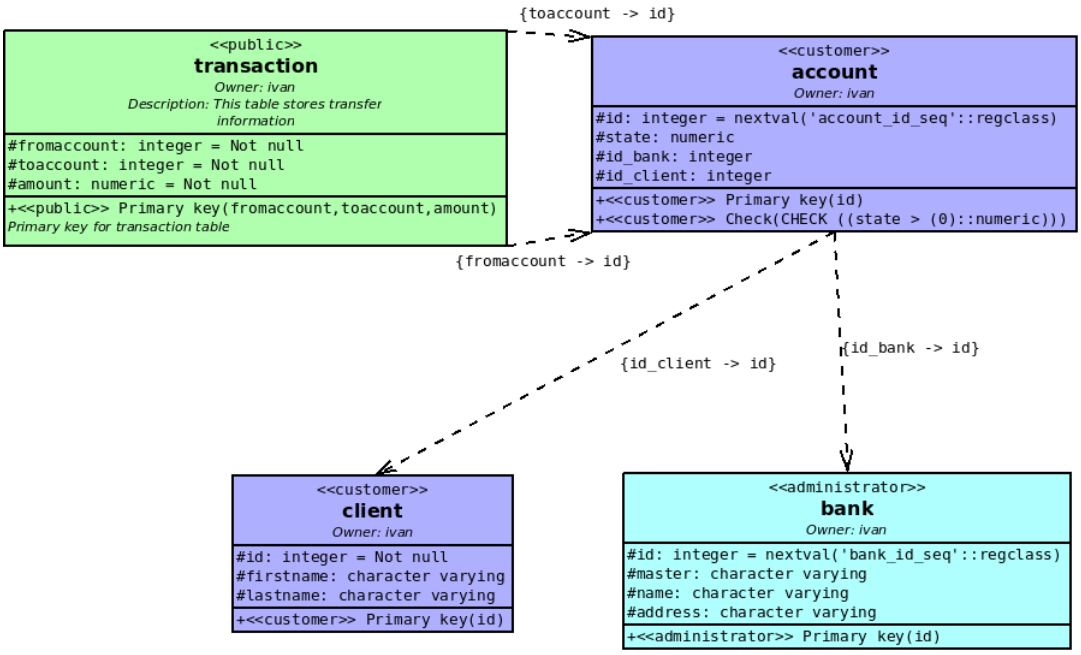
Významom pluginov v tomto projekte je rozširovanie možností použitia knižnice. To znamená pridávanie podpory pre ďalšie databázové systémy alebo pridávanie podpory pre ďalšie výstupné formáty. Každý plugin musí spĺňať nasledujúce špecifikácie:

- Rozhranie s knižnicou – Je nutné, aby knižnica po pripojení pluginu vedela spustiť plugin a preto každý vstupný plugin musí obsahovať funkciu *load*, ktorej vstupným parametrom je URI identifikátor zdroja. Pre výstupné pluginy platí, že musia obsahovať funkciu s názvom *save*, ktorej parameter je URI identifikujúci meno a umiestnenie novovzniknutého súboru.

- Rozhranie s vnútorným modelom – Pluginy musia využívať rozhranie, ktoré je zahrnuté v knižnice a slúži na prácu s vnútorným modelom. Popis tohto rozhrania je popísaný v časti 5.3.2.

5.3.1 Výstupný diagram

Výstupný plugin by mohol generovať výstup podobný ako je zobrazený na obrázku 5.3. Avšak podľa zvoleného editoru sa tento výstup môže meniť, pretože každý editor má svoje vlastné nástroje a identického zobrazenie sa nedá dosiahnuť. Preto by výstup na obrázku 5.3 mal slúžiť ako doporučené zobrazenie.



Obrázek 5.3: Výstupný diagram

Názvy tabuliek sa zobrazujú vo vrchnej časti objektu a sú zapísané tučným písmom. Nad názvom tabuľky je zobrazený názov schémy, do ktorej tabuľka patrí. Pod názvom je zobrazený vlastník tabuľky a popis tabuľky. V druhej časti objektu sú zobrazené jednotlivé stĺpce tabuľky oddelené od dátového typu dvojbodkou. Ak stĺpec obsahuje nejakú implicitnú hodnotu, tak je táto hodnota uvedená za znamienkom rovná sa. Ak neobsahuje žiadnu implicitnú hodnotu, ale vyžaduje sa uvedenie hodnoty v tomto stĺpci tabuľky, tak sa za znamienko rovná sa uvedie reťazec „Not null“. Inak sa za názov stĺpca neuvádza nič.

V tretej časti objektu sú zobrazené integritné obmedzenia. V každom riadku je najskôr názov schémy, do ktorého integritné obmedzenie patrí, potom je názov integritného obmedzenia (*Primary Key, Check, Unique*) a za názvom je v zátvorkách uvedený zoznam stĺpcov, ktorých sa integritné obmedzenie týka.

Cudzie kľúče sú zobrazené pomocou šípiek od jedného objektu k druhému. Pri každej šípke je uvedené, ktorý stĺpec v prvej tabuľke sa odkazuje na stĺpec v druhej tabuľke.

5.3.2 Rozhranie pre prácu s modelom

Pre prácu s vnútorným modelom slúži rozhranie, ktoré umožňuje do modelu vkladať tabuľky, stĺpce tabuliek a ďalšie informácie súvisiace s modelom. Rozhranie je využívané programátormi nových rozširujúcich pluginov. Pri ich vytváraní je odporúčané použiť práve toto rozhranie, pretože pri vkladaní nových informácií do vnútorného modelu sa kontroluje správnosť použitých dátových typov a tým je zabezpečená aj kompatibilita s ostatnými pluginami. Rozhranie je reprezentované objektom *PluginInterface* a má nasledujúce metódy, ktoré pracujú s vnútorným modelom.

Pre pridanie tabuľky do modelu slúži metóda *add_table* a jej vstupným argumentom je objekt triedy *Table*. Pridanie riadkov do tabuľky vykonáva funkcia *add_column* a jej argumentami sú názov tabuľky, do ktorej sa vkladá stĺpec a objekt triedy *Column*.

Pre vkladanie integritných obmedzení tabuliek sa využívajú metódy *set_primary_key, add_foreign_key, add_check, add_unique*. Každá táto metóda má ako jeden argument názov tabuľky a druhým argumentom je objekt triedy podľa typu integritného obmedzenia (triedy *PrimaryKey, ForeignKey, Check, Unique*).

Pre získavanie informácií z modelu slúžia metódy *get_table, get_column*. Prvá spomenutá metóda získa informácie o všetkých tabuľkách, ktoré sa v modeli vyskytujú. Druhá metóda získa všetky informácie o jednotlivých stĺpcoch v tabuľke podľa zadaného argumentu, ktorým je názov tabuľky.

Získavanie informácií o integritných obmedzeniach sa vykonáva metódami *get_check, get_primary_key, get_foreign_key, get_unique*, ktorých jediným argumentom je názov tabuľky, z ktorej sa integritné obmedzenia získavajú.

Týmto rozhraním je v modeli možné nastavovanie a získavanie formátovacích informácií pomocou metód, ktoré majú vždy ako prvý argument názov tabuľky, pre ktorú sa nová vlastnosť nastaví alebo získa. Nastavenie a získanie pozadia pre tabuľku sa deje pomocou *get_bgcolorHTML, set_bgcolorHTML*, kde argumentom pri nastavení farby je hexadecimálna hodnota farby.

Pri nastavovaní písma pre jednotlivé účely slúžia metódy *get_normalfont, set_normalfont, get_commentfont, set_commentfont, get_tablenamefont, set_tablenamefont*. Argumentami v týchto metódach sú názov tabuľky a informácie súvisiace s fontom v poradí: skupina písma, rez písma, názov písma, veľkosť, farba.

Pre nastavenie a získanie pozície tabuľky vo výslednom zobrazení sa používa metóda *set_position, get_position*, ktorej argumentami sú názov tabuľky a súradnice na osi X a osi Y.

Kapitola 6

Implementácia

Implementácia je predposledným krokom, ktorý sa v životnom cykle softvéru nachádza[1]. Je to proces, pri ktorom sa z návrhu stáva použiteľný softvér, môžeme povedať, že návrh sa transformuje do konštrukcií programovacieho jazyka a následne do spustiteľného kódu. V tejto kapitole budú popísané jednotlivé časti nástroja, nie z pohľadu „čo“ robia, ale z pohľadu „ako“ to robia. Popísaný bude URI parser, vstupné a výstupné pluginy.

6.1 URI Parser

URI parser slúži na rozdelenie URI reťazca a získanie informácií, ktoré v sebe reťazec zahŕňa. Zo štandardnej knižnice využíva modul *re*, ktorý slúži na prácu s regulárnymi výrazmi. Regulárny výraz, ktorý rozdeľuje vstupný URI reťazec na jednotlivé jeho časti bol použitý z RFC 3986[1] a jeho zápis vyzerá nasledovne:

```
^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
```

Pri identifikácii databázového systému je URI parser schopný získať zo zadaného reťazca:

- protokol – názov databázového systému napríklad postgres, mysql alebo sqlite,
- užívateľ – meno užívateľa, ktorý sa do databázy prihlasuje,
- heslo – heslo, ktorým sa užívateľ autentizuje,
- umiestnenie databázového systému – identifikácia pomocou doménového mena alebo IP adresy,
- port – číslo portu, na ktorom databázový systém očakáva spojenia,
- názov databázy – názov databázy, ktorú chceme vizualizovať a
- názov schémy – obmedzenie zobrazenia len na jednu schému.

Modul URI parser sa volá pomocou funkcie *uri_split()*, ktorej argumentom je URI reťazec. Návratovou hodnotou tejto funkcie je objekt URIparse a pomocou jeho metód je možné pristupovať k jednotlivým hodnotám, ktoré boli v URI reťazci zadané. V prípade, že hodnota v URI reťazci nebola zadaná, tak metóda vráti hodnotu `None`.

[1] Platí pre model Waterfall

6.2 Vstupné pluginy

Existujú dva typy vstupných pluginov môžeme ich označiť:

- Databázové – získavajú infomácie o databázovom modeli z databázového systému.
- Súborové – získavajú formátovacie informácie zo súboru, ktorý už bol vytvorený.

Databázové vstupné pluginy využívajú jeden z vytvorených modulov pre komunikáciu s databázovým systémom. Pri databázovom systéme PostgreSQL bol použitý modul *PyGreSQL*. Na vytvorenie spojenia slúži funkcia, ktorej návratovou hodnotou je objekt triedy *pgobject*. Jednou z metód tohto objektu je metóda *query*, ktorej vstupným argumentom je dotaz zapísaný v jazyku SQL a návratová hodnota tejto metódy je objekt *pgqueryobject*. Je to objekt reprezentujúci odpoveď, ktorú databázový systém vráti na dotaz. *Pgqueryobject* objekt sprístupňuje výsledok pomocou metódy *getresult* alebo *getdict*, ktoré vrátia výsledok ako zoznam n-tíc alebo ako zoznam asociatívnych polí. K n-ticiam je možné pristupovať pomocou číselných indexov a do asociatívneho poľa je možné pristupovať pomocou kľúča. V takom prípade je kľúčom názov stĺpca, z ktorého chceme hodnotu použiť.

Vytváranie modelu sa deje nasledovne a to získaním informácií o tabuľkách a pre každú tabuľku sa zistia informácie o stĺpcoch a integritné obmedzenia, ktoré musia hodnoty tabuľky spĺňať. Každá tabuľka je reprezentovaná objektom triedy *Table* a udržujú si referencie na objekty triedy *Column*, ktorá je abstrakciou pre stĺpce v databázovom modeli. Úplne rovnako je to aj s integritnými obmedzeniami, ktoré sú reprezentované objektami tried *PrimaryKey, ForeignKey, Check* a *Unique*.

Súborové vstupné pluginy nezískavajú informácie o tom, aké tabuľky sa v súbore nachádzajú a aké stĺpce či integritné obmedzenia sú tam, ale získavajú len formátovacie informácie z existujúceho súboru. Formáty súborov, ktoré sú momentálne podporované knižnicou, sú formáty značkovacieho jazyka XML. V Pythone existuje niekoľko modulov, ktoré umožňujú prácu s takto štruktúrovanými súbormi. Pre súborové pluginy bol vybraný modul etree, ktorý je súčasťou balíčka lxml. Tento modul zo zadaného súboru vytvorí jeho model a pomocou jazyka XPath je možné pristupovať k jednotlivým častiam dokumentu. Pomocou tohto jazyka sa získavajú jednotlivé formátovacie informácie, ako napríklad: umiestnenie tabuľky vo výslednom obraze, veľkosť, farba, typ písma použité v jednotlivých tabuľkách a podobne.

6.3 Výstupné pluginy

Výstupné pluginy transformujú model, ktorý bol vytvorený pomocou vstupných pluginov na model, ktorý reprezentuje štruktúru vo výstupnom súbore. Formát výstupných súborov je vytvorený pomocou značkovacieho jazyka XML. K vytváraniu uzlov a štruktúry súboru bol použitý modul etree z balíčka lxml. Po vytvorení modelu súboru sa tento model zapíše do súboru. Plugin pristupuje k vnútornej reprezentácii databázového modelu pomocou rozhrania, ktoré bolo pre tento účel vytvorené. Získava informácie o modeli a následne tieto informácie vkladá do štruktúry súboru.

Pre odlíšenie tabuliek podľa toho, do ktorej schémy patria sa využíva to, že sa ich podklad vo výslednom obraze farebne líši. Pri vytváraní štruktúry súboru je vždy vytvorené asociatívne pole, ktoré mapuje jednotlivé názvy schém na hodnotu farby pre ich podklad.

Výstupná štruktúra súboru začína koreňovým uzlom podľa definície štruktúry. Neskôr sa k nemu pripoja uzly, ktoré reprezentujú tabuľky a uzly, ktoré reprezentujú ďalšie informácie spojené s tabuľkou.

Kapitola 7

Záver

Cieľom tejto práce bolo vytvorenie nástroja, ktorý by bol schopný vizualizovať databázový model. Tento projekt nemal so sebou priniesť širokú podporu pre rôzne databázové systémy alebo podporu pre rozličné výstupné súborové formáty, ale jeho prioritou bolo vytvorenie jadra knižnice, ktoré by sa jednoducho mohlo rozširovať o podporu pre ďalšie databázové systémy a pre súborové formáty. Na rozširovanie slúžia tzv. pluginy, ktoré môžu byť rozdelené na vstupné a výstupné. Vstupné pluginy slúžia na získavanie informácií z databázového systému a vytváranie vnútorného modelu, ktorý je nasledovne transformovaný výstupnými pluginmi na štruktúrovaný obsah, ktorý možno zobraziť v prehliadači alebo v editore. Vstupné pluginy umožňujú načítavať formátovacie informácie zo zvoleného súboru, ktorý bol predtým vytvorený týmto nástrojom. Vďaka týmto pluginom je možné prevádzať jeden formát súboru na iný, v aktuálnej verzii je tento prevod možné vykonať medzi natívnym fomátom XML súboru a formátom pre existujúci program *Dia – diagram editor*.

Pri získavaní informácií z databázového systému sa vytvára model, ktorý v sebe ukladá základné informácie zistiteľné z každého systému. Ak by každý databázový systém získaval iné informácie a vytváral vlastný model, tak by bolo nutné pre každý databázový systém vytvárať vlastné pluginy. Vytvorením spoločného modelu sa zabezpečuje kompatibilita medzi jednotlivými pluginmi a pre jeden výstupný formát existuje práve jeden výstupný plugin.

7.1 Aktuálny stav

V aktuálnom stave vie tento projekt zobraziť základné informácie o tabuľkách, ktoré sa v databáze nachádzajú. Informácie, ktoré tento dokument zmieňuje sú nasledujúce:

- základné informácie o tabuľke – názov, vlastník, popis,
- informácie o stĺpcoch – názov, dátový typ, implicitná hodnota a
- integritné obmedzenia – primárny kľúč, cudzí kľúč, kontrola hodnoty, unikátnosť hodnoty, možnosť neuviesť hodnotu vo zvolenom stĺpci.

Podpora databázových systémov je v aktuálnej verzii na nízkej úrovni, ale vzhľadom na požiadavky, ktoré boli kladené na nástroj je možné povedať, že boli v tomto smere naplnené. Existuje podpora pre systém PostgreSQL, ktorý požadoval zástupca firmy Red Hat, Martin Bačovský.

Požiadavky týkajúce sa podpory výstupných súborových formátov sa tiež podarilo naplniť a preto je možné využívať v nástroji dva súborové formáty a tým sú natívny formát XML a formát pre *Dia – diagram editor*.

Pomocou nástroja je možné zo získaných informácií z databázového systému získať procesom vizualizácie obrazovú informáciu. Táto informácia môže byť popísaná jedným zo súborových formátov. Ako bolo spomenuté vyššie v tejto kapitole, tak je možné robiť konverziu jedného súborového formátu na druhý pomocou tohto nástroja. Pri tejto konverzii dochádza k načítaniu formátovacích informácií zo vstupného súboru a doplnenie týchto infomácií o aktuálne informácie získané z databázového systému.

7.2 Rozšírenia

Projekt sa nachádza v ranej verzii a je preto veľmi pravdepodobné, že by jeho možnosti mohli byť rozšírené. Jednotlivé rozšírenia, ktoré by sa mohli aplikovať na nástroj sa týkajú zobrazovaných informácií, databázových systémov, súborových formátov a rozloženia tabuliek vo výslednom zobrazení.

Na trhu existuje niekoľko databázových systémov a tie najznámejšie a najpoužívanejšie by mohli byť podporované v ďalších verziách nástroja. Napríklad by mohla byť vytvorená podpora databázových systémov MySQL, Microsoft SQL Server, Oracle database a iných. Ďalšou databázou, ktorá sa veľmi často využíva v aplikáciách napríklad pre rýchle, jednoduché a perzistentné ukladanie dát sa nazýva SQLite. Ide o databázu, ktorá nepotrebuje žiadny komunikačný proces medzi klientom a serverom. Databáza podporuje SQL dotazy a sú vykonané priamo nad súborom, ktorý sa nachádza na lokálnom disku alebo inom médiu. Vďaka jej jednoduchosti ju často podporujú aj programovacie jazyky a táto databáza by mohla mať svoju podporu vo vytváranom nástroji.

Možnými rozšíreniami v oblasti podpory súborových formátov by mohli byť podpora pre SVG formát, DocBook, HTML. Nástroj by mohol byť schopný vytvárať výstupný formát pre systém LaTeX, aby bolo možné využívať obrazovú informáciu priamo v odborných článkoch alebo publikáciách. Nástroj by mohol vygenerovať popis modelu pre prostredie *picture*, ktoré slúži na kreslenie vektorovej grafiky.

Získavanie informácií len o tabuľkách by sa mohlo rozšíriť o podporu získavania a neskoršieho zobrazenia ďalších informácií. Tými sú myslené informácie o indexoch alebo užívateľských pohľadoch, ktoré slúžia na zobrazenie istého výrezu informácií, ktoré sú podstatné pre istú skupinu užívateľov. Prípadne by sa mohli z databázy získavať informácie o triggeroch, ktoré sú vykonávané na základe nejakého podnetu. Ďalšími informáciami získavanými z databázového systému by mohli byť informácie o procedúrach alebo o štatistických údajoch, ktoré sa dajú získať z databázy.

Aktuálne sa tabuľky vo výslednom obraze skladajú na jedno miesto. Keďže nebola definována požiadavka na rozmiestnenie tabuliek vo výslednom zobrazení, tak neboli v nástroji implementované žiadne algoritmy, ktoré by mohli pomôcť s rozložením tabuliek vo výsledku. Preto by mohlo implementovanie algoritmov, ktoré sú schopné vytvoriť výsledné rozloženie pomocť, pri práci s týmto nástrojom. Najmä by sa ušetril čas, ktorý by musel byť použitý na rozmiestnenie tabuliek, ak by algoritmy neboli implementované. Algoritmy by mohli zahrňovať aj vlastnosť, ktorá hovorí o čo najmenšom križovaní jednotlivých vzťahov pri tabuľkách.

7.3 Vlastný prínos

Myšlienka vytvoriť tento projekt bola iniciovaná firmou Red Hat. Firmu zastupoval Martin Bačovský, ktorý definoval ciele a požiadavky projektu. Počas vývoja projektu bol každý pokrok konzultovaný s Martinom Bačovským a na základe konzultácií boli upravené niektoré požiadavky, prípadne doplnené nové špecifikácie. Kedže projekt sa radí medzi tie menšie, tak som nástroj pre vizualizáciu implementoval sám a stojím za jeho analýzou a návrhom.

Firma Red Hat by chcela, aby tento nástroj bol naďalej udržovaný a obohacovaný o nové možnosti. Preto verím, že spolupráca na tomto projekte bude pokračovať a dosiahneme nakoniec hlavného cieľu – zahrnutie balíku do distribúcie Fedora.

Literatura

[1] The Internet Society: *Uniform Resource Identifier (URI): Generic Syntax* [online]. 2005 [cit. 2008-5-15].
URL http://tools.ietf.org/html/rfc3986

[2] *PyGreSQL - PostgreSQL module for Python* [online]. 2006-01-17 [cit. 2010-05-15].
URL http://www.pygresql.org/

[3] Python Software Foundation: *Python Programming Language* [online]. 2010 [cit. 2008-05-15].
URL http://www.python.org/about/

[4] *argparse - Project Hosting on Google Code* [online]. 2010 [cit. 2010-05-15].
URL http://code.google.com/p/argparse/

[5] CLOTHES, J.: *Return to Chauvet Cave: excavating the birthplace of art : the first full report*. Thames & Hudson, 2003, iSBN 0500511195.

[6] CONNOLLY, T. M.; BEGG, C. E.: *Database systems: a practical approach to design, implementation, and management*. Addison-Wesley, 1999, 1093 s., iSBN 0201342871.

[7] FRISCH, N.; aj.: Visualization and pre-processing of independent finite-element meshes for car crash simulations. *The Visual Computer*, ročník 18, jún 2002, ISSN 0178-2789.

[8] JARDINE, D. A.: The ANSI/SPARC DBMS model: proceedings of the second SHARE Working Conference on Data Base Management Systems. *ACM SIGIR Forum*, ročník 12, december 1977, iSSN 0163-5840.

[9] MACH, P.: *Python - programování zábavou* [online]. 2009-02-24 [cit. 2010-05-15].
URL http://python.wraith.cz/index.php

[10] MACKINLAY, J. D.; CARD, S. K.; SHNEIDERMAN, B.: *Readings in information visualization: using vision to think*. Academic Press, 2003, 410 s., iSBN 1558609156.

[11] STEPHENS, R.; PLEW, R.: *SAMS teach yourself beginning databases in 24 hours*. Sams Publishing, 2003, iSBN 0-672-32492-X.

[12] VENERS, B.: The Making of Python. *Artima Developer* [online], 2003-01-13 [cit. 2010-05-15].
URL http://www.artima.com/intv/pythonP.html

[13] VENERS, B.: Programming at Python Speed. *Artima Developer*[online], 2003-01-17 [cit. 2010-05-15].
URL http://www.artima.com/intv/speed.html

Dodatek A

Obsah CD

CD-ROM obsahuje:

- visualizer – nástroj na vizualizáciu,
- potrebné pluginy – pluginy, ktoré je nutné nainštalovať, kvôli funkčnosti programu,
  - argparse-1.1
  - PyGreSQL-4.0
- teoretická časť bakalárskej práce – zdrojové súbory dokumentu napísane v LaTeXu a
- skript testovacej databázy.

Dodatek B

Návod na použitie nástroja

V tejto sekcii bude popísaná inštalácia a používanie nástroja, ktorý bol vyvíjaný v tomto projekte. V dobe písania dokumentu nebol nástroj zverejnený na internete a jediný spôsob ako ho získať, je skopírovanie adresára *visualizer* z prenosného nosiča, ktorý je priložený k dokumentu.

B.1 Inštalácia

Aktuálna verzia nástroja používa moduly, ktoré nie sú súčasťou štandardnej knižnice jazyka Python a preto je nutné ich doinštalovať. Jedným z modulov, ktorý je nutné doinštalovať, je argparser, ktorý slúži na získavanie informácií z príkazového riadku. Modul je možné stiahnuť a nainštalovať podľa návodu uvedeného na stránkach [4], alebo ho môžete skopírovať do počítača z CD. Nachádza sa v adresári „Potrebné moduly“. Adresár s názvom „argparse-1.1“ skopírujete a spustíte skript pomocou nasledujúceho príkazu:

```
python setup.py install
```

sa modul nainštaluje. Rovnako postupujte aj pri inštalovaní modulu s názvom „PyGreSQL-4.0“, ktorý je dostupný na stránkach [2] alebo na CD.

Nástroj na vizualizáciu je na CD neskomprimovaný a teda jeho inštalácia spočíva v skopírovaní adresára „visualizer“ do počítača.

B.2 Ovládanie

Nástroj sa spúšťa pomocou skriptu *main.py* z príkazového riadku. Pre zobrazenie nápovedy zadaj príkaz:

```
python main.py --help
```

Nasledujúce parametre je nutné zadávať vždy pri používaní nástroja.

- –input_uri – za ním je zadaný identifikátor URI pre databázový systém,
- –output_uri – za ním je zadaný identifikátor výstupného súboru

- –input_plugin – názov použitého pluginu. V prípade PostgreSQL databázového systému je to *in_postgres*.
- –output_plugin – názov použitého pluginu pre výstupný obsah. V prípade *Dia* je to *op_dia* a v prípade natívneho formátu XML je to *op_xml*.

Napríklad príkazom:

```
python main.py --input_uri postgres://user@192.168.2.1/test
   --input_plugin ip_postgres
   --output_uri file:///home/user/output.dia
   --output_plugin op_dia
```

sa získavajú infomácie z databázového systému spusteného na adrese *192.168.2.1*, konkrétne sa získavajú z databázy s názvom *test* a prihlásenie do databázy sa deje cez užívateľa *user*. Využíva sa pri tom plugin s názvom *ip_postgres*. Výstupom je súbor identifikovaný pomocou URI *file:///home/user/output.dia* a ako výstupný plugin sa použije *op_dia*.

Pri aktualizácií už existujúceho súboru by sa nástroj spustil nasledovne:

```
python main.py --input_uri postgres://user@192.168.2.1/test
   --input_plugin ip_postgres
   --output_uri file:///home/user/output2.dia
   --output_plugin op_dia
   --update_uri file:///home/user/output.dia
   --update_plugin ip_dia
```

V tejto situácií sa vytvorí model, podľa aktuálneho stavu databázy a doplnia sa formátovacie informácie zo súboru *file:///home/user/output.dia*. Pri tomto doplnení sa využije *ip_dia* plugin.