UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Grafická podpora návrhu relačních databází

Martin Němec

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin NĚMEC**
Osobní číslo:	**I08122**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Grafická podpora návrhu relačních databází**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Primárním cílem práce je vytvoření nástroje pro návrh relačních databází s grafickou podporou (SQL Data Modeler).

Teoretická část práce bude obsahovat:
vysvětlení základních pojmů a principů z oblasti databázových systémů, především s ohledem na relační databáze a relační model dat (RMD), základní popis SQL s podrobným popisem především příkazů pro definici datových struktur.

Praktická část:
zhodnocení vybraných existujících podobných produktů s podporou grafického modelování, návrh vhodných datových struktur pro popis databáze, implementace vlastního nástroje, který bude umožňovat grafické modelování tabulek databáze a vzájemných relací mezi nimi, vyznačení kardinality a parciality vztahu, vygenerování SQL kódu na základě modelu, export do XML, export obrázku s modelem, vytvoření modelu na základě existují DB (reverse engineering), apod.
Nástroj bude postaven především na uživatelské přívětivosti a snaze o co nejvyšší efektivitu při návrhu modelu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***GROFF, James R.; WEINBERG, Paul N. SQL : Kompletní průvodce. Brno : Computer Press, 2005. 936 s. ISBN 80-251-0369-2.**

***RIORDAN, Rebecca M. Vytváříme relační databázové aplikace. Praha : Computer Press, 2000. 280 s. ISBN 80-7226-360-9.**

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra softwarových technologií

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 13. 5. 2011 Martin Němec

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, Ing. Petru Veselému, za jeho názory a návrhy k této práci.

Anotace

Cílem práce je vytvoření aplikace umožňující interaktivní modelování diagramů databází. Teoretická část se zabývá základními pojmy z oboru relačních databází a popisem jazyka SQL. Dále popisuje návrh aplikace pro databázové modelování v programovacím jazyce Java.

Klíčová slova

relační, databáze, model, tabulka, atribut, vztah, relace

Title

Grphical support for relational database design

Annotation

Goal of this work is to create appliacation for interactive database diagram modelling. Theoretical part considers with relational database basic concepts and describing SQL language. Also describes design of application for database modeling implemented in Java programming language.

Keywords

relational, database, model, table, attribute, reference, relation

Obsah

Seznam zkratek

API	Application Programming Interface
DBMS	Database Management System
DDL	Data Definition Language
GPL	General Public Licence
GUI	Graphical User Interface
IDE	Integrated Development Environment
GIMP	GNU Image Manipulation Program
GTK	GIMP Toolkit
MFC	Microsoft Foundation Class
PDF	Portable Document Format
PNG	Portable Network Graphics
SQL	Structured Query Language
SVG	Scalable Vector Graphics
XML	Extensible Markup Language

Popis stylů

Tímto stylem je v práci psán běžný text.

Kurzívou jsou psána slova, týkající se implementace. Jsou to například datové typy, názvy tříd apod.

KAPITÁLKY JSOU POUŽITY PRO RŮZNÉ VOLBY V NABÍDKÁCH PROGRAMU.

```
Tento styl značí zdrojový kód.
```

Seznam obrázků

Seznam tabulek

1 Úvod

Relační databáze jsou již dlouhou dobu oblíbeným, efektivním a snadno spravovatelným řešením pro ukládání velkých objemů dat. Díky jazyku SQL nad nimi pracují různorodé aplikace napsané v různých programovacích jazycích.

Návrh databáze se tak stává velmi specifickým a důležitým procesem návrhu aplikace. Z tohoto důvodu je velice výhodné před samotnou implementací sestavit diagram, který názorně vystihuje fungování celého systému.

Aplikace, které jsou pro účely databázového modelování navrženy, jsou často závislé na konkrétním databázovém systému. I proto je primárním výstupem této práce nástroj, umožňující databázové modelování, který si klade za cíl být nezávislý na použitém databázovém systému. Program tak bude, pomocí rozšiřujících zásuvných modulů, umožňovat export vytvořených databází do různých podob databázového jazyka SQL.

Teoretická část práce zahrnuje vysvětlení základních pojmů, hlavně z relačních databází. Zmiňuje se i o navrhování databází a diagramech k tomu určených. Dále se zabývá popisem jazyka SQL a jeho nejdůležitějších konstrukcí.

Dále práce vysvětluje datové struktury, použité pro popis databázových součástí, které program ve svém kódu využívá. Dále popisuje důvody pro volbu jazyka Java jakožto implementačního nástroje a technologie, použité při implementaci vlastní aplikace. Následně vystihuje celý proces návrhu aplikace, její grafické reprezentace a fungování jejích jednotlivých částí. Popisuje také některé třídy jazyka Java použité pro určité pokročilé funkce programu.

Závěr mapuje míru úspěšnosti implementace vlastní aplikace a splnění cílů. Popisuje některé problémy, které se při implementaci vyskytly. Hodnotí využitelnost aplikace a možnost jejího užití.

2 Základní pojmy

Základními pojmy z databázových systémů, především z relačních databází, se zabývá kniha Vytváříme relační databázové aplikace od Rebeccy M. Riordan [7], v níž jsem při sestavování této kapitoly hledal inspiraci.

2.1 Databáze

Databází, v obecném smyslu slova, chápeme jakékoliv úložiště uchovávající nějaká data. Za databázi můžeme označit například různé seznamy telefonních čísel, kartotéky, soubory na disku, apod. V užším smyslu slova může být databází zamýšlen databázový systém či konkrétní implementace datového úložiště v databázovém systému pro danou aplikaci.

2.2 Datový model

Datový model představuje způsob, jak je možné data reprezentovat, jak se ukládají a jak se s nimi pracuje. Příklady datových modelů mohou být: relační model, hvězdicový model nebo hierarchický model.

2.3 Databázový systém

Databázový systém neboli systém řízení báze dat, často označovaný anglickou zkratkou DBMS, je nástroj, který slouží k manipulaci s daty, k ukládání a načítání dat, přičemž před uživatelem skrývá fyzickou implementaci struktur sloužících k uchovávání těchto dat. Uživatel (programátor) potom přistupuje k těmto datům nejčastěji za pomoci nějakého vyššího jazyka (např. SQL).

2.4 Relační databáze

Jako relační databáze se označuje databázový systém, jež implementuje relační datový model a na jeho základě s daty pracuje. Někdy se tímto slovním spojením označuje i konkrétní softwarové řešení implementované v relačním databázovém systému. Některými z běžných relačních databázových systémů jsou například: Oracle, Microsoft SQL server nebo MySQL.

2.5 Relační model

2.5.1 Princip relačního modelu

Relační model navrhnul koncem šedesátých let Dr. Edgar Frank Codd. Jeho název je odvozen od relace, která je základním stavebním kamenem relačních databází (pozn. dnešní relační databáze neimplementují tento model zcela). Je to pomyslná struktura dat s uspořádanými řádky a sloupci. V relačních databázích může být relací tabulka nebo třeba výsledek dotazu, který nemá žádnou fyzickou paměťovou reprezentaci. Mylně se často lidé domnívají, že pojem relační databáze má své označení díky vztahům, kterým se rovněž někdy říká relace.

2.5.2 Atribut

Atributem je sloupec tabulky. Běžně vyjadřuje nějakou vlastnost, míru nebo kvantitu konkrétního nebo abstraktního objektu, o kterém uchováváme data.

2.5.3 Vektor

Vektor neboli řádek (případně záznam) je instancí vložených a následně tedy uchovávaných dat. Vektory nejsou v tabulce v žádném konkrétním pořadí. Neznamená tedy, že poslední vložená data musí být na konci (nebo na jiném konkrétním místě) tabulky.

2.5.4 Primární klíč

Primárním klíčem (primary key) programátor zvolí jeden nebo více sloupců, které jednoznačně identifikují řádek v tabulce. S tím souvisí pojem kandidátní klíč, který označuje libovolnou množinu sloupců, jednoznačně identifikující řádek v tabulce. Na obrázku 1 je tabulky, ve které je sloupec osobní číslo typickým atributem, který může tvořit primární klíč. Předpokládá se, že každá osoba má jiné osobní číslo.

Osobní číslo	***Jméno***	***Příjmení***
118523	Jan	Novák

Obrázek 1 - Příklad primárního klíče

2.5.5 Vztah

Vztah mezi entitami reflektuje vztah těchto objektů v reálném světě. Tento vztah může být realizován obecně mezi jakýmikoliv relacemi, nicméně zde se budu soustředit na vztahy mezi entitami z hlediska návrhu databáze. Entity v tomto vztahu nazýváme účastníky. Vztahy dělíme podle počtu účastníků v nich vystupujících na binární a ternární. Vztah v relačních databázích pomáhá řešit problém duplicity dat a s tím spojené náklady na správu databáze. V těchto jsou vztahy řešeny pomocí cizích klíčů.

2.5.6 Kardinalita vztahu

Kardinalita určuje, kolik řádků tabulky se vztahu účastní. Podle ní dělíme vztahy na 1:1, 1:N a M:N. Vztah 1:1 (jedna k jedné) znamená, že pro každý řádek první tabulky existuje právě jeden odpovídající řádek ve druhé tabulce. Analogicky u vztahu 1:N platí, že každému záznamu první tabulky odpovídá libovolný počet záznamů druhé tabulky. Vztah M:N říká, že množině záznamů první tabulky odpovídá jiná množina záznamů druhé tabulky.

2.5.7 Parcialita vztahu

Parcialita vztahu udává, zda je entita povinna účastnit se vztahu, či nikoliv.

2.5.8 Cizí klíč

Cizím klíčem (foreign key) nazýváme sloupec (nebo množinu sloupců) tabulky, který odkazuje na celý primární klíč jiné tabulky. Díky tomuto lze vytvářet vzthy typu N:1 mezi tabulkou s cizím klíčem a tabulkou s primárním kíčem, na který je v tomto vztahu odkazováno. Dalšími akcemi lze docílit chování, jaké je očekáváno od vztahů 1:1 nebo M:N. Na obrázku 2 je znázorněna tabulka, ve které je cizím klíčem sloupec „osobní číslo“. Odkazuje se na tabulku na obrázku 1. Každá osoba může mít několik telefonů. Nemusí mít ovšem žádný. V tomto případě má osoba s číslem 118523 (Jan Novák) dvě telefonní čísla.

Osobní číslo	*Telefon*
118523	+420 123 456 789
118523	+420 987 654 321

Obrázek 2 - Příklad cizího klíče

2.6 Normalizace tabulek

2.6.1 Normální formy

Normální formy jsou doporučení pro tvorbu databázových tabulek, která zajišťují základní optimalizaci těchto tabulek z hlediska úspory prostředků. Čím je tabulka ve vyšší normální formě, tím je optimalizovanější. Každá z normálních forem obsahuje všechny předešlé a přidává další požadavky. Normálních forem je celkem šest, ale běžně se používají pouze tři. Vyšší normální formy se týkají speciálních případů, ke kterým příliš často nedochází.

2.6.2 První normální forma

Tabulka je v první normální formě v případě, že všechny její atributy jsou dále nedělitelné (atomické) hodnoty. Na obrázku 3 je tabulka, která první normální formu nesplňuje. Jména a příjmení jsou v tomto případě v jednom sloupci, stejně tak email a telefonní číslo. Na obrázku 4 je tabulka po úpravě takové, aby první normální formu splňovala.

Jméno a příjmení	*Kontakt*
Jan Novák	jan(zavináč)novak.cz, +420 123 233 896
Josef Novotný	josef(zavináč)novotny.cz, +420 123 456 789

Obrázek 3 - Tabulka nesplňující první normální formu

Jméno	*Příjmení*	*Email*	*Telefon*
Jan	Novák	jan(zavináč)novak.cz	+420 123 223 896
Josef	Novotný	josef(zavináč)novotny.cz	+420 123 456 789

Obrázek 4 - Tabulka splňující první normální formu

2.6.3 Druhá normální forma

Ve druhé normální formě je tabulka, jejíž všechny atributy jsou závislé na jejím primárním klíči. Na obrázku 5 je zobrazena tabulka, která splňuje třetí normální formu. Klíčovým atributem (primárním klíčem) je atribut id, který je unikátním číslem, na němž jsou všechny ostatní atributy závislé.

Id	*Jméno*	*Příjmení*	*Email*	*Telefon*
1	Jan	Novák	jan(zavináč)novak.cz	+420 123 223 896
2	Josef	Novotný	josef(zavináč)novotny.cz	+420 123 456 789

Obrázek 5 - Tabulka splňující druhou normální formu

2.6.4 Třetí normální forma

Třetí normální formu tabulka splňuje, pokud neexistuje závislost mezi jejími jednotlivými neklíčovými atributy.

2.7 Entitně-relační vztahy

2.7.1 E-R diagram

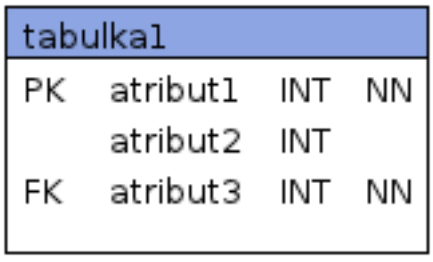
E-R diagram neboli entitně relační diagram slouží k návrhu relační databáze, vyznačení tabulek, jejich atributů, datových typů atributů a vztahů mezi tabulkami. Grafické symboly používané k vyznačování jednotlivých prvků diagramu se mohou lišit podle použité notace. Jednou z běžně používaných notací je tzv. Information Engineering Notation. Dále se zde budu zabývat pouze touto notací.

2.7.2 Návrh databáze

Návrh databáze pomocí entitně relačních diagramů může být logický nebo fyzický. Jako logický se označuje návrh, u kterého zatím není dáno, v jakém databázovém systému bude implementován. Zde například datové typy atributů nemohou být vyznačeny konkrétně, protože každý databázový systém neobsahuje stejné datové typy pro atributy. Ve fyzickém návrhu už se model vztahuje ke konkrétní implementaci v daném databázovém systému a například datové typy atributů jsou vyznačeny tak, aby korespondovaly s datovými typy v tomto systému.

2.7.3 Entita

Entita (tabulka) se v enitně relačním diagramu vyznačuje jako obdélník (někdy se používají například zaoblené rohy). V záhlaví je obvykle jméno entity a její tělo obsahuje seznam atributů s jejich datovými typy (ať už obecnými v logickém modelu nebo konkrétními ve fyzickém modelu). U atributů zde také vyznačujeme zda jsou součástí primárního (PK), cizího (FK) klíče nebo zda jejich hodnota nesmí nabývat hodnoty *null* (NN). Na obrázku 6 je příklad zobrazení entity.

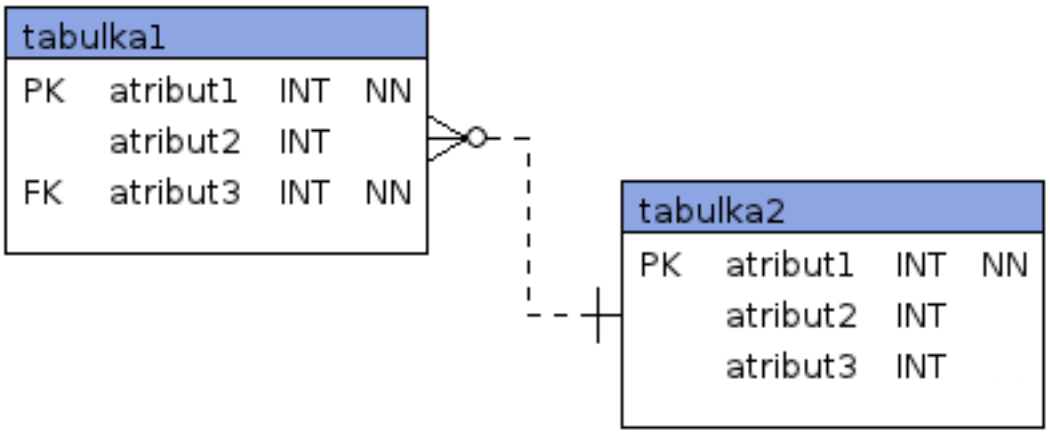


Obrázek 6 - Zobrazení tabulky v entitně-relačním diagramu

2.7.4 Relace

Relace je mezi tabulkami v entitně relačním diagramu symbolizována lomenou čarou, na jejímž každém konci vyznačujeme příslušnou kardinalitu a parcialitu. Kardinalita se vyznačuje jako kolmá spojnice na straně „jedna“ a jako trojnožka na straně „N“.

Parcialita prázdným kolečkem na straně, která se nemusí účastnit vztahu a čarou kolmou na spojnici na straně, která se musí vztahu účastnit. Příklad takového vztahu je uveden na obrázku 7.



Obrázek 7 - Zobrazení vztahu v entitně-relačním diagramu

3 Popis jazyka SQL

Při psaní této kapitoly jsem využíval vědomostí nabytých v knize SQL: Kompletní průvodce od autorů Jamese R. Groffa a Paula N. Weinberga. [2]

3.1 Jazyk SQL

SQL (Structured Query Language) je dnes nejběžnějším prostředkem ke komunikaci s relačními databázovými systémy. Je to dotazovací jazyk, jímž lze spravovat data uložená v databázovém systému, případně tento databázový systém. Jazyk SQL je ve světě relačních databází standardem. Poprvé byl standardizován v roce 1986. Dalším standardem byl SQL-92 (SQL2) v roce 1992 a zatím posledním SQL3. Běžné databázové systémy dnes tyto standardy sice do velké míry podporují, nicméně každý dodavatel si jej částečně přizpůsobuje pro své potřeby, případně přidává do jazyka SQL úplně nové konstrukce. Díky tomu nelze obecně prohlásit, že daný příkaz se, na různých databázových systémech podporujících SQL, bude chovat stejně.

Příkazy jazyka SQL jsou navrženy tak, aby byly snadno konstruovatelné a srozumitelné. Proto většina z nich připomíná anglické věty, ze kterých lehce poznáme, co dělají.

3.2 Datové typy

Datový typ je množina hodnot, jichž může sloupec označený tímto datovým typem nabývat. Pro různé druhy dat volíme různé datové typy. Dnešní databázové systémy ve své implementaci SQL často o mnoho rozšiřují množinu datových typů definovaných ve standardech.

Tabulka 1 - Datové typy definované ve standardu SQL2 [2]

Název datového typu	Stručný popis
CHARACTER	Řetězec znaků s pevnou délkou
NATIONAL CHARACTER	Řetězec národních znaků s pevnou délkou
CHARACTER VARYING	Řetězec znaků s proměnnou délkou
NATIONAL CHARACTER VARYING	Řetězec národních znaků s proměnnou délkou
BIT	Řetězec bitů s pevnou délkou
BIT VARYING	Řetězec bitů s proměnnou délkou
NUMERIC	Desetinné číslo
DECIMAL	Stejné jako NUMERIC
INTEGER	Celé číslo
SMALLINT	Celé číslo
FLOAT	Desetinné číslo
REAL	Desetinné číslo
DOUBLE PRECISCION	Desetinné číslo s vysokou přesností
DATE	Datum
TIME	Čas
TIMESTAMP	Časové razítko
INTERVAL	Časový interval

3.3 Příkazy pro definici dat

3.3.1 Rozdělení

Příkazy pro definici dat, souhrnně označovány DDL, jsou navrženy k realizaci změn ve struktuře databáze. Tyto příkazy obvykle pracují s tabulkami a sloupci, ale i s jinými objekty databáze. Tři základní příkazy jazyka DDL zahrnují:

- příkaz *CREATE*, který slouží k vytváření nových objektů,
- příkaz *DROP*, který slouží k mazání objektů,
- příkaz *ALTER*, který slouží k modifikaci existujících objektů.

Tyto příkazy se dále pojí s dalšími konstrukcemi, které blíže identifikují konkrétní objekt, se kterým se pracuje. Mezi tyto konstrukce patří například: *TABLE* (tabulka), *DATABASE* (databáze) nebo *VIEW* (pohled).

3.3.2 Vytvoření tabulky

Pro vytvoření nové tabulky v naší databázi užíváme příkazu *CREATE TABLE*, který následuje název tabulky a v závorkách uvedený seznam sloupců s jejich datovými typy, volitelně i omezením, výchozími hodnotami a dalšími vlastnostmi. Při úspěšném vykonání tohoto příkazu je založena nová tabulka s požadovanými vlastnostmi a je připravena k ukládání dat.

```
CREATE TABLE PREHLED_ZRIZENI (
      ID INTEGER NOT NULL,
      ZRIZENI VARCHAR(32) NOT NULL,
      PRIMARY KEY(ID)
)

CREATE TABLE STATY (
      KOD INTEGER NOT NULL,
      NAZEV VARCHAR(64) NOT NULL,
      POPULACE INTEGER DEFAULT 1,
      ZRIZENI INTEGER,
      PRIMARY KEY(KOD),
      CONSTRAINT STATNI_ZRIZENI
            FOREIGN KEY(ZRIZENI)
            REFERENCES(PREHLED_ZRIZENI)
            ON DELETE SET NULL
)
```

3.3.3 Změna struktury tabulky

Pokud chceme strukturu již existující tabulky modifikovat, poslouží nám k tomu příkaz *ALTER TABLE*, za který uvedeme název požadované tabulky a příslušnou operaci, kterou chceme realizovat. Tyto operace můžou být různé, od přidání nebo odebrání sloupců přes přidání nových omezení po změny v klíčových atributech.

```
ALTER TABLE STATY (
      ADD ROZLOHA FLOAT NOT NULL,
      DROP CONSTRAINT STATNI_ZRIZENI RESTRICT,
      DROP ZRIZENI
      ALTER POPULACE DROP DEFAULT
)
```

3.4 Smazání tabulky

V případě, že chceme daný objekt odstranit z databáze, použijeme příkaz *DROP*. Pro odstranění tabulky tak použijeme konstrukci *DROP TABLE*, za kterou následuje název mazané tabulky. Tabulka je tímto odstraněna z databáze včetně dat, která v sobě uchovává. Na konci příkazu je dle standardu SQL2 vyžadováno jedno z dvojice slov *CASCADE*, *RESTRICT*, na jehož základě databáze určí dopad na jiné objekty závisející na tabulce.

Následující příklad odstraní tabulku *STATY* a všechny objekty, které na ní závisejí.

```
DROP TABLE STATY CASCADE
```

3.5 Další SQL příkazy

3.5.1 Příkaz INSERT

Tento příkaz slouží v drtivé většině svého využití ke vkládání řádků do tabulky. Za slovy *INSERT INTO* uvádíme název tabulky, do které chceme vkládat data. Za název tabulky můžeme volitelně vypsat sloupce, do kterých budeme vkládat data (do sloupců, které zde nebudou uvedeny, bude automaticky vložena hodnota *NULL* – žádná data). Potom následuje klíčové slovo *VALUES*, za které opět do závorky vypíšeme hodnoty, které chceme vložit ve stejném pořadí, ve kterém jsou sloupce. Pokud seznam sloupců vynecháme, je třeba uvést hodnotu pro každý sloupec tabulky ve správném pořadí, tj. v jakém je daný sloupec definován v tabulce. Obecně po uvedení nebo neuvedení seznamu sloupců lze uvést nejen slovo *VALUES* se seznamem hodnot, ale jakoukoliv relaci (například *SELECT* vracející více řádků) obsahující data.

```
INSERT INTO PREHLED_ZRIZENI VALUES (1, 'Republika')
INSERT INTO STATY (KOD, NAZEV, ZRIZENI) VALUES (32, 'Argentina', 1)
```

3.5.2 Příkaz UPDATE

Příkaz slouží pro modifikaci řádků v tabulce. Za *UPDATE* uvádíme název tabulky, následovaný slovem *SET* a seznamem dvojic *sloupec = hodnota* oddělených čárkou. Velmi důležitou součástí dále bývá klauzule *WHERE*, za kterou uvádíme podmínku pro modifikované řádky. Po vykonání budou upraveny pouze řádky, které splňovaly tuto podmínku. V případě neuvedení podmínky se úpravy provedou pro všechny řádky tabulky.

```
UPDATE STATY SET NAZEV='Argentinská republika', ZRIZENI=NULL
      WHERE KOD=32
```

3.5.3 Příkaz DELETE

Konstrukce *DELETE FROM* následovaná názvem tabulky smaže všechny řádky z dané tabulky. Ke smazání konkrétních řádků v tabulce slouží opět klauzule *WHERE*, za kterou uvedeme podmínku, kterou mají mazané řádky splňovat.

```
DELETE FROM STATY WHERE KOD=32
```

3.5.4 Příkaz SELECT

Příkaz *SELECT* slouží k získávání dat z databáze. Za klíčovým slovem *SELECT* uvádíme seznam sloupců, které chceme vybrat. Po nich následuje slovo *FROM*, následované názvem tabulky, ze které data vybíráme. Seznam sloupců, které uvádíme (jejichž data nás zajímají), se nazývá projekcí. Pokud vybíráme data z více tabulek naráz, uvádíme na místě tabulky seznam tabulek oddělný čárkami. Sloupce v projekci je, v případě shody jejich názvu ve více tabulkách, nutno vypsat i s názvem tabulky, ke které patří. Tabulky v dotazu lze i různě spojovat, nejčastěji za pomoci cizích klíčů. Ke spojování slouží klauzule *JOIN*, případně lze využít spojení na základě restrikce.

```
SELECT POPULACE, ZRIZENI FROM STATY WHERE NAZEV='Argentina'
```

4 Zhodnocení vybraných produktů

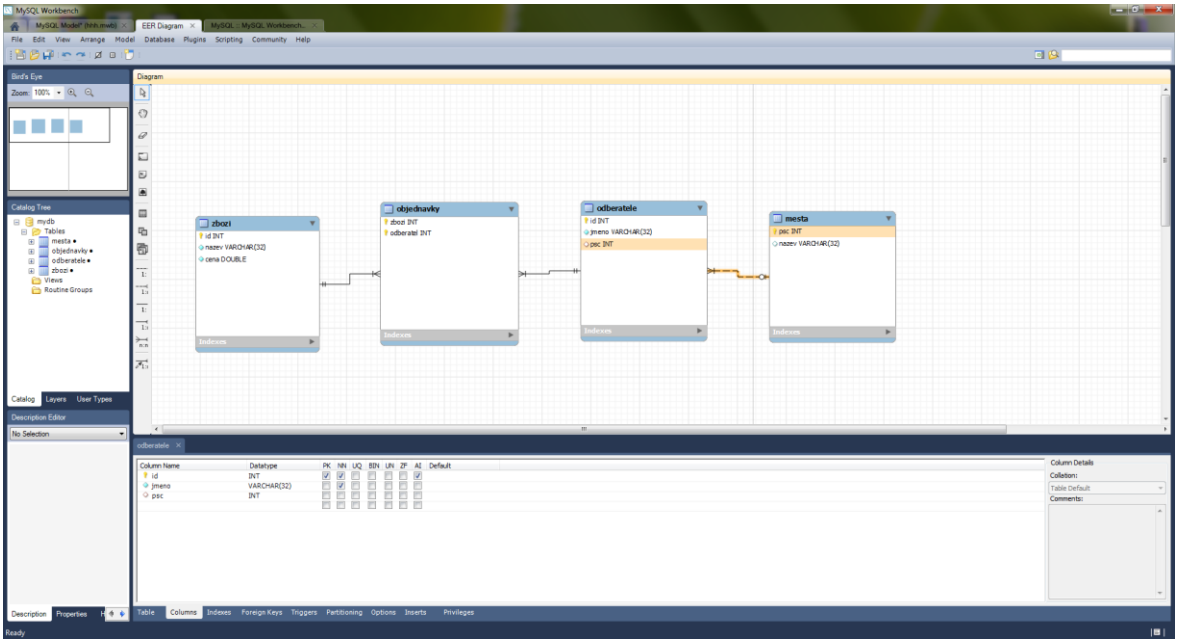
4.1 MySQL Workbench Community Edition

4.1.1 Představení produktu

MySQL Workbench 5.2.33 je komplexní nástroj od společnosti Oracle pro práci s databází MySQL. Kromě vizuálního modelování návrhu databáze umožňuje například i správu databázového serveru. Program je multiplatformní a lze jej provozovat na systémech Microsoft Windows, GNU/Linux nebo Mac OS X. Produkt existuje ve dvou verzích s odlišnými licencemi – „Community Edition“, která je plnohodnotným programem a je šířena pod licencí GPL a „Standart Edition“, která je komerční variantou nabízející více nástrojů a rozšíření. Dále zde budu hodnotit pouze komunitní verzi. [5]

4.1.2 Ovládání

Program nabízí celkem intuitivní ovládání. Tabulky lze velmi jednoduše vytvářet, stejně tak jako k nim přidávat atributy a jejich vlastnosti. Vše je realizováno pouhými několika kliky myší. Relace mezi tabulkami lze vytvářet přímo graficky nebo definováním cizího klíče. Při přejetí myší přes relaci se v tabulkách označí atributy, mezi kterými je tato relace vytvořena – obvykle cizí klíč v jedné tabulce a primární klíč ve druhé. V tabulkách jsou přehledně graficky zobrazeny primární klíče i některé vlastnosti atributů (např. *NOT NULL* jako tyrkysový kosočtverec). Na obrázku 8 je zobrazen snímek obrazovky s programem.



Obrázek 8 - MySQL Workbench Community Edition

4.1.3 Vlastnosti

Program umožňuje zobrazení návrhu v několika různých notacích. Umožňuje vytváření dalších databázových objektů – triggerů nebo pohledů. Umí reverse engineering z existujícího MySQL skriptu. Exportovat návrh umí do MySQL skriptu, případně do PNG, SVG nebo PDF formátu.

4.1.4 Celkový dojem

MySQL Workbench Community Edition nabízí veškerý komfort, který je k modelování databází potřeba. Navíc jsou jeho zdrojové kódy kompilovatelné i na jiných platformách než Microsoft Windows. Jeho největší nevýhodou je, že pracuje pouze s databází MySQL. Dále by dle mého názoru mohlo být v tabulkách vyznačeno více informací o atributech (například zda je zaškrtnuta volba auto_increment).

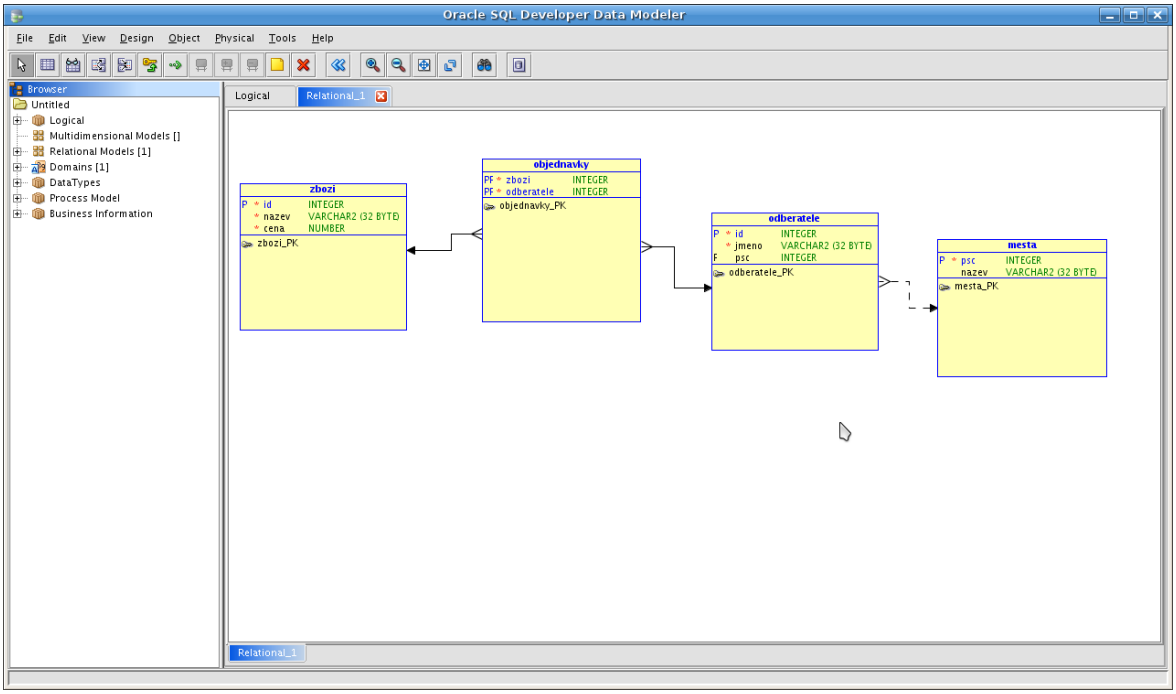
4.2 Oracle SQL Developer Data Modeler

4.2.1 Představení produktu

SQL Developer Data Modeler je samostatná aplikace, která ve své podstatě doplňuje SQL Developer, sloužící ke správě databázového serveru Oracle, o grafické modelování databází. Existuje ve verzích pro Microsoft Windows, GNU/Linux i Mac OS X. Je šířen jako freeware. [6]

4.2.2 Ovládání

Navrhovat databázi zde uživatel může ve dvou hlavních módech – logický a relační. V logickém návrhu lze vytvořit model databáze, který je z pohledu relačního modelu dat obecný. Relační model oproti tomu vykazuje rysy databázových systémů používaných v praxi. Mezi oběma modely se nechá převádět. V obou případech je navrhování tabulek jednoduché. Vztahy lze opět vytvářet interaktivně nebo přes cizí klíče. Uživatelsky ne příliš přívětivé je přidávání atributů tabulky – nutno otevřít dialogové okno. Celkové ovládání se mi jeví spíše jako chaotické. Obrázek 9 zachycuje pracovní prostředí programu během relačního návrhu.



Obrázek 9 - Oracle SQL Developer Data Modeler

4.2.3 Vlastnosti

Program nabízí obvyklé funkce mezi které patří vygenerování skriptu a reverse engineering. Tentokrát lze skript vygenerovat pro několik různých databází zahrnujících několik verzí Oraclu. Nicméně chybí například známé MySQL nebo PostrgreSQL. Exportovat lze i do několika dalších formátů. Grafickou podobu návrhu můžeme uložit do formátů PNG, SVG nebo PDF.

4.2.4 Celkový dojem

Ačkoliv graficky vypadá program na první pohled velice stroze, umožňuje v podstatě vše, co se od něho dá očekávat. Chybí pouze podpora pro několik často používaných databází. Samotný návrh by mohl být rychlejší.

4.3 Shrnutí vlastností produktů

Tabulka 2 přehledně zobrazuje vlastnosti jednotlivých hodnocených produktů a dále plánované vlastnosti pro navrhovanou aplikaci.

Tabulka 2 - Shrnutí vlastností produktů

	MySQL Workbench Community Edition	Oracle SQL Developer Data Modeler	Navrhovaný nástroj
Multiplatformní	Ano	Ano	Ano
Tvorba komplexních omezení	Ano	Ano	Ne
Export do DDL	Ano	Ano	Ano
Export do PNG	Ano	Ano	Ano
Reverse Engineer	Ano	Ano	Ano
Podpora různých DBMS	Ne	Některých	Ano

5 Návrh základních tříd nástroje

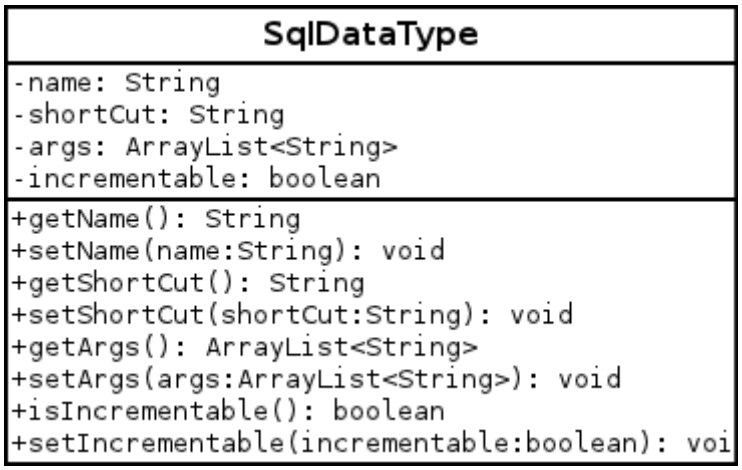
5.1 Třída pro datový typ

Datový typ databáze je základní třídou, kterou je třeba navrhnout. Pro účely snadné tvorby databáze budu předpokládat, že datový typ, v nástroji umožňující modelování databází, bude reprezentován třídou mající jako atributy následující položky:

- Název atributu,
- zkrácený název atributu,
- počet argumentů datového typu,
- seznam názvů argumentů datového typu,
- příznak, zda je pro atribut možno využít volu automatické inkrementace.

Jelikož implementačním nástrojem použitým k vytvoření vlastního programu umožňující datové modelování bude programovací jazyk Java, využiji jeho možností a v případě potřeby budu volit komplexní datové typy, které nabízí ve svých knihovnách. Dále předpokládám následnou manipulaci se stavem této struktury. Proto bude obsahovat i metody poskytující a nastavující stav.

Pro název atributu, stejně jako pro jeho zkrácený název využiji typ *String* reprezentující řetězec znaků. Seznam názvů argumentů bude reprezentován typem *ArrayList*. *ArrayList* je datová struktura dynamického pole implementující rozhraní spojového seznamu. V mém případě bude pracovat s typem *String*, protože jednotlivé názvy parametrů budou řetězce. Poslední položkou je podpora automatické inkrementace, kterou zde bude vyjadřovat, respektive nevyjadřovat, datový typ *boolean*. Diagram třídy je zobrazen na obrázku 10.



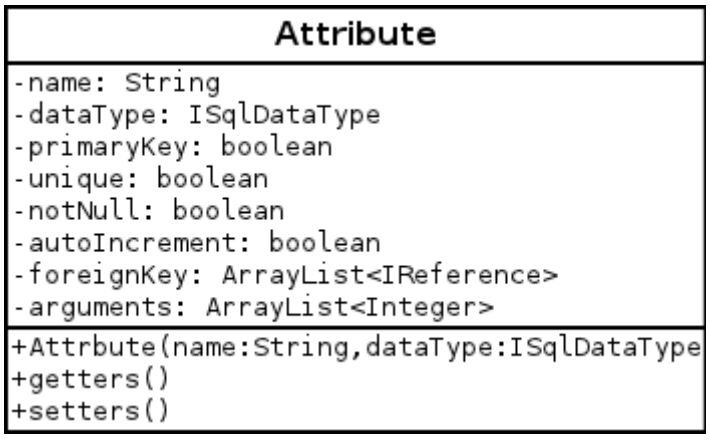
Obrázek 10 - Diagram třídy SqlDataType

5.2 Třída pro atribut tabulky

Atribut tabulky bude reprezentovat třída (respektive instance této třídy), která bude reflektovat vlastnosti atributu stavem svých následujících atributů:

- Jméno atributu tabulky,
- reference na datový typ atributu,
- příznak, zda je daný atribut součástí primárního klíče tabulky,
- příznak, zda je daný atribut unikátní,
- příznak, zda je daný atribut nenulový,
- příznak, zda lze u atributu nastavit automatickou inkrementaci,
- seznam relací, ve kterých atribut vystupuje jako cizí klíč,
- seznam argumentů.

V jazyce Java se pro název opět jeví jako nejrozumnější varianta využití datového typu *String*. Reference na datový typ tohoto atributu je dána strukturou, jež je uvedena v bodě 4.1. Všechny příznaky, které atribut tabulky uchovává jsou jednoduchého logického datového typu *boolean* a mohou tedy nabývat hodnot pravda nebo nepravda. Seznam všech relací, kterých se atribut účastní jako cizí klíč lze uchovávat jako strukturu *ArrayList* pracující s typem *IReference*, což je rozhraní pro třídu *Reference*, která bude definována v bodě 4.4. Seznam argumentů datového typu bude opět reprezentován strukturou *ArrayList*, nicméně tentokrát bude pracovat s celočíselným datovým typem *Integer*, protože jako argumenty očekáváme celá čísla. Na obrázku 11 je zachycen diagram třídy *Attribute*.



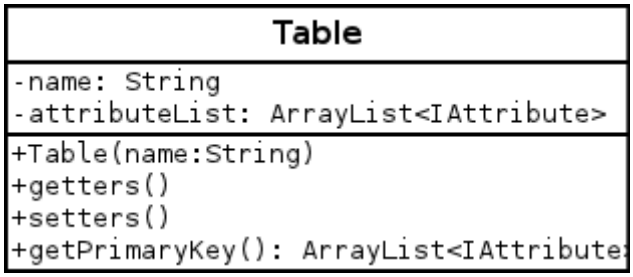
Obrázek 11 - Diagram třídy Attribute

5.3 Třída pro tabulku

Databázová tabulka z pohledu nástroje pro modelování databází nemá příliš mnoho vlastností, které je třeba uchovávat. Mezi ně patří:

- Název tabulky,
- seznam atributů tabulky.

Nic nebrání použít pro název opět datový typ *String* a pro seznam atributů opět strukturu *ArrayList* pracující s rozhraním k atributu tabulky popsanému v bodě 4.2. Diagram třídy *Table* je na zobrazen na obrázku 12.



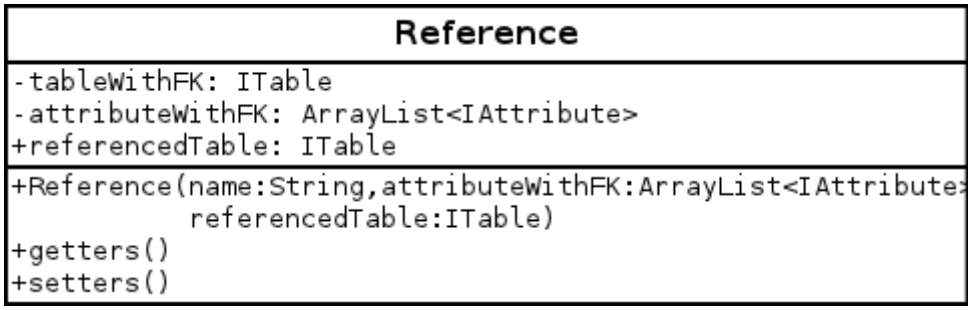
Obrázek 12 - Diagram třídy Table

5.4 Třída pro relaci

Relace neboli přesněji vztah je jeden z nejdůležitějších rysů relačních databází. Definuje spojení dvou tabulek na základě cizího klíče v jedné a primárního klíče ve druhé tabulce. Popis vztahu pro aplikaci bude realizován třídou fixující tyto atributy:

- Tabulka, ze které se odkazuje,
- seznam atributů tvořících cizí klíč,
- tabulka, na kterou se odkazuje.

Atribut uchovávající referenci na tabulku s cizím klíčem bude datového typu, který je rozhraním pro třídu *Table*, definovanou v bodě 4.3. Analogicky i tabulka, na kterou se odkazuje, musí být stejného datového typu. Seznam atributů bude představovat znovu *ArrayList* pracující s rozhraním pro typ *Attribute* z bodu 4.2. Na obrázku 13 je zobrazen diagram třídy *Reference*.



Obrázek 13 - Diagram třídy Reference

6 Implementace nástroje pro databázové modelování

6.1 Volba implementačního nástroje

Programovacích jazyků a různých frameworků, ať komerčních nebo volně dostupných je dnes k dispozici celá řada. Při volbě konkrétní varianty je důležité předně zvážit povahu problému a podporu jeho řešení v daném programovacím jazyce. Neméně důležitá je dostupnost dokumentace, stejně jako dostupnost rozšíření a knihoven. Tvorba vlastního programu bude závislá na technologiích, které jsou pro daný jazyk nebo platformu dostupné.

Program bude v interaktivním grafickém režimu, proto je třeba volit jazyk, kterému není cizí práce s grafikou. Naštěstí u většiny dnešních moderních programovacích jazyků je podpora grafiky (ať už nativní nebo s pomocí frameworků) standardem. Grafikou je zde myšlena tvorba formulářových aplikací, obsluha událostí a vykreslování dvojrozměrných grafických primitiv.

Z hlediska návrhu aplikace není striktně vyžadován objektový model. Nicméně z hlediska znovupoužitelnosti kódu, přehlednosti a bezpečnosti je objektový přístup takřka nutností.

Mezi některé dnes nejrozšířenější technologie určené k programování (mimo jiné) desktopových formulářových aplikací patří:

- Java Standard Edition (Oracle),
- v podstatě libovolný jazyk platformy Microsoft .NET,
- QT framework (Nokia),
- open source knihovna GTK+,
- knihovna Microsoft Foundation Class,
- Borland Delphi.

Značnou nevýhodou při použití platformy Microsoft .NET je nepřenositelnost výsledné aplikace na jiné operační systémy. Toto se snaží změnit projekt Mono, který je jakousi portací knihoven Microsoft .NET na jiné operační systémy, především unixového typu. Ovšem funkčnost některých takto portovaných programů je značně nízká. Stejný problém, nepřenositelnost na jiné platformy, nastává při použití Microsoft Foundation Class, což je vlastně knihovna zastřešující Windows API. Dá se tedy očekávat, že na jiných operačních systémech než Microsoft Windows je výsledný program nespustitelný, v lepším případě nepoužitelný.

QT framework, GTK+ a Java jsou multiplatformními nástroji, přičemž u prvních dvou lze zdrojový kód vytvořené aplikace zkompilovat na různých operačních systémech.

V případě platformy Java lze díky virtuálnímu stroji spouštět již přeložené zdrojové kódy v podstatě na jakémkoliv operačním systému, který má virtuální stroj nainstalován.

Přehled zmíněných technologií je zobrazen v tabulce 3.

Tabulka 3 - Přehled technologií vhodných pro implementaci nástroje

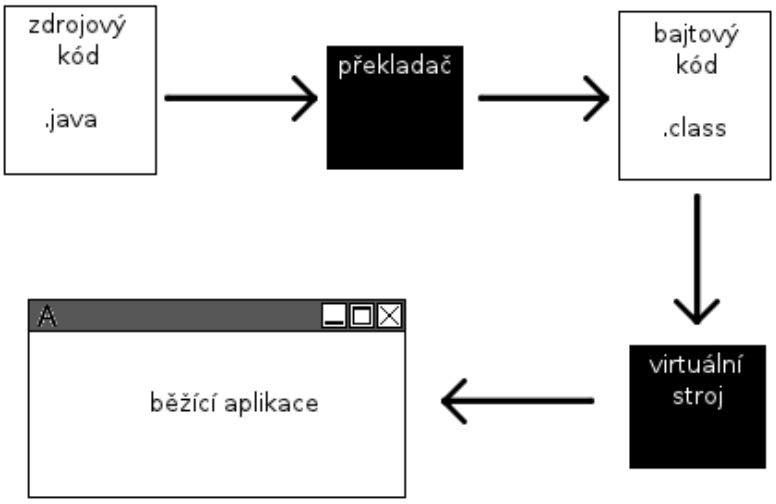
Technologie	Programovací jazyk	Multiplatformní
Java Standard Edition	Java	Ano
Microsoft .NET	C#, C++, VB a další	Částečně
QT framework	C++, Python a další	Ano
GTK+	C++, Python a další	Ano
MFC	C++	NE
Borland Delphi	Object Pascal	NE

Za implementační nástroj volím platformu Java Standard Edition. Jazyk Java je intuitivní, má jednoduchou syntaxi a je silně objektový. Další výhodou je, že obsahuje velké množství knihoven. Program vytvořený v jazyce Java bude přenositelný na různé operační systémy bez nutnosti nového sestavení. Zdrojové kódy se přeloží do mezikódu, který je pak za pomoci virtuálního stroje spustitelný na různých operačních systémech a dokonce i různých hardwarových platformách.

6.2 Java Standard Edition

Jazyk Java byl vyvinut společností Sun Microsystems jako silně objektový jazyk, který je přenositelný na různé platformy. V roce 2009 byla společnost Sun Microsystems koupena společností Oracle. V současnosti se Java nachází ve verzi 6, avšak Oracle plánuje vydání verze 7.

Zdrojové kódy mají v jazyce Java příponu *.java*. Při překladu vznikají soubory mezikódu, které mají příponu *.class*. Tento mezikód označovaný jako bajtový kód (bytecode) dokáže interpretovat a spustit virtuální stroj nainstalovaný na kterémkoliv operačním systému, pro který tento existuje. Tento proces graficky zachycuje obrázek 14.



Obrázek 14 - Překlad do bajtového kódu

6.3 Vývojové prostředí

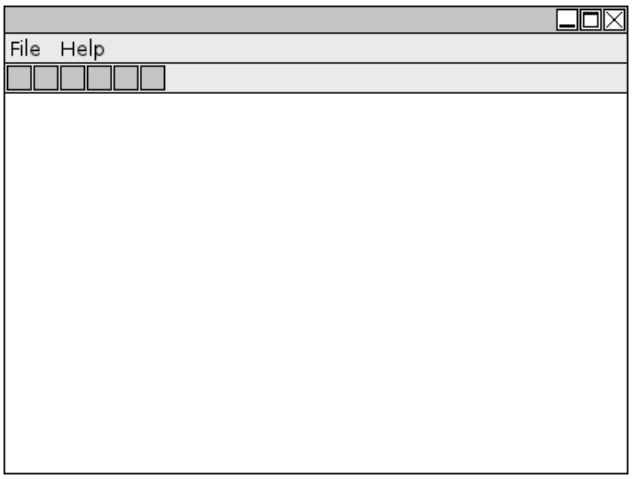
Žádná aplikace většího rozsahu se nedá dobře programovat bez využití služeb integrovaného vývojového prostředí (IDE). Zjednodušení spočívá v integraci překladače, textového editoru a obvykle i debuggeru do jedné aplikace, přičemž některé tyto mají zabudované i další komplexní nástroje pro pohodlný vývoj. Pro jazyk Java existuje mnoho takových vývojových prostředí. Jedním z nich je i NetBeans IDE od společnosti Oracle.

NetBeans IDE je nástroj, který k obvyklým funkcím vývojového prostředí přidává i některé pokročilé služby. Mezi tyto patří zejména interaktivní návrhář uživatelského prostředí, našeptávač při psaní kódu, generátor kódu a další. Program, ve kterém budu aplikaci vyvíjet je ve verzi 6.9.1.

6.4 Hlavní okno aplikace

Většinu aplikací uživatel hodnotí dle prvního dojmu. Ovládání je na první pohled buďto chaotické nebo přehledné. Až po nějaké době užívání programu uživatel přijde na jeho skutečné přednosti a slabiny.

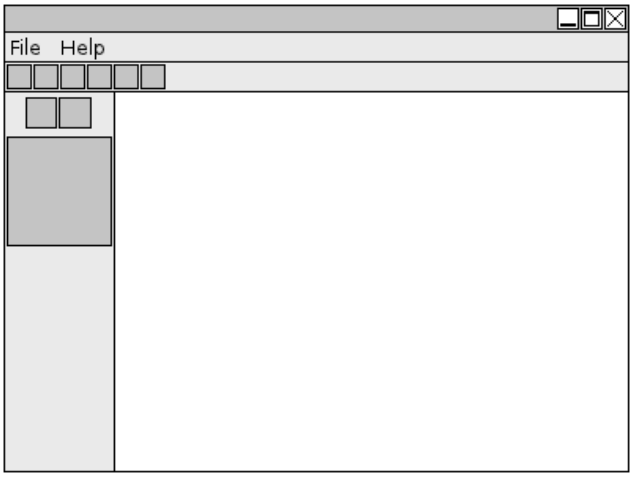
Okenní aplikace se často drží standardního vzhledu. U značné části z nich nalezneme menu v horních partiích okna, pod ním nástrojovou lištu a zbytek okna vyplňuje pracovní plocha. Mnoho vývojářů volí tento model z prostého důvodu – uživatelé jsou na něj již zvyklí. Standardní model formulářové aplikace ukazuje obrázek 15.



Obrázek 15 - Tradiční rozložení formulářové aplikace

Aby vytvářená aplikace byla uživatelsky přívětivá, je třeba tento systém z velké části zachovat. Nicméně kvůli rychlosti a efektivitě návrhu je třeba velké množství ovládacích prvků umístit do prostoru nástrojových lišt, aby byly co nejrychleji k dispozici.

Aplikace bude tedy používat systém se dvěma nástrojovými lištami – jedna tradiční horizontální, umístěná hned pod hlavní menu, druhá svislá, umístěná vlevo. Zbytek aplikace bude vyplňovat pracovní plocha. Systém s vertikální lištou nalevo, dle mého názoru, lépe využívá prostor na monitorech a displejích, které mají poměr stran 16:10 nebo 16:9. Návrh také umožňuje umístění daleko větších komponent ve vertikálním směru, než jaké by byly možné v horizontální liště. Tento návrh je zobrazen na obrázku 16.



Obrázek 16 - Rozložení pro vytvářený nástroj

Dále je třeba rozhodnout, jaké ovládací prvky na tyto lišty umístit. Rozhodně se bude jednat o akce, které přímo souvisejí s návrhem a které budou často využívané. Proto zde umístím prvky pro vytváření tabulek, vytváření a odebírání atributů a manipulaci s atributy.

V kódu je hlavní okno realizováno třídou odvozenou od třídy *JFrame*, ve které vytváříme instance dalších jednotlivých komponent. O tyto základní akce se postará interaktivní návrhář grafického uživatelského rozhraní (GUI).

6.5 Pracovní plocha

Požadavky na pracovní plochu jsou prosté – musí být dostatečně velká. Není mnoho možností jak toto zařídit na malých monitorech, a tak nezbývá než vytvořit plochu, kterou lze v případě potřeby posunout. Řešením tohoto případu v jazyce Java je vytvoření komponenty *ScrollPane* (například v grafickém návrháři), na kterou umístíme komponentu představující pracovní plochu.

```
scrollPane.setViewportView(panelModel);
```

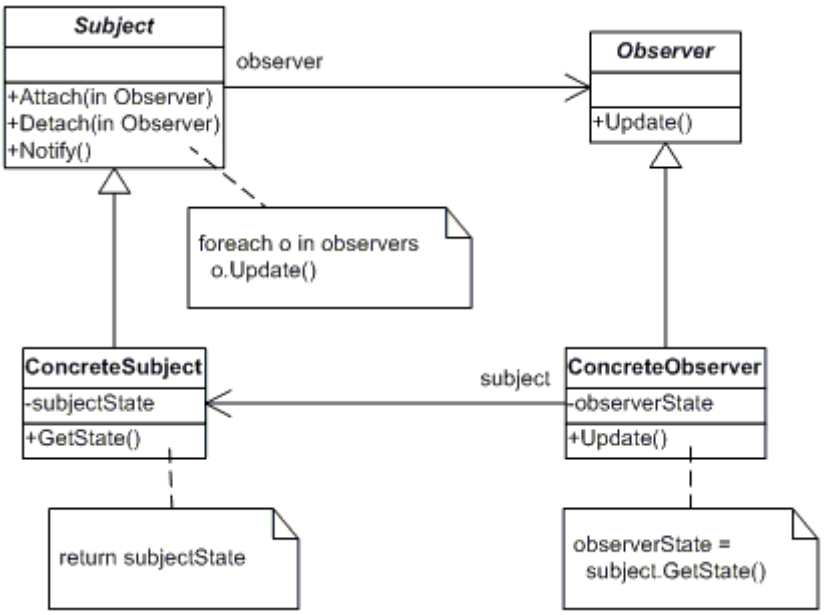
Pracovní plochu bude představovat instance třídy odvozené od komponenty *JPanel*. Třída bude pojmenována *PanelModel* a její instance *panelModel*. Na tuto komponentu (potomka *JPanelu*) lze vykreslovat grafická primitiva nebo umisťovat další komponenty.

6.6 Komunikace mezi komponentami

Důležitým krokem v návrhu je zajištění komunikace mezi jednotlivými komponentami. Ať už mezi kreslícím plátnem a hlavním oknem nebo mezi budoucími komponentami představujícími tabulky a zbytkem grafické části aplikace.

Předkem každé z komponent je abstraktní třída *Component*, která fixuje některé důležité vlastnosti grafické komponenty. Hlavně ale implementuje veřejnou metodu *getParent*, která vrací komponentu, na níž je aktuální komponenta umístěná. Tuto metodu tedy každá komponenta dědí a má tak prostředky ke komunikaci s předky ve smyslu rozložení komponent.

Jelikož tento přístup znamená mnoho přetypování a počítání hierarchií předků, rozhodl jsem se zvolit návrhový vzor Observer, který je určen ke komunikaci jednoho odesílatele a několika příjemců. S použitím návrhového vzoru Observer, stačí aby si každá nově vznikající komponenta zaregistrovala všechny ty, které mají být ovlivněny stavem této komponenty. Pak už stačí zavolat metodu notifyObservers a všichni zaregistrovaní posluchači dostanou zprávu. Na obrázku 17 je zobrazen diagram návrhového vzoru Observer.



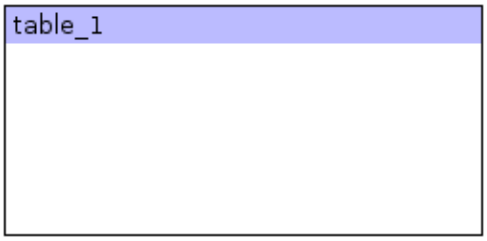
Obrázek 17 - Diagram návrhového vzoru Observer [1]

Situaci tu značně znepříjemní jednoduchost programovacího jazyka Java. Jedná se o nemožnost třídy dědit zároveň od více tříd. Návrhový vzor Observer má v Javě svou vlastní implementaci a stačí ho tedy jen využít. Jenže grafické komponenty, se kterými pracujeme, dědí ze svých grafických rodičů. Třída *Observable*, ze které by měl dědit producent zpráv se tedy rodičem jen tak nestane. Řešením, které jsem zvolil, je vytvořit v producentovi instanci třídy *Observable* a referenci na ní držet v nějakém atributu. Třídy, jejichž instance mohou být registrovány jako posluchači událostí, musí implementovat rozhraní *Observer*.

6.7 Vytváření tabulek

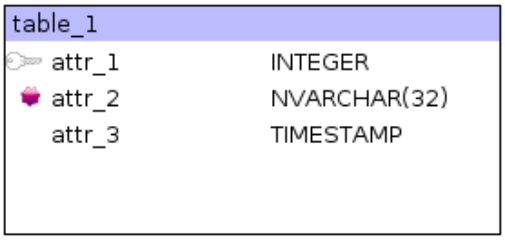
Třída pro uchování databázové tabulky je již navrhnutá (bod 4.3). Nyní je třeba navrhnout grafickou reprezentaci pro data v této tabulce. Toto zajistí nová třída, která bude dědit od abstraktní třídy *JComponent*. Tím je zajištěno, že instance této třídy budou grafické komponenty se všemi vlastnostmi, které má taková komponenta mít. Důležité je, aby třída měla referenci na tabulku, kterou představuje.

Dalším krokem je určit, jak se má tato komponenta vykreslit. To lze provést přepsáním metody *PaintComponent*, která je implementována v abstraktní třídě *JComponent*. V metodě se dá za pomoci instance třídy *Graphics* kreslit na libovolné souřadnice komponenty různá grafická primitiva. Posloupností několika příkazů dostane tabulka vzhled tabulky. Tato tabulka je zobrazena na obrázku 18.



Obrázek 18 - Grafická reprezentace tabulky

Grafická reprezentace atributů tabulky bude řešena dalšími komponentami, na kterých bude umístěno textové pole představující název tabulky, textové pole zobrazující datový typ a nakonec grafická reprezentace vlastností atributů, která bude řešena vykreslováním obrázků. Jednotlivé komponenty atributu budou umisťovány pod sebe. Na obrázku 19 je zobrazena tabulka se třemi atributy.



Obrázek 19 - Grafické reprezentace atributu

Grafická reprezentace tabulky by nebyla nic platná, pokud by se nedala v programu dynamicky vytvořit včetně tabulky samotné. Zde napomůže událostní systém Javy.

Komponentám lze registrovat posluchače událostí. Po vzniku takovéto události se předá zpráva o události registrovanému posluchači. V tomto případě je třeba registrovat posluchače událostí myši, jehož rozhraní se nazývá *MouseListener*. Řešením, které jsem zvolil, je vytvořit metodu třídy *PanelModel*, která vrací instanci anonymní třídy postavené na třídě *MouseAdapter*. *MouseAdapter* je abstraktní třída implementující rozhraní *MouseListener*, která má všechny implementované metody prázdné. Využívám ji především k úspoře kódu, protože psaní nové anonymní třídy přes rozhraní *MouseAdapter* by mě donutilo implementovat všechny jeho metody. V konstruktoru třídy *PanelModel* je potom zaregistrována jako posluchač instance, kterou vytvoří a vrátí tato metoda.

```
private MouseListener returnPanelModelMouseListener() {
      return new MouseAdapter() {
            @override
            public void mouseClicked(MouseEvent e) {
                  // obsluha události kliknutí
            }
      };
}
```

Klik na pracovní plochu nyní emituje signál, který je obsloužen metodami, jež si přepíši v těle anonymní třídy. Obsluha události kliknutí je zajištěna metodou *mouseClicked*, která v parametru dostává informace o události ve struktuře *MouseEvent*. Lze tedy jednoduše zjistit, v jaké pozici bylo kliknuto.

V případě, že je stisknuto tlačítko k vytvoření tabulky a bude volána tato metoda, vytvoří se nová komponenta představující tabulku o nějaké velikosti a bude umístěna v místě kliku.

Atributy tabulky lze přidávat stiskem odpovídajícího tlačítka na horní nástrojové liště. Nový atribut se přidá do aktivní tabulky. Aktivní tabulka je ta, na kterou bylo naposled kliknuto myší – atribut *active* této komponenty se nastaví na *true*, pracovní plocha dostane požadavek na překreslení a komponenta bude vykreslena jako zvýrazněná.

6.8 Editace tabulek

Další potřebou je snadná editace názvu tabulky, názvu atributu, datového typu atributu a vlastností atributu. Opět k tomu využiji událostní systém a reagování posluchačů na konkrétní akce.

Zde se hodí metody, které budou reagovat na kliknutí pravým tlačítkem myši na název atributu nebo na název tabulky. V případě, že se takto stane, místo textového popisku se zobrazí editační pole, kde bude možno název tabulky nebo atributu editovat.

Datový typ bude editovatelný v seznamu datových typů na levém nástrojovém panelu. Tam budou standardní typy jazyka SQL2, ze kterých bude možno vybrat libovolný. Druhý seznam bude obsahovat datové typy specifické pro konkrétní databázi, jejíž modul si uživatel do programu načte.

Další vlastnosti atributu budou editovatelné tlačítky v horizontální nástrojové liště. Po kliknutí na atribut se tato tlačítka zpřístupní a nastaví se do takové polohy, aby vystihovaly rysy atributu. Uživatel bude tlačítka moci zamáčknout či vytáhnout podle potřeby, což u daného atributu způsobí změnu jeho vlastností.

Změnu pořadí atributů zajistí opět tlačítka na horizontálním toolbaru, která se zpřístupní po označení atributu. Pomocí tlačítek lze označený atribut posouvat nahoru nebo dolu a měnit tak výsledné pořadí atributů v tabulce.

Událostní systém bude řešit i přesouvání a změnu velikosti tabulek. Zde je třeba využít posluchačů pohybu myši, kteří jsou představováni rozhraním *MouseMotionListener*. Po implementaci metody *mouseDragged* bude mít uživatel možnost myší přesouvat tabulku do libovolného místa na pracovní ploše nebo tabulku zvětšovat a zmenšovat.

```
@Override
public void mouseDragged(MouseEvent e) {
      Point newPosition;
      newPosition = getLocation();
      newPosition.translate(
            e.getX() - lastMousePressedPoint.x,
            e.getY() - lastMousePressedPoint.y);
      if (newPosition.x < 0) {
            newPosition.x = 0;
      }
      if (newPosition.y < 0) {
            newPosition.y = 0;
      }
      setLocation(newPosition);
      getParent().repaint();
}
```

6.9 Vytváření vztahů

Vytvoření vztahu bude řešeno tak, aby bylo pro uživatele co možná nejsnazší. Uživatel si vybere atributy, které mají být součástí cizího klíče, následně klikne na ikonu cizího klíče a označí tabulku, na kterou se má odkazovat.

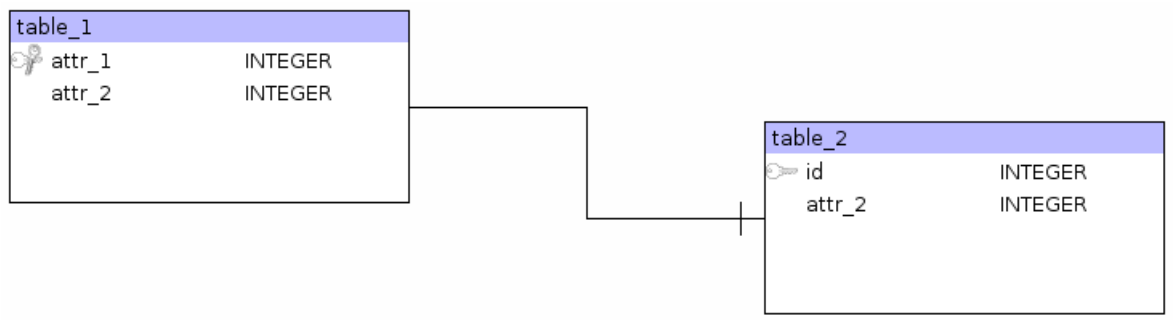
V případě nesrovnalostí mezi počtem atributů cizího klíče a počtem atributů primárního klíče v odkazované tabulce se zobrazí zpráva, informující o této chybě a vztah se nevytvoří. Podobně tomu bude v případě, že nebudou souhlasit datové typy odpovídajících atributů.

```
if (!dataTypeMatch) {
      JOptionPane.showMessageDialog(
            getParent(), "Primary key data type does not match.");
      // další kód
}
```

Při označování jednotlivých atributů (pomocí držení klávesy CTRL) se tyto ukládají do seznamu, který je atributem třídy *PanelModel*. Tyto atributy jsou zde uvedeny v pořadí, v jakém byly vybrány. Test shody datových typů probíhá v pořadí vybrání atributů s pořadím atributů primárního klíče v odkazované tabulce.

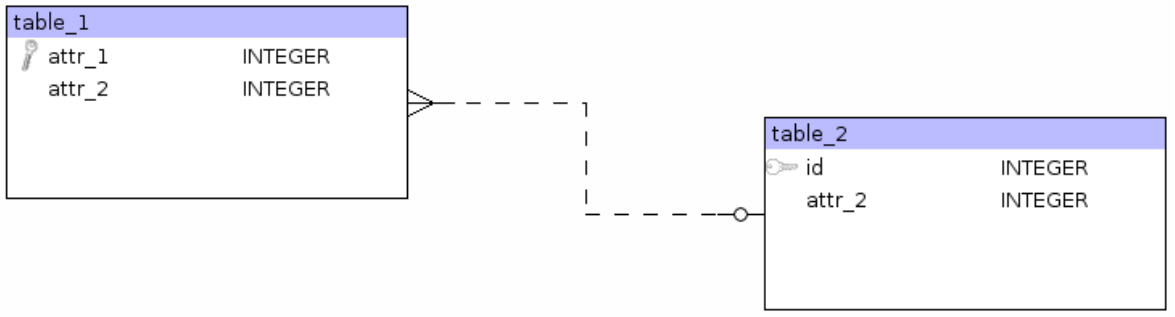
Každý vztah je reprezentován strukturou, která uchovává referenci na vztah definovaný v bodě 4.4. Dále bude uchovávat seznam manipulačních bodů, aby se čára symbolizující referenci dala přesouvat a zalomovat podle potřeby návrhu. Je-li tento seznam prázdný, vztah se vykreslí automaticky podle aktuálního umístění tabulek.

Vzhled vztahu z databázového pohledu je dán automaticky stavem atributů cizího klíče. Na základě tohoto stavu se vykreslí obrázek symbolizující stranu „jedna“ nebo stranu „N“ u databázového vztahu. V případě, že je tedy například cizí klíč zároveň celým primárním klíčem tabulky, vztah se na straně tabulky cizího klíče vykreslí jako jednoduchá čára a na straně odkazované tabulky se vykreslí kolmá čára signalizující, že vztah je pro tuto stranu povinný. Tuto situaci zachycuje obrázek 20.



Obrázek 20 - Grafická reprezentace vztahu jedna k jedné

V případě, že cizí klíč není primárním klíčem, není unikátní a není ani nenulový, vztah bude vykreslen „trojnožkou“ na straně tabulky s cizím klíčem a nepovinností účastnit se vztahu na straně odkazované tabulky. Tuto situaci zobrazuje obrázek 21. Jelikož je tato relace neidentifikující, vykreslí se vztah přerušovanou čarou.



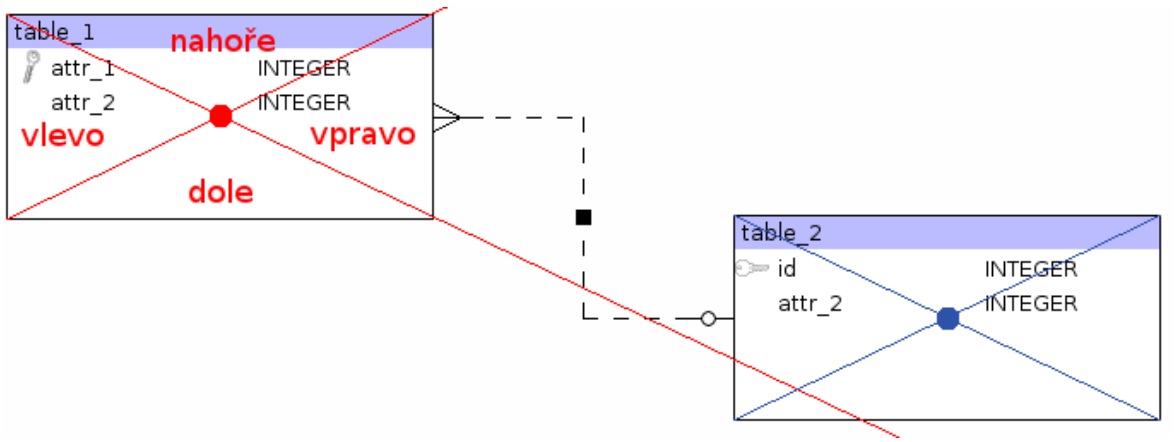
Obrázek 21 - Grafická reprezentace vztahu jedna k N

Další obslouženou událostí je například najetí myší nad atribut, který je součástí cizího klíče. Pokud se tak stane, zvýrazní se ostatní atributy tvořící tento cizí klíč a zároveň dojde ke zvýraznění vztahu, který tomuto cizímu klíči odpovídá. Je-li atribut součástí více cizích klíčů, zvýrazní se všechny tyto cizí klíče a všechny odpovídající vztahy.

6.10 Vedení vazby

Za vykreslování vztahů je zodpovědná instance třídy *PanelModel*, tedy komponenta reprezentující pracovní plochu. V místech manipulačních bodů vykresluje čtverce, které reagují na tah myší a přemísťují se do požadované polohy. Po kliku pravým tlačítkem myši na konkrétní manipulační bod se zobrazí nabídka, která dává na výběr mezi odstraněním daného manipulačního bodu a přidáním nového. Odstraněním všech manipulačních bodů se zobrazení vazby řídí automatickým výpočtem mezi polohami tabulek. V případě automatického výpočtu a vykreslení vazby bez manipulačních bodů je vykreslen uprostřed vazby černý čtverec. Kliknutím pravým tlačítkem na něj se vloží první manipulační bod.

Automatické vykreslování vazby je řízeno výpočtem, který porovnává pozici středů tabulek účastnících se vztahu. Jako výchozí tabulka je brána tabulka obsahující cizí klíč. Na základě souřadnic rohů tabulky je okolní prostor rozdělen na čtyři nestejně velké části. Po zjištění, v jaké z těchto částí se nachází střed „druhé“ (odkazované) tabulky, se vykreslí kolmice směrem doleva, doprava, nahoru nebo dolů od středu kraje tabulky s cizím klíčem do poloviny vzdálenosti k odkazované tabulce. Od středu bližšího kraje odkazované tabulky se taktéž vykreslí kolmice do poloviny vzdálenosti ke kraji tabulky s cizím klíčem. Nakonec se tyto kolmice spojí a doprostřed této spojnice se vykreslí náhrada manipulačního bodu (černý čtverec). Proces rozhodování je zachycen na obrázku 22.



Obrázek 22 - Způsob výpočtu automatického vedení vazby

Rozhodnutí, zda se střed odkazované tabulky nachází v té či jiné části rozdělené obrazovky, se uskuteční dále popsaným způsobem.

Nejprve je třeba získat rovnice obou přímek protínající protilehlé rohy tabulky. Pro přímku, která protíná levý horní roh a pravý dolní roh to je:

$$P_1: y = a \cdot x + c \tag{1.1}$$

$$CY = a \cdot CX + c \tag{1.2}$$

$$TY = a \cdot (TX + W) + c \tag{1.3}$$

kde CX – je x-ová souřadnice středu tabulky, lze zapsat jako $CX = TX + \frac{W}{2}$,

CY – je y-ová souřadnice středu tabulky, lze zapsat jako $CY = TY + \frac{H}{2}$,

TX – je x-ová souřadnice levého horního rohu tabulky,

TY – je y-ová souřadnice levého horního rohu tabulky,

W – je šířka tabulky,

H - je výška tabulky.

Odečtením rovnic dostáváme:

$$a = -\frac{H}{W} \tag{1.4}$$

$$c = TY + H \cdot \left(1 + \frac{TX}{W}\right) \tag{1.5}$$

a výsledná rovnice této přímky je:

$$P_1: -\frac{H}{W} \cdot x + TY + H \cdot \left(1 + \frac{TX}{W}\right) - y = 0. \tag{1.6}$$

.

Rovnici přímky, která protíná levý dolní roh a pravý horní roh lze získat:

$$P_2: y = a \cdot x + c \tag{2.1}$$

$$CY = a \cdot CX + c \tag{2.2}$$

$$TY = a \cdot TY + c. \tag{2.3}$$

Odečtením rovnic se získá:

$$a = \frac{H}{W} \tag{2.4}$$

$$c = TY - \frac{H}{W} \cdot TX \tag{2.5}$$

a rovnici této přímky lze potom zapsat jako:

$$P_2: \frac{H}{W} \cdot x + TY - \frac{H}{W} \cdot TX - y = 0. \quad (2.6)$$

Při překreslování plátna se pak do obou rovnic dosadí souřadnice středu odkazované tabulky a na základě výsledků porovnaných s nulami se rozhodne, na kterou stranu se vztah bude vykreslovat.

Na horní nástrojové liště se nachází tlačítko, které dokáže skrýt všechny zobrazené manipulační body. To může být výhodné zejména před exportem do obrázkového formátu PNG. V kódu je toto řešeno tak, že *PanelModel*, při překreslování plochy, vykreslení těchto bodů jednoduše vynechá. Stejně tak přestane reagovat na události kliku pravým i levým tlačítkem myši do prostoru bodů.

6.11 Mazání vztahů

Zrušení vztahu je realizováno tlačítkem v horní nástrojové liště. V případě, že je označen atribut, který je součástí alespoň jednoho cizího klíče, tlačítko je zpřístupněno.

Klik na tlačítko vyvolá zrušení vztahu, jehož se označený atribut účastní v případě, že je tento pouze jeden. V případě, že je atribut součástí více cizích klíčů a tím i více vztahů, zobrazí se dialogové okno se seznamem vztahů, kterých se atribut účastní. Uživatel má možnost označit vztah, který chce zrušit a potvrdit jeho zrušení.

```
void unsetForeignKey() {
      // kód
      if (referencesForThisFK.isEmpty()) {
            return;
      } else if (referencesForThisFK.size() == 1) {
            // zrušení vztahu
      } else {
            ReferenceChooser rch = new ReferenceChooser(
                                          referenceForThisFK)
            if (JOptionPane.showConfirmDialog(…, rch) == 0){
                  // zrušení vybraného vztahu
            }
      }
}
```

6.12 Export do PNG

Pokud programátor navrhuje nějakou databázi, často potřebuje její grafický model v některém z grafických formátů. Může to být důležité z důvodů tvorby dokumentace, vysvětlení vztahů v rámci vývojového týmu, apod. Navrhovaný nástroj tedy bude disponovat možností exportu do grafického formátu PNG (portable network graphics).

Exportovat se bude za pomoci třídy *ImageIO* a její metody *write*, která umí zapsat do souboru jako obrázek instance tříd implementujících rozhraní *RenderedImage*. Takovou

třídou je například třída *BufferedImage* (třída *BufferedImage* neimplementuje přímo rozhraní *RenderedImage*, ale implementuje rozhraní *WritableRenderedImage*, které je potomkem *RenderedImage*).

V případě požadavku na export modelu do PNG se tedy vytvoří instance třídy *BufferedImage*, která bude představovat exportovaný obrázek. Následně zavoláme metodu *paint* hlavní plochy, které předáme instanci kreslícího nástroje *Graphics*, patřící tomuto obrázku. Plocha tak vykreslí svůj obsah přímo na obrázek. Následně za pomoci třídy *ImageIO* a statické metody *write* se instance *BufferedImage* zapíše jako obrázek PNG do již předem zvoleného umístění na disku.

```
BufferedImage bi = new BufferedImage(
                panelModel.getWidth(),
                panelModel.getHeight(),
                BufferedImage.TYPE_INT_ARGB);
Graphics g = bi.createGraphics();
panelModel.paint(g);
g.dispose();
try {
    ImageIO.write(bi, "png", new File("test.png"));
} catch (IOException e) {
      JOptionPane.showMessageDialog(this, "Export error",
                              "Error", JOptionPane.ERROR_MESSAGE);
}
```

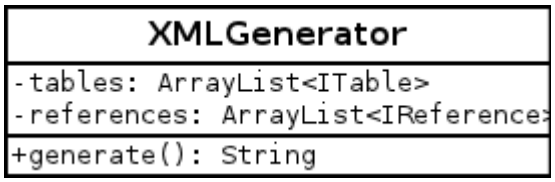
6.13 Export do XML

Formátem, který řeší různorodost výstupů jednotlivých aplikací se stal formát XML (extensible markup language). Díky tomuto formátu je možné vyměňovat data mezi různými aplikacemi, které XML podporují. Syntaxe jazyka je velmi jednoduchá. Každá entita je začata otevíracím tagem a ukončena ukončujícím tagem. Tyto entity se do sebe mohou libovolně vnořovat. Výhodou formátu XML je, že je snadno čitelný, nevýhodou jeho velikost.

Java nabízí nástroje na parsování XML, nicméně jelikož výstupem bude v podstatě textový soubor, není nutné tyto nástroje použít.

O generování XML výstupu se bude starat třída *XMLGenerator*, která jako parametry svého jediného konstruktoru přijme seznam tabulek a seznam vztahů, obojí ve struktuře *ArrayList*. Diagram třídy zobrazuje obrázek 23. Jedinou metodou této třídy bude metoda *generate*, která projde všechny tabulky a vztahy. Text, který bude uložen v XML souboru se postupně kumuluje do instance třídy *StringBuilder*, která je k podobným účelům navržena. Nakonec metoda vrátí výsledný text ve struktuře *String*.

```
public String generate() {
    String NL = System.getProperty("line.separator");
    StringBuilder s = new StringBuilder();
    s.append("<database>").append(NL);
    for (int i = 0; i < tables.size(); i++) {
        // přidání informací o tabulkách a atributech
    }
    for (int i = 0; i < references.size(); i++) {
        // přidání informací o vztazích
    }
    s.append("</database>");
    return s.toString();
}
```



Obrázek 23 - Diagram třídy XMLGenerator

Vznikne-li v hlavním programu požadavek na export do XML souboru, po zvolení cesty se vytvoří instance třídy *XMLGenerator* a následně se text vrácený metodou *generate* zapíše do souboru.

6.14 Export do DDL skriptu

Jedním ze základních kamenů aplikace, která umožňuje navrhovat databáze by měla být jednoduchá realizace navrhnuté databáze přímo v databázovém systému. To může aplikace zajistit například vytvořením sady příkazů jazyka SQL, jejichž volání v databázovém systému způsobí vygenerování odpovídajících tabulek a vztahů. Příkazy, které bude aplikace generovat budou pouze příkazy DDL, protože nic jiného než vytvářet tabulky primární a cizí klíče, sekvence, apod. nebude potřeba.

Abych umožnil pozdější rozšíření aplikace pomocí zásuvných modulů, budu předpokládat, že generovat výsledný skript bude moci jakákoliv třída, která splní určité požadavky. Tímto požadavkem bude implementace metody *generate*. Vytvořím proto rozhraní *IGenerator*, které bude tuto metodu předepisovat. Aktuálně vybraný modul bude uchováván v atributu hlavního okna, který bude datového typu *IGenerator*. V případě, že uživatel v programu požádá o vygenerování skriptu, zavolá se metoda *generate* u tohoto atributu, což zavolá příslušnou metodu.

Generování SQL kódu podle standardu SQL2 bude vestavěnou funkcí. Za tímto účelem je vytvořena třída *DefaultGenerator*, která implementuje rozhraní *IGenerator*. Třída bude v konstruktoru přebírat seznam tabulek a seznam vztahů. Metoda generate bude

využívat, podobně jako při generování XML, instance třídy *StringBuilder*. Procházením jednotlivých tabulek se vytvářejí konstrukce *CREATE TABLE* s konkrétními parametry. Dalším krokem je tvorba primárních klíčů. Ten lze u každé tabulky vytvořit (pokud existuje) v dané iteraci po vytvoření konstrukce vytvářející tabulku konstrukcí *ALTER TABLE*, opět s požadovanými parametry. Druhý cyklus prochází jednotlivé vztahy a generuje konstrukce *ALTER TABLE*, které u daných tabulek vytvářejí cizí klíče z požadovaných atributů. Metoda vrací výsledek ve struktuře *String*.

```
public String generate() {
      StringBuilder s = new StringBuilder();
      for (int i = 0; i < tables.size(); i++) {
            s.append("CREATE TABLE …");
            s.append("ALTER TABLE… PRIMARY KEY…");
      }
      for (int i = 0; i < references.size(); i++) {
            s.append("ALTER TABLE… FOREIGN KEY…");
      }
      return s.toString();
}
```

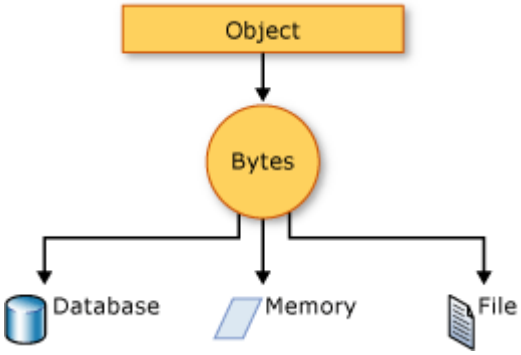
6.15 Ukládání a načítání projektu

Aby bylo možné rozpracovanou práci uložit a později se k ní vrátit, je třeba zajistit ukládání aktuálního projektu na pevný disk, případně jiné nosné médium. Struktura souboru může být různá. Mnou zvolený způsob bude využívat možnosti jazyka Java a instance, které jsou z pohledu stavu modelu důležité, bude ukládat pomocí serializace objektů.

Serializace je metoda, která objekty převádí na sekvenci bajtů, kterou lze snadno uložit do souboru. Tento soubor je potom nečitelný, na rozdíl od textového. Názorně je proces serializace zobrazen na obrázku 24.

Objekty, které budou do souboru projektu serializovány, budou instance obalových tříd pro tabulky a reference. Tyto obalové třídy zajišťují uchovávání dalších atributů, které bude při zpětném načtení potřeba zjistit. U tabulek jsou to například pozice na ploše nebo velikost. U vztahů je to seznam manipulačních bodů, kterými je čára symbolizující vztah vedena.

Obalové třídy, které budou v programu využity se jmenují *SaveTable* a *SaveRef*, třída, která bude obalovat seznamy objektů těchto dvou (bude mít dva atributy typu *ArrayList*) se bude nazývat *SavingWrapper*. Instance třídy *SavingWrapper* bude serializována do souboru na disku. Tento soubor bude představovat rozpracovaný projekt, který bude možno kdykoliv opět načíst.



Obrázek 24 - Serializace objektů [4]

Aby třída byla v Javě „serializovatelná“ (tzn. aby její instance bylo možno serializovat), musí implementovat rozhraní *Serializable*. Je tedy nutné ke všem třídám, jejichž objekty budou nějakým způsobem ukládány, připsat, že implementují toto rozhraní. Rozhraní nepředepisuje žádnou metodu.

O serializaci a zápis do souboru se stará třída *ObjectOutputStream* a její metoda *writeObject*, která v parametru vezme instanci, která bude serializována a zapsána do souboru. Ještě před tím je třeba vytvořit instanci třídy *FileOutputStream*, která bude představovat proud dat směřujících do požadovaného souboru. Tuto instanci využijeme v konstruktoru třídy *ObjectOutputStream.*

```
// s = konstruktor třídy SavingWrapper
// f = výstupní soubor
FileOutputStream fileOut = new FileOutputStream(f);
ObjectOutputStream out = new ObjectOutputStream(fileOut);
out.writeObject(s);
out.close();
fileOut.close();
```

Opětovné načtení takto uloženého projektu je realizováno opačným principem. Slouží k tomu třídy *FileInputStream* a *ObjectOutputStream*. Instance třídy *FileInputStream* vytvoří proud dat ze souboru a instance třídy *ObjectOutputStream* a její metoda *readObject* přečte zasaný objekt.

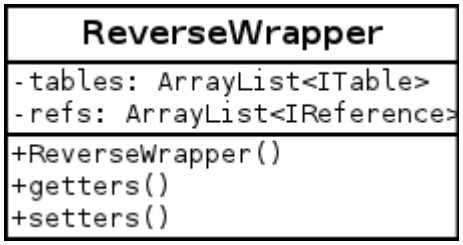
```
FileInputStream fileIn = new FileInputStream(f);
ObjectInputStream in = new ObjectInputStream(fileIn);
SavingWrapper s = (SavingWrapper) in.readObject();
in.close();
fileIn.close();
```

Po načtení objektu třídy *SavingWrapper* stačí již seznamy tabulek a vztahů předat pracovní ploše (instanci třídy *PanelModel*), která je pomocí svých metod správně zařadí a vykreslí.

6.16 Reverse Engineering

Jako reverse engineering se souhrně označují techniky rekonstrukce nějakého problému například podle existujících výstupů. V navrhovaném programu se bude jednat o transformaci, kde vstupem bude skript s DDL příkazy a výstupem bude model na pracovní ploše programu. To může být užitečné pro prezentaci již existující databáze dalším lidem, či z důvodu přehlednosti apod.

Jelikož program bude moci podporovat více různých databázových systémů díky zásuvným modulům, lze využít již navrženého rozhraní *IGenerator*. Rozšířím ho tedy o metodu s názvem *reverseEngineer*, jejímž výstupem bude seznam tabulek a seznam vztahů v nějaké předem definované struktuře. Tato obalující struktura ponese název *ReverseWrapper*. Její diagram je zobrazen na obrázku 25. Vstupem metody *reverseEnginner* bude instance třídy *Scanner*, která představuje jednoduchý parser textových řetězců. V tomto případě je předpokládáno, že ona instance bude nejčastěji pracovat s textovým souborem.



Obrázek 25 - Diagram třídy ReverseWrapper

Metodu *reverseEngineer* tedy implementujeme ve třídě *DefaultGenerator*. Začátkem zpětného inženýrství (jak se někdy reverse engineering v češtině označuje) bude tedy vytvoření instance třídy *Scanner* pro čtení ze souboru se skripty. Tuto instanci předáme jako vstup metodě *reverseEngineer* a na výstupu budeme očekávat tabulky a vztahy.

Vestavěný reverse engineering bude parsovat SQL skripty ve standardu SQL2. Samotná metoda *reverseEngineer* začne odstraňováním komentářů. Na každém řádku hledá dvojici znaků --, která je začátkem komentáře v jazyce SQL. V případě že tuto dvojici nalezne, odstraní jí a veškerý text na řádku, který se nachází za ní. Dalším krokem je rozdělení textu do jednotlivých částí na základě výskytu znaku „;“, který označuje konec příkazu. Následně se rozpozná DDL konstrukce.

V případě rozpoznání konstrukce pro vytvoření tabulky se zjistí její název a potom jednotlivé atributy oddělené čárkami. U každého atributu se pak rozpozná název, datový typ a nakonec jeho vlastnosti.

Pokud metoda narazí na konstrukci pro změnu tabulky, zjistí zda se jedná o vytvoření primárního klíče nebo o vytvoření cizího klíče. Tak jako tak vyhledá tabulku podle názvu a vytvoří na daných atributech primární nebo cizí klíč.

Reference na instanci třídy *ReverseWrapper*, která uchovává seznamy všech tabulek a vztahů, je vrácena návratovou hodnotou metody. Hlavní program jej potom může předat pracovní ploše, která tabulky a vztahy uloží a vykreslí.

6.17 Tvorba zásuvných modulů

Aby byl program univerzální, je potřeba aby nebyl závislý jen na jednom databázovém systému. Vestavěný generátor, seznam datových typů a další věci vztahující se ke kódu databázového jazyka SQL jsou navrženy tak, aby podporovaly standard SQL2. Nicméně mnoho dialektů jazyka SQL má svá specifika a liší se v těch či oněch rysech.

Jedním z možných řešení je použití zásuvných modulů, takzvaných pluginů. Plugin je obvykle binární soubor (v zásadě může být jakýkoliv) jehož přidáním do adresáře aplikace se tato rozšíří o další funkce. V tomto případě se bude jednat o podporu dalších databázových systémů.

Plugin pro vytvářenou aplikaci se bude skládat ze dvou hlavních instancí. Jednou z nich bude instance třídy implementující rozhraní *IGenerator*. Tato třída tedy bude mít implementované metody *generate*, která generuje DDL skript, a *reverseEngineer*, která je schopná naopak z DDL skriptu sestavit grafický model. Druhým objektem, který je potřeba v pluginu uchovat je seznam datových typů, specifických pro tento databázový systém. Za účelem sjednocení struktury tříd sloužících k uchování datových typů je vytvořeno rozhraní *IDataTypes*, které předepisuje jedinou metodu – *getTypeList*. Metoda vrací seznam datových typů ve struktuře *ArrayList*.

Nyní tedy postačí vytvořit potřebné třídy, naimplementovat jejich metody podle požadavků databázového systému a serializovat je do souboru. Například plugin pro databázi MySQL se bude sestávat z objektů tříd pojmenovaných *MySqlDataTypes* a *MySqGenerator*. Obě třídy implementují příslušná rozhraní a jejich metody jsou implementovány tak, aby vyhovovaly databázovému systému MySQL. Finálním krokem bude vytvoření třídy *MySqlCreate* obsahující statickou metodu main. V metodě *main* vytvoříme obě instance, které zabalíme do jedné instance nově vytvořené třídy *PluginWrapper*. Třída *PluginWrapper* bude mít tři atributy – *dataTypeList* datového typu *ISqlDataTypes*, *generator* datového typu *IGenerator* a *name* datového typu *String*. Instance této třídy již bude obsahovat všechno, co je pro plugin potřeba, včetně názvu pluginu. Stačí jej tedy serializovat a uložit do souboru.

Vzniklý soubor se stává pluginem až po té, co ho aplikace dokáže načíst. To udělá již popsaným způsobem načítání serializovaných objektů. Po načtení získá instanci třídy *PluginWrapper* obsahující specifický generátor včetně specifických datových typů a může je začít využívat.

Aby měl uživatel možnost zvolit si ze všech zásuvných modulů nacházejících se v pracovním adresáři aplikace, program prochází všechny soubory v tomto adresáři a porovnává jejich příponu s řetězcem *.dpl*. Tuto koncovku jsem zvolil tak, aby alespoň přibližně připomínala zkratku anglických slov „data plugin“. V případě shody se program pokusí z tohoto souboru načíst serializovaný objekt třídy *PluginWrapper* a zobrazit název zásuvného modulu uložený v této třídě.

Načtený generátor se ukládá do atributu *generator* hlavního okna. Při požadavku na vygenerování DDL skriptu se tedy zavolá generátor modulu, který je právě načten. V případě nezvolení žádného modulu (ponechání vestavěného generátoru pro SQL2) se použije vestavěný generátor.

Seznam datových typů standardu SQL2, se po načtení modulu vždy zachovává. Nový seznam datových typů je načten do druhého seznamu (v levé nástrojové liště dole), který je ke zobrazení datových typů, specifických pro konkrétní databázový systém, určen.

Načtení do seznamu je provedeno pomocí instance třídy *DefaultListModel*, které se postupně předají instance všech načtených datových typů. Nakonec je seznamu nastaví tento model pomocí metody *setModel*.

```
DefaultListModel lm = new DefaultListModel();
for (ISqlDataType type : dbSpecificTypes.get(0).getTypeList()) {
      lm.addElement(type);
}
listSpecificDataTypes.setModel(lm);
```

Grafický seznam vypisuje objekty, které má ve svém modelu pomocí metody *toString* konkrétního objektu. Na vrcholu hierarchie dědičnosti v Javě je třída *Object*. Znamená to tedy, že každý objekt dědí všechny metody této třídy. Třída *Object* implementuje metodu *toString*. Pokud je třeba objekt vypsat, jako v tomto případě v seznamu, nějakým jiným způsobem, než předepisuje třída *Object*, lze zděděnou metodu *toString* přepsat do podoby, která bude vyhovovat našim požadavkům.

Přepsáním metody *toString* u třídy *SqlDataType* tedy docílíme požadovaného zobrazení datových typů v seznamu.

```
@Override
public String toString() {
    return shortCut;
}
```

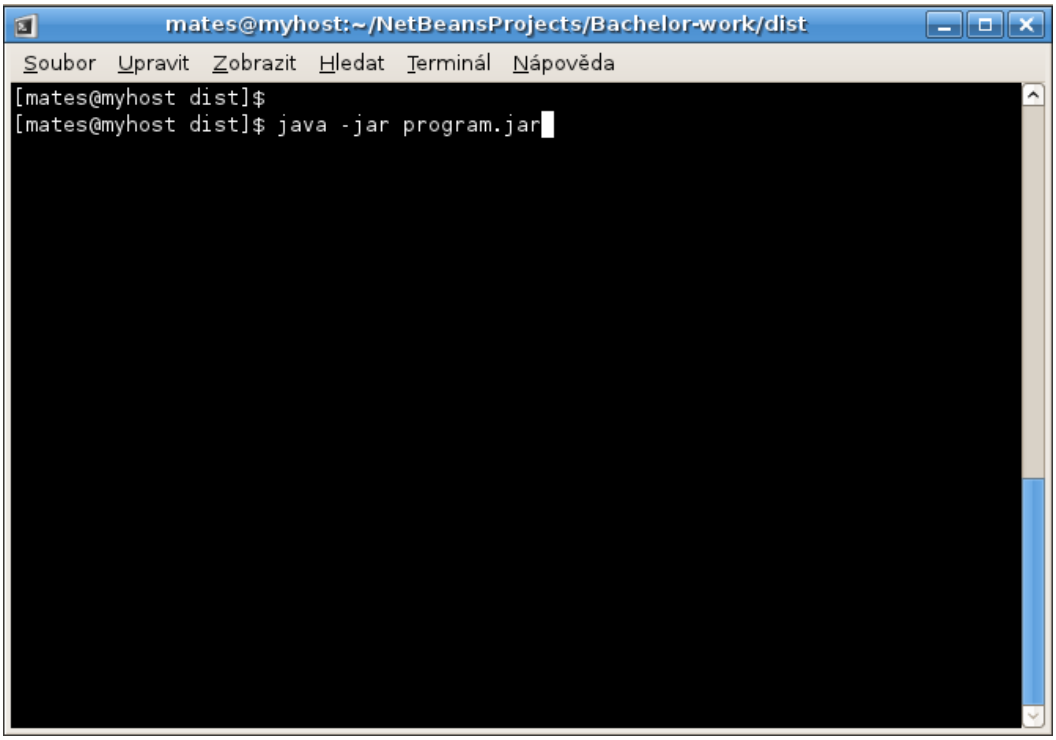
7 Uživatelská příručka programu

7.1 Instalace a spuštění

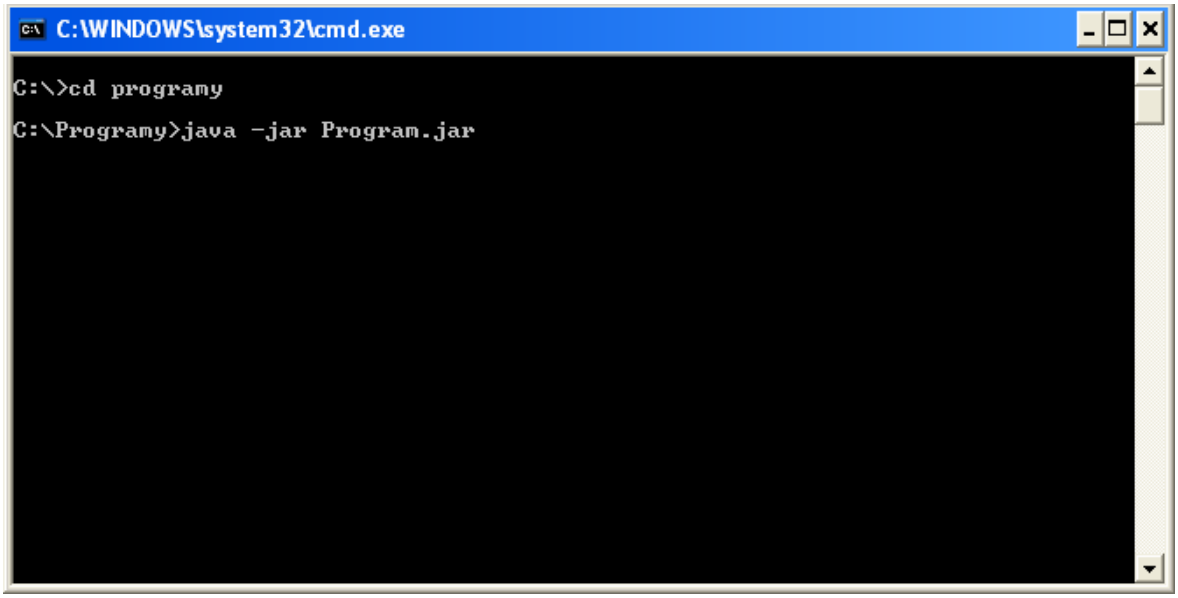
Celý program se nachází na přiloženém kompaktním disku a je zabalený v archivu JAR. Program se před spuštěním nemusí nijak instalovat, soubor stačí zkopírovat kamkoliv na pevný disk počítače. Archiv je spustitelný pomocí programu java, který je součástí virtuálního stroje Javy (Java Virtual Machine). Pro spuštění programu na daném operačním systému je tedy nutností mít tento virtuální stroj nainstalován.

Ve většině běžných operačních systémů je po instalaci virtuálnío stroje asociována přípona souboru jar s programem java. Stačí tedy na soubor programu otevřít v grafickém uživatelském rozhraní. V případě, že se tak nestane a systém aplikaci nedokáže spustit přímo, je třeba program spustit z příkazové řádky pomocí programu java. Ke spuštění archivů JAR slouží parametr –jar. Dalším parametrem je již název samotného archivu.

Nejprve je potřeba dostat se do adresáře, kam je program nakopírován. Potom se v příkazové řádce do tohoto adresáře dostaneme pomocí příkazů cd (change directory). Nakonec program spustíme pomocí programu java s přepínačem –jar. Spuštění programu z příkazového řádku názorně ukazují obrázky 26 a 27.



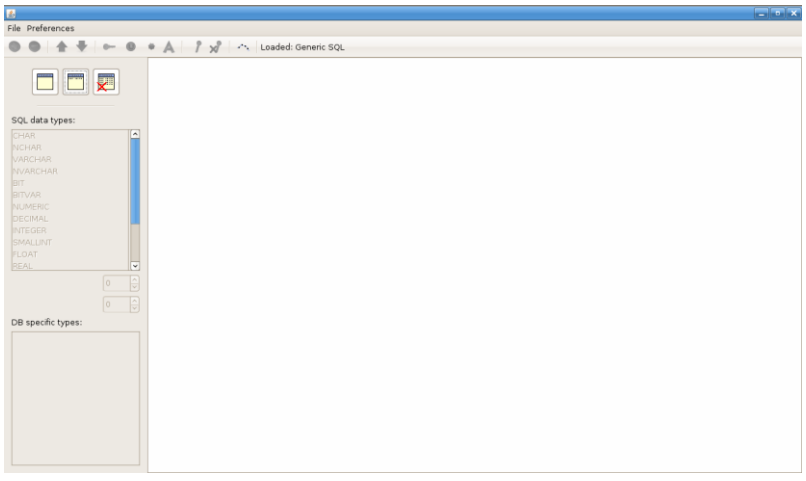
Obrázek 26 - Spuštění programu z příkazové řádky systémů GNU/Linux



Obrázek 27 - Spuštění programu z příkazové řádky systémů Windows

7.2 Popis programu

Hlavní okno programu je rozděleno do tří hlavní částí. Hned pod hlavním menu se nachází nástrojová lišta pro práci s atributy. Zde jsou tlačítka pro přidávání nebo odebírání atributů, pro změnu jejich pozice, změnu jejich vlastností a tlačítka pro správu cizích klíčů. Dále se zde nachází tlačítko pro zobrazení nebo skrytí manipulačních bodů. Napravo od něho je zobrazen název modulu, který je momentálně načten. Druhou částí hlavního okna je nástrojová lišta na levém okraji. Tato nabízí tlačítka pro tvorbu tabulek, seznamy a další prvky pro správu datových typů atributů. Třetí a největší částí hlavního okna je pracovní plocha, na kterou se umisťují tabulky. Obrázek 28 zobrazuje celé hlavní okno programu.



Obrázek 28 - Okno hlavního programu

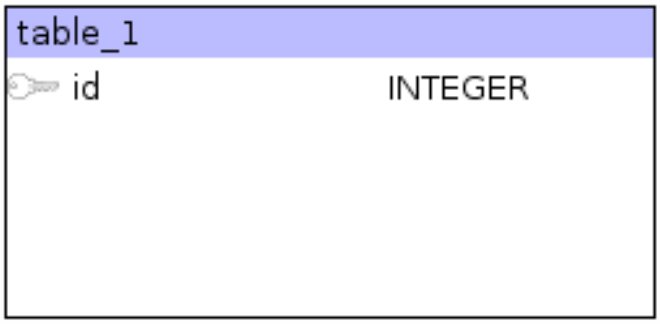
V hlavním menu nalezneme položky FILE a PREFERENCES. Pod položkou FILE nalezneme akce pro uložení a načtení projektu, pro export do různých formátů zahrnujících PNG, XML nebo DDL skript. Dále zde nalezneme položku pro import DDL skriptu pro automatické vytvoření modelu databáze. V položce PREFERENCES nalezneme pouze možnost, která zobrazí dialogové okno pro výběr pluginů.

7.3 Vytváření tabulek

Je-li program spuštěn, lze začít vytvářet návrh databáze. To nelze učinit bez vytvoření první tabulky. Tlačítka pro tvorbu tabulek jsou dvě a jsou umístěna na levé nástrojové liště, v její horní části. Tato tlačítka jsou zobrazena na obrázku 29. Na tlačítkách jsou ikony představující tabulky. Levé z těchto tlačítek slouží k vytvoření jednoduché tabulky bez atributů. Pravé vytváří tabulku s již připraveným atributem s názvem id, který je datového typu INTEGER a je zároveň primárním klíčem. Trojici doplňuje tlačítko sloužící ke smazání tabulky. To je úplně vlevo a je označeno ikonou, na které je zobrazena tabulka s červeným křížkem.

Obrázek 29 - Ikony pro vytváření a mazání tabulek

Kliknutím na jedno ze dvou tlačítek vytvářejících tabulku program očekává následný klik na pracovní plochu. Po kliknutí na plochu se tlačítko vrátí do původního neoznačeného stavu a na ploše se vytvoří požadovaná tabulka. Na obrázku 30 je zobrazena nově vytvořená tabulka obsahující předpřipravený primární klíč.

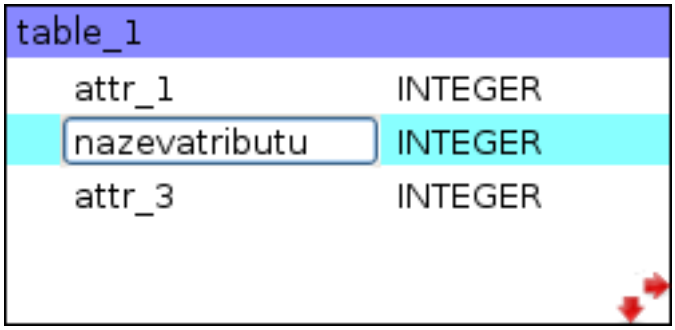


Obrázek 30 - Tabulka s předpřipraveným primárním klíčem

Další atributy lze do tabulky přidávat pomocí krajního levého tlačítka v horní nástrojové liště. Aby se nový atribut přidal na konec tabulky, po které je to požadováno, musí být tato tabulka aktivní. To se provede kliknutím na plochu tabulka. Tabulka následně změní barvu svého záhlaví na tmavě modrou. Do takto označené tabulky se nyní budou klikem na ono tlačítko přidávat atributy typu *integer* s názvem attr_číslo, kde u každého přidaného atributu bude číslo o jedničku stoupat. Podobně je tomu s názvy tabulek.

Odstranění atributu se provádí tlačítkem, které je druhé zleva na horní nástrojové liště. Aby byl odstraněn správný atribut, je třeba tento atribut označit. To se provede klikem myší do prostoru atributu. Atribut získá tyrkysovou barvu, která signalizuje, že je označen. Klikem na tlačítko pro odstranění atributu se atribut odstraní.

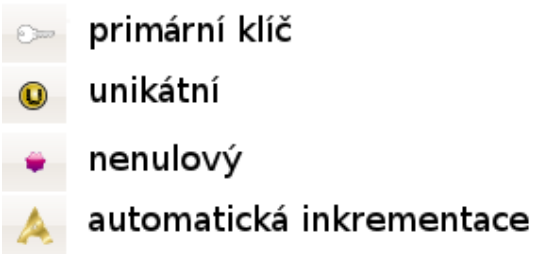
Změna názvu atributu a změna názvu tabulky se provádí klikem pravého tlačítka myši do prostoru názvu. Po kliku se zobrazí editační okno, ve kterém lze název upravit. Po úpravě názvu stačí stisknout Enter a změna názvu atributu nebo tabulky je dokončena. Moment editace názvu atributu zachycuje obrázek 31.



Obrázek 31 - Editace návrhu atributu

Pořadí atributů v tabulce lze měnit šipkami, nacházejícími se taktéž na horní nástrojové liště. Pro změnu pořadí atributů je třeba jeden z atributů označit a šipkami nahoru a dolů s ním lze potom posouvat.

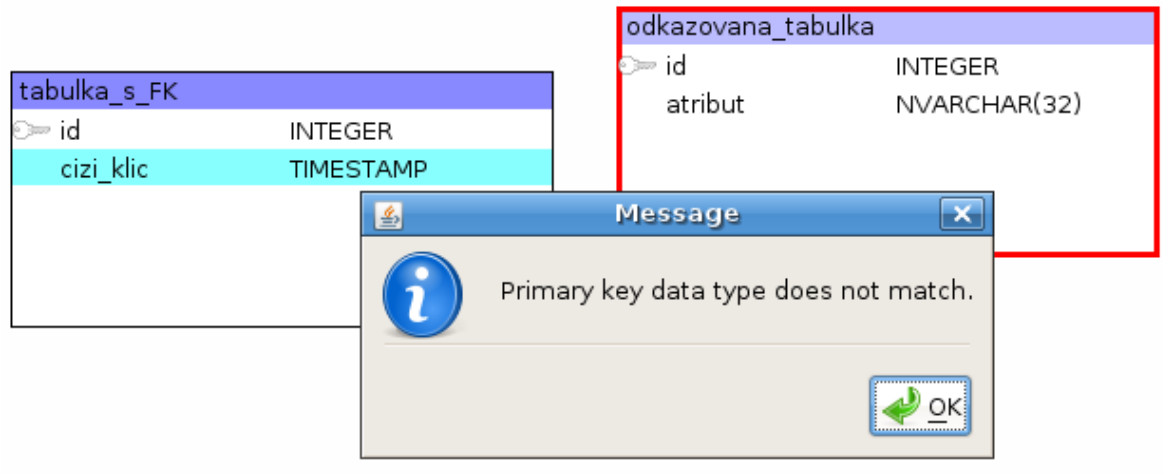
Vlastnosti atributů lze měnit po označení konkrétního atributu dalšími tlačítky horní nástrojové lišty. Tato tlačítka po řadě znamenají: primární klíč, unikátní, nenulový, automatická inkrementace. Na obrázku 32 je přehledně zobrazen význam jednotlivých obrázků na tlačítkách.



Obrázek 32 - Význam tlačítek pracujících s atributem

7.4 Vytváření cizích klíčů

Další dvojice tlačítek na horní nástrojové liště má za úkol pracovat s cizím klíčem. V případě označení konkrétního atributu se levé z těchto tlačítek zpřístupní. Po kliknutí na toto tlačítko program očekává výběr tabulky, na kterou bude cizí klíč (vybraný atribut) odkazovat. Klikem levým tlačítkem myši na tabulku, na kterou má cizí klíč odkazovat, je tato tabulka vybrána jako odkazovaná. Než se ovšem cizí klíč vytvoří, zkontroluje se datový typ primárního klíče tabulky s datovým typem cizího klíče tabulky, ze které odkazujeme. V případě neshody program zobrazí informaci o neodpovídajících datových typech a relace se nevytvoří. Dialogové okno zobrazující tuto informaci je zachyceno na obrázku 33.



Obrázek 33 - Zpráva o neshodě datových typů

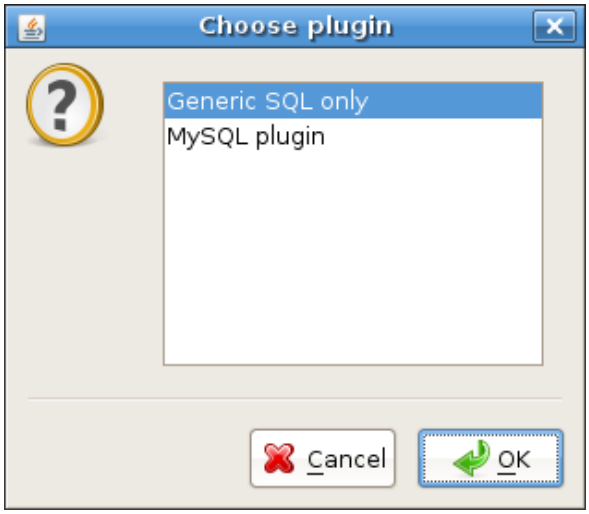
V případě shody datových typů se vztah vytvoří a vykreslí. Pokud uživatel bude chtít vytvořit cizí klíč skládající se z více atributů, označí tyto pomocí podržení klávesy CTRL. Atributy je nutno označit v pořadí, ve kterém mají odkazovat na pořadí primárního klíče.

Odebrání vztahu se provádí označením libovolného atributu cizího klíče a kliknutím na ikonu zrušení vztahu. V případě, že atribut je členem pouze jednoho cizího klíče, vztah se odstraní. V případě že je členem více vztahů, zobrazí se uživateli dialogové okno, ve kterém si vybere vztah, který chce odebrat. Okno je zobrazeno na obrázku 34.

Obrázek 34 - Dialogové okno pro výběr vztahu

7.5 Výběr zásuvného modulu

Zásuvný modul lze vybrat v hlavním menu pod položkou PREFERENCES volbou CHOOSE DB SYSTEM. Zobrazí se dialogové okno, ve kterém se zobrazí všechny dostupné zásuvné moduly. Po výběru modulu stačí kliknout na tlačítko OK a modul se načte. Dialogové okno s dostupnými moduly je zobrazeno na obrázku 35.



Obrázek 35 - Dialogové okno pro výběr modulu

7.6 Export a import databáze

Ukládání modelu zajišťuje volba SAVE v hlavním menu FILE. Po výběru umístění a názvu souboru je model uložen a připraven pro pozdější načtení. Načtení uloženého souboru se provádí volbou OPEN opět v hlavním menu pod položkou FILE. Program otevře dialog pro výběr souboru, který chceme načíst.

Export se nachází opět pod volbou FILE. Pod položkou EXPORT se ukrývají tři další volby, které udávají, v jakém formátu chceme model exportovat. Možnosti jsou PNG, XML a DDL script. V případě exportu do DDL se vytvoří skript v SQL dialektu modulu, který je právě načten.

Volba IMPORT zobrazí dialog pro výběr souboru s koncovkou SQL. Po výběru souboru se aktuálně načtený modul pokusí soubor parsovat a převést do modelu.

8 Závěr

Primárním cílem práce bylo implementovat aplikaci umožňující jednoduché a interaktivní modelování diagramů databází. S možností využití grafického návrhu pro implementaci konkrétní databáze.

Cíl se podařilo splnit dle zadání, se všemi požadavky. Navržená aplikace umožňuje snadný návrh tabulek, atributů s jejich datovými typy a databázových vztahů. Dále umožňuje export modelu do grafického formátu PNG, do formátu XML a do skriptu v jazyce SQL obsahujícího příkazy pro definici datových struktur. Zvládá také reverse engineering na úrovni standardu SQL2.

Program je rozšířitelný pomocí zásuvných modulů, které rozšiřují využití programu na další dialekty jazyka SQL.

Jedním z nejsložitějších úkolů byl návrh základních tříd tak, aby vyhovovaly pozdějším požadavkům aplikace. Dalším problémem se stal systém komunikace mezi komponentami, kterou vyřešilo použití návrhového vzoru *Observer*.

Aplikace by se dala do budoucna rozšířit, aby podporovala více nastavení tabulek a atributů. Mohly by například přibýt možnosti jako defaultní hodnota, možnost sestavovat komplexní omezení, vytváření triggerů apod. Dále by pro aplikaci mohlo být napsáno více zásuvných modulů, aby podporovala co nejvíce různých databází.

Literatura

[1] **Data & Object Factory, LLC. 2001 - 2011.** Observer Design Pattern in C# and VB.NET. *Dofactory.* [Online] Data & Object Factory, LLC, 2001 - 2011. [Citace: 11. Květen 2011.] http://www.dofactory.com/Patterns/PatternObserver.aspx.

[2] **GROFF, James R., WEINBERG, Paul N. 2005.** *SQL: Kompletní průvodce.* Brno : Computer Press, 2005. ISBN 80-251-0369-2.

[3] **HEROUT, Pavel. 2007.** *Java - grafické uživatelské prostředí a čeština.* České Budějovice : Kopp, 2007. 80-7232-328-9.

[4] **Microsoft. 2011.** Serializace (C# a Visual Základní). *MSDN.* [Online] Microsoft, 2011. [Citace: 11. Květen 2011.] http://msdn.microsoft.com/cs-cz/library/ms233843.aspx.

[5] **MySQL Inc. 2007 - 2009.** About. *MySQL Workbench 5.2.33.* [Online] MySQL Inc., 2007 - 2009. [Citace: 11. Květen 2011.] http://wb.mysql.com/?page_id=6.

[6] **Oracle. 2011.** SQL Developer Data Modeler. *Oracle | Hardware and Software, Engineered to Work Together.* [Online] Oracle, 2011. [Citace: 11. Květen 2011.] http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/index.html.

[7] **RIORDAN, Rebecca M. 2000.** *Vytváříme relační databázové aplikace.* Praha : Computer Press, 2000. ISBN 80-7226-360-9.

Příloha A – Kompaktní disk se zdrojovými kódy

Přiložený kompaktní disk obsahuje zdrojové kódy vyvinuté aplikace, zkompilovanou aplikaci v archivu JAR a text práce ve formátu PDF.