Univerzita Pardubice

Fakulta elektrotechniky a informatiky

SQL Manažer

Tomáš Váňa

Bakalářská práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Tomáš VÁŇA**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **SQL Manažer**

Zásady pro vypracování:

Cílem práce bude napsat Java GUI aplikaci s názvem SQL Manažer, který bude realizovat inteligentní rozhraní mezi klientem a vybranou databází. Teoretická část: V teoretické části bakalářské práce budou představeny a zhodnoceny komerčně používané programy podobného typu. Budou popsány hlavní funkce SQL manažeru. Dále zde bude charakteristika jazyka SQL, Javy a používaných databází.
Implementační část: Vytvořit GUI aplikaci pomocí technologie J2SE, která bude obsahovat základní funkce klientského rozhraní (manažeru) pro komunikaci s databázovým serverem. Je to např. možnost připojení k vybrané databázi, změna hesla, zadávání SQL příkazů a zobrazení výsledku, tvorba základních databázových objektů atd.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

*** B. Eckel: Myslíme v jazyku Java. Knihovna zkušeného programátora, Grada Publishing, 2001. * Brett Spell: Java Programujeme profesionálně, Computer Press, 2002. * David C.Kreines: Oracle DBA, kapesní průvodce, nakladatelství OReilly, 2006. * L'uboslav Lacko: Oracle Správa, programování a použití databázového systému, Computer Press, 2007.**

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

Univerzita Pardubice
Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlášení

Prohlašuji:

Tuto práci jsem 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 15. 5. 2009 …….

Tomáš Váňa

Poděkování

Tímto bych rád poděkoval panu Ing. Zdeňku Šilarovi za to, že mi umožnil pracovat na mnou zvoleném tématu a za vedení a rady při tvorbě této práce.

Dále bych chtěl také poděkovat své rodině za podporu při studiu a tvorbě této práce.

Anotace

Práce se zabývá programovacím jazykem Java a jeho možnostmi využívat databáze. Jsou v ní všeobecně popsány dnes využívané databáze, jazyk SQL, programovací jazyk Java a rozhraní JDBC. Dále se věnuje představení aplikací pro správu databází. Práce se kromě propojení s databází také zabývá základy tvorby GUI aplikací v Javě.

Úkolem této práce bylo vytvořit aplikaci pro jednoduchou správu databáze. Detailnímu návrhu této aplikace je zde věnována celá jedna kapitola.

Klíčová slova: JDBC, Java, JVM, SQL, databáze, Oracle, MySQL, nástroje pro správu databáze, relační databáze, Java GUI aplikace, Swing

Annotation

This work is about programming language Java and about its opportunities for using databases. Today's generally used databases, SQL language, programming language Java and interface JDBC are described here. Next it attends to present applications for database management. Except connection with database it also involves the basic of creating Java GUI applications.

The load of this work was to create application for simple database administration and one whole section of this work is about this application.

Keywords: JDBC, Java, JVM, SQL, database, Oracle, MySQL, database management tools, relational database, Java GUI applications, Swing

Obsah

Seznam obrázků

Použité zkratky

AWT	Abstract Window Toolkit. Rozhraní pro tvorbu GUI aplikací.
GPL	General Public License. Licence pro svobodný software.
GUI	Graphical User Interface. Grafické uživatelské rozhraní.
IDE	Integrated Development Environment. Integrované vývojové prostředí. Sada vývojových nástrojů pro rychlou a pohodlnou tvorbu aplikací.
J2EE	Java Platform, Enterprise Edition. Součást Javy určená pro tvorbu podnikových aplikací a informačních systémů.
J2ME	Java Platform, Micro Edition. Platforma Javy určená pro tvorbu aplikací pro kapesní počítače a mobilní telefony.
J2SE	Java Platform, Standard Edition. Základní verze Javy vhodná pro tvorbu konzolových aplikací a aplikací s grafickým rozhraním.
JDBC	Java Database Connectivity. Rozhraní pro jednotný přístup k databázím
JFC	Java Foundation Classes. Třídy pro tvorbu grafického uživatelského rozhraní. Novější než AWT. Známé též jako Swing.
JNI	Java Native Interface. Rozhraní pro volání nativních metod z Javy.
JVM	Java Virtual Machine. Virtuální stroj jazyka Java vykonávající tzv. bytecode do něhož jsou Java aplikace překládány.
JWS	Java Web Start. Technologie distribuce Java aplikací pomocí internetu.
ODBC	Open Database Connectivity. Rozhraní společnosti Microsoft pro jednotný přístup k datům.
SQL	Structured Query Language. Strukturovaný dotazovací jazyk používaný pro práci s databázemi.

1 Úvod

Databáze, především relační, dnes představují jednu z nejdůležitějších součástí dnešního průmyslového světa. Databáze vděčí za svůj obrovský rozmach především jazyku SQL. V současnosti je jazyk SQL standardem. SQL dnes můžeme najít v obrovské spoustě databázových produktů vyskytujících se od velkých střediskových počítačů až po počítače osobní anebo dokonce počítače kapesní. Síla jazyka SQL je vyjádřena i tím, že se jej za základ svých databází zvolily i největší hráči v oboru (Microsoft, Oracle, IBM,…).

Za druhou stranu mince můžeme označit aplikace využívající databáze. A to ať už se jedná o běžné aplikace, webové aplikace či o rozsáhlé podnikové aplikace. V případě aplikací hraje velmi důležitou roli vybraný programovací jazyk. V této práci se budu výhradně věnovat programovacímu jazyku Java.

Cílem této práce je poukázat na možnosti propojení Java aplikací s databázemi. Pro konkrétní demonstraci jsem vytvořil aplikaci s názvem SQL Manažer, jejímž úkolem je zprostředkovat rozhraní mezi uživatelem a konkrétní vybranou databází. Program je napsán pro práci s databázemi Oracle a MySQL.

V práci budou nastíněny základní rysy jazyka Java a především se budu snažit popsat rozhraní JDBC. Kapitolu také věnuji SQL a porovnaní různých aplikací pro správu databázových systémů.

2 Databáze a jazyk SQL

2.1 Relační databáze a jejich historie

S původní koncepcí relační databáze [1] přišel v roce 1970 Dr. Codd. Jeho motivací bylo zjednodušit původní koncept explicitní struktury rodič-potomek. Koncepce rodič-potomek byla založena na hierarchickém modelu ukládání dat. Codd tuto hierarchii eliminoval tím, že reprezentoval veškerá data v databázi jako jednoduché tabulky datových hodnot sestávajících se z řádků a sloupců.

Bohužel obecná definice pojmu relační databáze byla mnohem méně přesná než její matematické vyjádření. To vedlo k tomu, že mnoho pokusů vytvořit pravé relační databáze selhalo, a mnoho databází označující se jako pravé relační databáze, ve skutečnosti relačními databázemi nebylo. To vedlo v roce 1985 Dr. Codda k vydání článku, který definoval 12 pravidel, jimiž by se měli řídit všechny databáze, které by se chtěly označovat jako „pravé relační databáze“.

Relační databáze je tedy taková databáze, kde veškerá uživatelsky viditelná data jsou uspořádána do tabulek datových hodnot a kde se veškeré databázové operace odehrávají právě nad těmito tabulkami. Účelem takové definice je explicitně vyloučit takové struktury, jako jsou ukazatele z hierarchických či síťových databází. I tak mohou relační databáze reprezentovat vztah rodič-potomek, ale ze předpokladu, že jsou reprezentovány pouze na základě hodnot dat obsažených v tabulkách. Nyní uvedu základní prvky relační databáze:

- **Tabulka.** Je to základní organizační prvek relační databáze. Každá tabulka je identifikována jednoznačným jménem. Sloupce v tabulkách nazýváme atributy, řádky pak nazýváme záznamy. Sloupce si oproti řádkům zachovávají své specifické pořadí.
- **Primární klíč.** Je jednoznačným identifikátorem záznamu (řádku v tabulce). Ve sloupci, který je označen jako primární klíč, se tudíž nesmí vyskytovat některá hodnota vícekrát a nesmí ani obsahovat prázdnou hodnotu NULL.
- **Relace.** Přestože jsou v relačních databázích zakázané explicitní ukazatele, přesto podobné relace můžeme najít i u relačních databází. Rozdílná je

ovšem jejich realizace. V relačních databázích se pro vyjádření relace využívá společných datových hodnot uložených v různých tabulkách (i v jedné, v případě spojení tabulky sama se sebou).

- **Cizí klíč.** Je to hodnota ve sloupci, jež odpovídá hodnotě primárního klíče v jiné tabulce. Právě kombinace primární-cizí klíč se velmi podobá relaci rodič-potomek z hierarchického modelu databáze.

2.2 Databázový systém Oracle

Společnost **Oracle** [2] patří k těm největším a nejsilnějším společnostem na trhu, jenž nabízí databázové produkty. Kromě databází se Oracle zaměřuje na tvorbu nástrojů pro vývoj a správu databází (Oracle SQL Developer), tvorbu **JavaEE** aplikací (JDeveloper) nebo na tvorbu **CRM (Customer Relationship Management)** systémů. Firmě Oracle patří historický primát v tom, že se jedná o vůbec prvního komerčního dodavatele SQL databáze. Dokonce se mu podařilo předběhnout i takovou společnost, jakou je IBM.

Původní platformou pro provoz databází Oracle byli minipočítače Digital. Jednou z hlavních výhod databází od Oraclu je jejich velmi snadná přenositelnost, databázové systémy mohou běžet na desítkách různých operačních systémů, od Windows až po sálové počítače od IBM. Díky síťové vrstvě Oracle mohou všechny tyto databáze spolupracovat v distribuované databázové síti. Díky této vlastnosti oslovil Oracle velké společnosti, které na produktech Oracle vybudovali celou svou datovou infrastrukturu. Tím se Oracle de facto stal nepsaným standardem v oblasti správy dat podnikových informačních systémů.

Původní databázový systém Oracle byl založen na **Systém/R** od IBM a v hlavních rysech zůstával kompatibilní s jinými SQL produkty od IBM. Nejnovější verzí databáze od Oraclu je **Oracle Databese 11g**. Jedná se o komerční databázi, tudíž je její užívání zpoplatněno. Nová verze těží z výhod **gird computeringu** (spojení více samostatných počítačů do jednoho celku), navíc je kladen důraz na zjednodušení automatické správy. Databáze Oracle jsou vždy vydávány v několika edicích:

- **Express Edition.** Jedná se o základní verzi databáze. Databáze funguje pouze na 32 bitových systémech, kde využívá maximálně jeden procesor a pouze jen 1 GB paměti. Pro uživatelská data jsou vyhrazeny čtyři GB dat. Na serveru smí běžet jen jedna instance Oracle Database XE.
- **Standard Edition One.** Podporuje maximálně dva procesory. Umožňuje provoz nejen klasických transakčních aplikací, ale i aplikací pracující s daty ve formátu XML.
- **Standard Edition.** Rozšiřuje Standard Edition One o možnost využít vyššího výkonu díky využití serveru s až čtyřmi procesory. Ve verzi 11g je díky technologii Real Application Clusters (tato technologie je jinak dostupná pouze jako součást Enterprise Edition) možnost nasadit SE na clustery serverů, jenž mohou mít celkovou kapacitu až 4 procesorových jader.
- **Enterprise Edition.** Naprostá špička v oboru databázových systému. Splňuje kritéria kladené těmi nejnáročnějšími aplikacemi, jako např. transakční aplikace, datové sklady provádějící složité dotazy nad velkými množinami dat či robustní internetové aplikace s vysokými nároky na výkon a spolehlivost. Enterprise Edition není nijak omezena z hlediska využívání systémových zdrojů.

2.3 Databáze MySQL

MySQL [3] je databázový systém vyvinutý švédskou společností **MySQL AB**, v současnosti ale práva na MySQL drží společnost **Sun Microsystems**. MySQL je vydáváno pod dvěma licencemi. Kromě **GPL licence**, vychází i pod komerční licencí, která je zpoplatněna. MySQL je dostupné pro širokou škálu platforem (Windows, Linux, atd.). A jelikož jde o poměrně výkonnou databázi, která je navíc volně šiřitelná, získala si značně vysoký podíl mezi v současnosti používanými databázemi. Velmi často ji nalezneme společně v kombinaci s webovým serverem **Apache** a skriptovacím jazykem **PHP**.

Cenou za rychlost MySQL však bylo to, že MySQL neobsahovalo tolik možností a funkcí jako konkurenční produkty. V dnešní době už je však značná část funkcí (podpora triggerů, uložených procedur, …) v MySQL obsažena.

2.4 Jazyk SQL

Jazyk SQL [4] slouží pro organizování, správu a získávání dat, která jsou uložena v databázovém systému. **SQL (Structured Query Language)** znamená strukturovaný dotazovací jazyk. Jazyk SQL je primárně určen pro práci s relačními databázemi. Dotazovací proto, že proces, při kterém požadujeme po databázi data, se nazývá databázový **dotaz (query)**. I když je primární funkcí jazyka SQL získávání dat, poskytuje i další funkce, s jejichž pomocí můžeme řídit databázový systém. Jsou to:

- **Definice dat.** SQL nám umožňuje vytvářet nové struktury, ty existující měnit a umožňuje nám definovat jejich vzájemné vztahy.
- **Manipulace s daty.** SQL nám dává možnost data v databázi aktualizovat tím, že přidáme nová, stávající odstraníme nebo změníme.
- **Řízení přístupu.** Pomocí příkazů jazyka SQL můžeme omezovat schopnosti uživatelů číst či nějakým způsobem modifikovat určitá data v databázi.
- **Sdílení dat.** Umožňuje sdílení tabulek mezi více uživateli.
- **Integrita dat.** Definují tzv. integritní omezení, jenž chrání databázi před porušením z důvodu špatných aktualizací.

Musíme však vzít v úvahu, že jazyk SQL není úplným počítačovým jazykem, jako jsou třeba C/C++ nebo Java. SQL neobsahuje podmínky, neobsahuje žádné příkazy pro řízení běhu programu. Jazyk SQL obsahuje přibližně 40 dílčích příkazů určených pro řízení databáze. Navíc, ač název napovídá něco jiného, jazyk SQL není strukturovaný jazyk, jako klasické programovací jazyky. Spíše se jeho zápis podobá anglicky psaným větám. I přes tuto nepřesnost v názvu, se jazyk SQL stal de facto standardem v relačních databázích, především díky svým mnohým výhodám:

- SQL je nezávislý na výrobci databázového systému,
- angažovanost velkých společností (Microsoft, IBM, Oracle)
- jazyk je postaven na relačních základech,
- integrace s jazykem Java (JDBC),
- snadno srozumitelný zápis příkazů,
- kompletní databázový jazyk.

3 Programovací jazyk Java

3.1 Historie a současnost

Vše spustil zájem komerčních organizací o programovací jazyk, jehož aplikace by mohly běžet na různých operačních systémech a různých typech počítačů, bez nutnosti je znovu a znovu kompilovat. Tohoto úkolu, byť nepřímo, se zhostila společnost Sun Microsystems uvedením jazyka Java. Celý projekt odstartoval v roce 1991 pod pracovním názvem **Zelený projekt (Green project)** [5], jehož cílem bylo vyvinout programovací jazyk, jenž by našel uplatnění v zařízeních spotřební elektroniky či počítačích využívaných v automobilech.

Jeden z vedoucích inženýrů projektu, **James Gosling**, vytvořil jazyk, který zadaná kritéria splňoval. Pojmenoval ho **Oak** (anglicky dub), prý podle stromu, jenž mu rostl před kanceláří. Ale jméno Oak již bylo rezervováno pro jiný programovací jazyk a tak bylo nutné najít jméno nové. A tím jménem byla **Java**.

Původní předpoklad týmu, že hlavním kolbištěm Javy se stanou domácí elektrické spotřebiče, se nenaplnil. V té době potřeba po tomto druhu aplikací prostě neexistovala. Naštěstí v té době zažil boom Internet, a to především díky **WWW (World Wide Web)**. V té době byly klíčovým prvkem komunikace webové stránky napsané v **HTML (HyperText Markup Language)**. Tyto stránky byly však statické. Vývojáři hledali nějaký robustní způsob tvorby dynamických www stránek. Byla zde poptávka po programech, které by byly schopné obstarávat generování stránek, a zároveň zvládali komunikaci s databázovými servery. A na to se vynikajícím způsobem hodila právě Java. Její velkou předností byla možnost spustit jí na téměř jakémkoliv druhu počítače bez nutnosti znovu kompilovat.

Java se oficiálního vypuštění dočkala v roce 1995 a přibližně po čtyřech letech se stala dominantním programovacím jazykem v sektoru velkých podnikových aplikací s vazbou na technologii WWW.

V současnosti je jazyk Java značně rozrostlý a obsahuje několik verzí, označovaných též jako edice. Nejběžněji užívanou edicí je **Java 2 Standard Edition (J2SE)**, standardně využívaná pro tvorbu konzolových aplikací, či aplikací s grafickým rozhraním.

Další edici je **Java 2 Enterprise Edition (J2EE)** využívaná pro tvorbu rozsáhlých podnikových aplikací. Velmi rozšířenou edicí je i **Java 2 Micro Edition (J2ME)**, jenž nachází uplatnění při tvorbě aplikací pro mobilní telefony či digitální osobní asistenty. Nedávno navíc Sun představil úplně novou technologii, kterou pojmenoval **JavaFX**. JavaFX je softwarovou platformou pro tvorbu tzv. **„Rich Internet Applications“**(webová aplikace s některými charakteristickými znaky běžných desktopových aplikací), které mohou běžet na nejrůznějších zařízeních. Současná verze (JavaFX 1.1.1 vydána v březnu 2009) umožňuje spouštění aplikací v klasických prohlížečích a mobilních telefonech, do budoucna přijdou na řadu set-top boxy či herní konzole.

3.2 Architektura jazyka

Na Javu se velmi často nahlíží pouze jako na programovací jazyk, pomocí kterého lze vyvíjet aplikace. Ale samotný programovací jazyk je jen jedna z mnoha součástí Javy. Java má tzv. **podkladovou architekturu**, což jí poskytuje značné množství výhod. Jednou z výhod je nezávislost na použité platformě. Celá architektura se dá shrnout do čtyř součástí [6]:

- Samotný programovací jazyk Java,
- formát souboru **.class** (Java bytecode),
- aplikačního programového rozhraní Javy (API),
- virtuálního stroje jazyka Java.

Pokud píšeme v Javě program, provádí se jeho zápis pomocí programovacího jazyka Java. Takto napsaný program je uložen v souboru s příponou **.java**. A tady přichází základní rozdíl oproti jazykům, jako jsou C++ a některé další vysokoúrovňové programovací jazyky. Nedochází zde ke kompilaci do objektového kódu. Na místo toho se kompiluje do bajtového kódu a výsledek je uložen do souboru s příponou **.class**. Takto vygenerovaný bajtový kód je poté interpretován pomocí **virtuálního stroje Javy (JVM – Java Virtual Machine)**. Virtuální stroj překládá bajtový kód do strojového kódu běžícího na daném počítači. Protože kompilátor produkuje bajtový kód, může být výsledný program interpretován libovolným JVM bez ohledu na to, na jaké platformě je počítač, na kterém JVM běží, postaven.

Rozhraní pro programování Java aplikací (též označovaného jako **Java API**) je předem připravený kód, jenž je tematický roztříděn do jednotlivých balíčků. Java API obsahuje například třídy pro tvorbu GUI (v případě J2SE), či třídy pro tvorbu servetů (J2EE).

3.3 Automatická správa paměti

Java využívá k ukládání dynamických nebo dočasných dat tzv. **haldu** (oblast dočasné paměti). Pokud už paměť nepotřebujeme, měli bychom ji vrátit zpět operačnímu systému, tak aby ji bylo možné využít i pro ostatní běžící programy. Pokud bychom tuto paměť neuvolňovali zpět, docházelo by k tzv. **memory leaku**, jevu při kterém dochází k vyčerpávání paměťových zdrojů. A to proto, že pokud alokujeme objektu paměť, avšak poté jí zapomeneme uvolnit, je tato část paměti pro operační systém nepoužitelná.

Proto, aby se zabránilo plýtvaní s pamětí, Java implementuje mechanismus **čištění dynamické pamětí (tzv. grabage collector)** [6]. V podstatě se jedná o kód, který v pravidelných intervalech prohledává paměť a snaží se v ní najít objekty, které již delší dobu nebyly využívány či dokonce už neexistují (neexistuje na ně odkaz v programu). K tomu Java využívá tzv. **měkkých ukazatelů** (**soft pointers**), které sledují odkazy k objektům. Čistič dynamické paměti pak při svém spuštění pak pomocí měkkých odkazů ověřuje platnost všech objektů. Čistič dynamické paměti je spouštěn automaticky, avšak můžeme jeho spuštění vynutit pomocí statické metody `gc()`, kterou nalezneme ve třídě `System`.

3.4 Charakteristika programovacího jazyka Java

Programovací jazyk Java [6][7] je objektově orientovaný jazyk, který vychází z C++, k němuž má syntakticky velmi blízko, pokud tedy nepočítáme C#, který byl ovšem uveden na trh až po Javě. Oproti C++ neobsahuje Java některé konstrukce, které mnohým programátorům způsobovali problémy. Především se jedná o ukazatele,

které Java neimplementuje. Proto se dá říci, že začínající programátoři budou v Javě dělat méně chyb. Java navíc obsahuje mnoho zajímavých vylepšení, jako například:

- O přidělování a uvolňování pamětí se stará sama Java (využívá k tomu garbage colector). Objekty už nejsou rušeny pomocí `delete`, ale pokud objektu přiřadíme neplatnou referenci `null`, v podstatě ho tím označíme jako zrušitelný a garbage colector ho odstraní
- S výjimkou několika málo primitivních datových typů, je Java plně objektová.
- V Javě neexistují ukazatelé. Jsou nahrazeny referencemi, přičemž o dereferencování se stará kompilátor. Odpadá tím spoustu problémů s neplatným pokusem o přístup do paměti.
- Java obsahuje sofistikovaný mechanismus výjimek, jenž umožňuje veškeré běhové chyby odchytit a adekvátně na ně reagovat.
- Umožňuje serializaci, jež umožňuje uložit aktuální stav objektu do souboru.
- Podpora programování vícevláknových aplikací.
- Java obsahuje velmi rozsáhlou standardní knihovnu. Obsahuje například třídy pro tvorbu GUI aplikací, práci se soubory, připojení k databázi či pro práci s XML

3.5 Java aplikace a GUI

Když Java vyšla ve verzi 1.0, její grafická část [8] byla velmi nedokonalá a neohrabaná. Proto si v té době velká část vývojářů vytvořila určitou averzi k tvorbě Java GUI aplikací. Ale v průběhu času se GUI dočkalo rozsáhlých vylepšení a dnes již máme tři různé implementace grafického rozhraní, které si právě teď představíme.

3.5.1 AWT

Abstract Window Toolkit (AWT) se v Javě objevil jako první. Ale už od počátku byl kritizován za svojí nesystematičnost a nepříjemné ovládání. Sice rozhraní bylo několikrát vylepšeno a inovováno, přesto velká část nevýhod přetrvala. Základní myšlenou AWT bylo to, že každá Java komponenta má v hostitelském operačním systému svůj nativní protějšek. Ač by se mělo jednat o abstraktní rozhraní, přenositelné mezi

různými systémy, přenositelnost byla velmi problematická z důvodů rozdílných vlastností nativní úrovně GUI. Dnes už se AWT prakticky k tvorbě GUI aplikací nepoužívá. Co se ale z AWT stále používá, je balík `java.awt.event`. Tento balík obsahuje třídy a rozhraní zodpovědná za zachytávání a zpracovávání událostí generovaných GUI komponentami.

3.5.2 Swing

Java ve verzi 1.2 přišla s novou grafickou knihovnou, která byla nazvána **Java Foundation Classes (JFC)**, známou spíše jako **Swing**. Tvůrci Javy zavrhli původní koncepci AWT, a Swing byl kompletně napsán od začátku (s výjimkou AWT API, které Swing obsahuje pro obsluhu událostí). Základní vlastnosti Swingu:

- Je kompletně napsán v Javě a už nevyužívá mechanismu propojení s nativními komponentami operačního systému.
- Na platformě zcela nezávislý
- Oddělení funkcionality od vzhledu **GUI (tzv. look and feel)**. Možnost změny vzhledu aplikace pomocí vzhledů ze standardní knihovny, či možnost tvorby vlastních vzhledů
- U složitých komponent, jako třeba strom nebo tabulka došlo k oddělení dat od samotných komponent GUI. Využívá se tzv. modelů, což činí aplikace pružnější a jejich kód znovupoužitelný.

Díky těmto vlastnostem se tvorba GUI pomocí Swingu stala mnohem příjemnější. Jedinou vadou na kráse Swingu jsou jeho relativně vyšší nároky na výkon, především co se paměti týče.

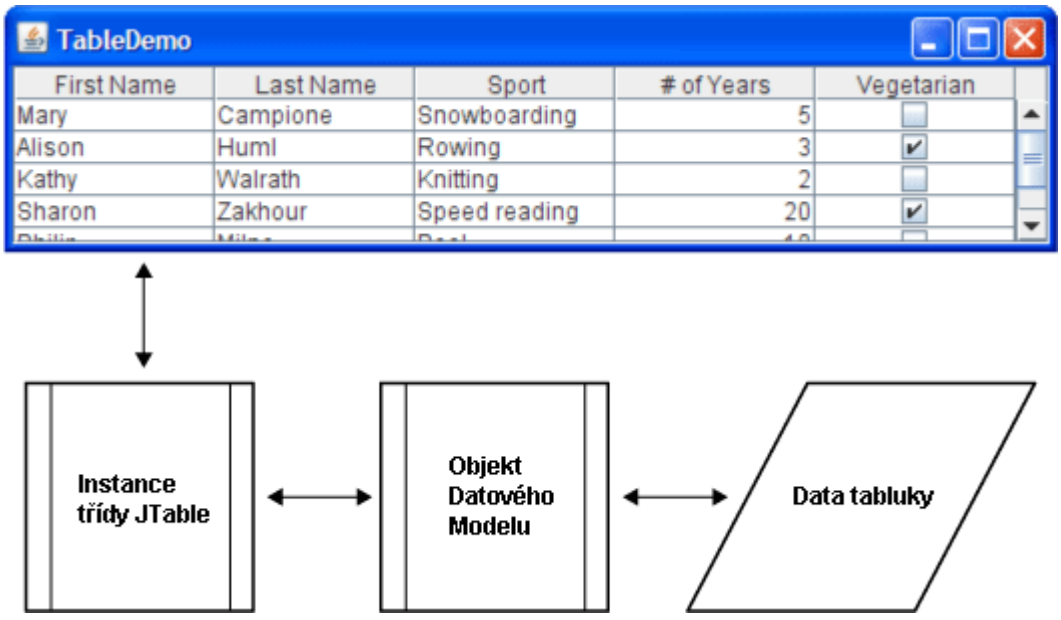
3.5.3 SWT

Standard Widget Toolkit je knihovna, jež se v mnohém velmi podobá knihovně Swing. Původním cílem bylo vyvinout GUI knihovnu, jenž má nižší systémové nároky. SWT znají především uživatelé **Eclipse IDE**, či uživatelé aplikacích postavených na Eclipse platformě. Bohužel původní záměr, nižší systémové nároky, se nepodařilo naplnit. Aplikace postavené na SWT jsou přibližně stejně náročné, jako ty Swingové.

3.5.4 Třída JTable

`JTable` [9] je třída a zároveň komponenta sloužící pro zobrazení dat v tabulkové podobě. Třída `JTable` se nachází v balíčku `javax.swing`. `JTable` neumožňuje data pouze zobrazovat, ale umožňuje i jejich snadnou editaci. Ale hlavní funkcí `JTable` je zobrazení dat. Data je nutné nejprve připravit v tzv. datovém modelu, který pak předáme k zobrazení tabulce. Obdobně jako je tomu u většiny komponent Swingu, je i `JTable` založen na architektuře **model/pohled/radič (MVC – model/view/controller)**. Právě proto je asi nejdůležitější součástí `JTable` datový model.

Datový model (obrázek 1) je jedno z nejdůležitějších rozhraní třídy `JTable` a je definován rozhraním `TableModel`. Je v něm definováno rozhraní mezí instancí třídy `JTable` a samotným datovým modelem. Tato architektura je velmi flexibilní a umožňuje nám vytvářet aplikace se znovupoužitelným kódem. Trošku nevýhodu této koncepce je, že práce s rozhraním `TableModel`, především pro začátečníky, je poměrně nepřehledná a komplikovaná.



Obrázek 1 Model fungovaní JTable. Zdroj [9]

Z uvedeného obrázku 1 je tedy patrné, že rozhraní `TableModel` je zodpovědné za naplnění tabulky daty. Rozhraní ovšem poskytuje i některé další metody, jež poskytují o tabulce různé informace, jako třeba:

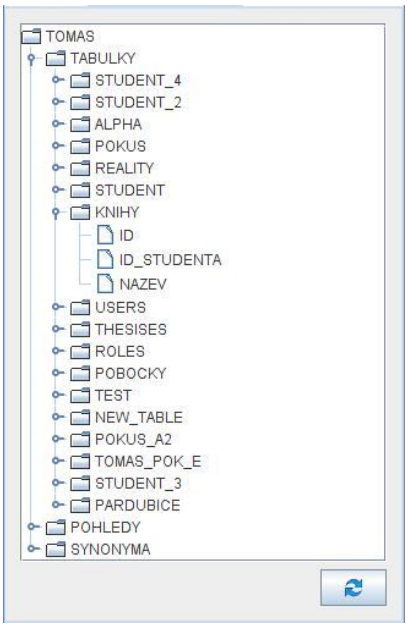
- informace o rozměrech tabulky (počet řádků, počet sloupců),
- typ dat v jednotlivých sloupcích tabulky
- názvy jednotlivých sloupců, které je možné zobrazit

Ve zdrojovém kódu v kapitole 6.3 je uveden příklad, který vytvoří data pro tabulku z výsledku dotazu do databáze.

3.5.5 Třída JTree

Tuto třídu (komponentu) [10] běžně používáme, pokud potřebujeme zobrazit nějakou množinu dat, která má hierarchické uspořádání (obrázek 2). Typickým příkladem může být souborový systém, kdy každá složka může obsahovat další složky a ty zas další složky. U stromů je důležité si ujasnit základní pojmy:

- **uzel (node) –** uzlem označujeme každou zobrazenou položku uvnitř stromu,
- **kořenový uzel (root node) –** tak označujeme uzel, který je umístěn nad všemi ostatními,
- **uzel rozvětvení (branch node) –** uzel, který obsahuje, nebo může obsahovat další uzly,
- **koncový uzel, list (leaf node) –** uzel, který neobsahuje, nebo nesmí obsahovat žádné listy,
- **rodičovský uzel (parent) –** jedná se o nadřazený uzel,
- **dceřiný uzel (child) –** podřazený uzel,
- **sourozenci (siblings) –** tak se nazývají uzly na stejné úrovni



Obrázek 2 Příklad použití JTree. Zdroj vlastní.

Stejně tak jako u jiných komponent definovaných v knihovně Swing, je i instance třídy `JTable` dalším typickým zástupcem komponenty typu pohled. Ostatní objekty tvoří model. To znamená, že veškerá data zobrazovaná pomocí instance `JTree` jsou uložena v modelu. Pokud tedy chceme data zobrazit ve stromu, musí jejich datový model implementovat rozhraní `TreeModel` (`javax.swing.tree`).

Cest jak vytvořit funkční `JTree` komponentu je hned několik. Od prostého předání parametrů konstruktoru při vytváření, až po předání kompletního `TreeModelu` instanci `JTree`. Ve své práci jsem si vybral řešení, při kterém jsem nejdříve vytvořil samotné uzly (pomocí třídy `DefaultMutableTreeNode`), které jsem přesně podle svých představ vzájemně propojil. Takto propojené uzly jsem předal jako parametr při vytváření instance třídy `TreeModel` a samotnou instanci `TreeModel` jsem potom předal k zobrazení instanci `JTree`.

4 Připojení databáze (JDBC)

JDBC (Java Database Connectivity) poskytuje základní rozhraní pro jednotný přístup k databázím v jazyce Java. Základní koncepcí JDBC je snaha odstínit programátora od specifických API databází. Sun zahrnul JDBC do Java API, a tím způsobil, že ve světě Javy je JDBC prakticky standardem pro přístup k databázi a že prakticky všichni velcí hráči na trhu s databázemi nabízejí podporu pro toto rozhraní.

4.1 Historie JDBC

První verze, JDBC 1.0 [6], obsahovala pouze ty nejzákladnější a nejnutnější funkce pro přístup do databáze a pro přístup k datům. Jádrem celého API byl správce připojení, jenž byl schopen rozeznávat připojení k různým databázím a byl schopen řídit paralelní připojení k jednotlivým databázím.

Značného rozšíření se JDBC dočkalo ve verzi 2.0. Mezi rozšíření patřila:

- Dávkové operace. V rámci jediného volání můžeme databázovému stroji předat ke zpracování více příkazů. Tím zvyšujeme efektivitu a výkonnost, především při operacích s velkými objemy dat.
- Možnost pohybovat se ve výsledných množinách dat dopředu i dozadu.
- Možnost aktualizovat výsledky. Můžeme tak aktualizovat určitý řádek či přidat řádek nový podle aktuální pozice ve výsledné množině dat.
- Možnost sdílet připojení k databázi mezi více Java aplikacemi díky zásobníku připojení

V únoru roku 2002 byla představena verze JDBC 3.0 a Sun ji okamžitě zakomponoval do J2SE 1.4. Mezi charakteristické vlastnosti verze 3.0 patří:

- Pomocí návratových bodů je možné se vrátit k určitému označenému bodu uvnitř transakce, a zrušit tedy pouze jenom její část.
- Možnost ponechat otevřené kurzory i po skončení transakce.

4.2 Typy ovladačů

Ovladač [11] rozhraní se nachází v balíčku `java.sql`. V tomto rozhraní je definován způsob přístupu aplikace k relačním databázím. Je zde definováno rozhraní `Driver`, jenž obsahuje metodu používanou k vytvoření databázového připojení. Často se můžeme setkat s tím, že za ovladač JDBC se označuje právě implementace rozhraní `Driver`. Ale toto označení se hodí spíše pro skupinu souvisejících tříd a rozhraní poskytujících připojení k **systému správy databází (DBMS)**. Ovladač můžeme získat z rozličných zdrojů. Většina velkých či známých výrobců databází poskytují alespoň jeden ovladač pro komunikaci s databází. Ve valné většině případů jej poskytují bezplatně. Na adrese http://industry.java.sun.com/products/jdbc/drivers naleznete seznam dostupných ovladačů. Naleznete zde i případné informace o podpoře specifických vlastností JDBC u jednotlivých výrobců.

Ovladače JDBC dělíme do čtyř kategorií, podle způsobu poskytování připojení k databázi. Každý typ ovladače má své výhody a nevýhody. Proto někteří dodavatelé poskytují více druhů ovladačů pro připojení do své databáze.

- **Typ 1 –** Připojení prostřednictvím zdroje dat ODBC. ODBC bývá standardně dodáváno s operačními systémy společnosti Microsoft. Výhodou je, že jedním ovladačem se lze připojit do jakékoliv databáze, která je dostupná rozhraním ODBC. Za nevýhodu lze považovat nižší výkon a horší přenositelnost aplikací.
- **Typ 2 –** Někdy nazýván jako připojení prostřednictvím kódu nativního klienta pro přístup k síti. Tento ovladač obsahuje jak kód v jazyce Java, tak i nativní kód, jenž je zodpovědný za komunikaci s příslušným databázovým klientem. Výhodu tohoto typu ovladače je vyšší výkon než u typu 1. Ale opět trpí problémy s přenositelností. Musíme zajistit, aby na cílovém systému běžel klientský síťový software daného databázového systému.
- **Typ 3 –** Při připojení tímto typem připojení odesílá ovladač databázové příkazy pomocí transportní síťové vrstvy, jenž je nezávislá na konkrétním databázovém systému. Nevýhodu tohoto typu připojení je nutnost existence serverové komponenty, která je zodpovědná za překlad databázových dotazů do formátu srozumitelného pro konkrétní databázový systém. Výhodu je že

můžeme kdykoliv na serveru změnit databázový systém, aniž bychom nějak museli měnit kód klientské aplikace.

- **Typ 4 –** Tento typ ovladače je kompletně napsaný v jazyce Java. Pro komunikaci s databází používá protokol, jenž je vždy specifický pro daný databázový systém. Výhodou tohoto typu ovladače je snadnost jeho použití a vynikající přenositelnost (ovladač je kompletně napsán v Javě) aplikací, které tento typ ovladače využívají. Za nevýhodu lze považovat nutnost používat různé ovladače pro různé druhy databází.

4.3 Rozhraní JDBC

Jelikož je Java objektově orientovaný jazyk, není nic překvapivého na tom, že JDBC API využívá strukturu databázově orientovaných objektů. Objekty můžeme rozdělit na šest základních skupin:

- **Objekt správce ovladačů.**
- **Objekty připojení.** Objekty, které reprezentují jednotlivá připojení.
- **Objekty příkazů.** Umožňují vykonávat příkazy jazyka SQL na aktivních připojeních.
- **Objekty množin výsledků.** Obsahují data zaslaná jako odpověď na databázový dotaz.
- **Objekty metadat.** Obsahují informace o databázi, tabulkách či jednotlivých sloupcích.
- **Objekty výjimek.** Obsahují informace o chybách, které nastaly v průběhu připojení k databázi.

4.3.1 Připojení k databázi

Za hlavní rozhraní JDBC je považován **objekt správce** `DriverManager` [6]. Po načtení ovladače JDBC pro konkrétní databázi (obvykle pomocí metody `Class.forName()`), požádáme objekt `DriverManger` o připojení pomocí tohoto ovladače. K tomu se využívá metoda `getConnection()`. Příklad použití:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url = " jdbc.odbc.Ukazka";
Connection con = DriverManager.getConnection(url, "dba", "sql" );
```

Metoda `getConnection()` vrací rozhraní typu `Connection`, které řídí komunikaci mezi programem a ovladačem a na které se budeme odkazovat při používání databáze. Při nepovedeném pokusu o přístup do databáze `DriverManager` vyvolá výjimku.

Mezi nejdůležitější funkce rozhraní `Connection`, tedy kromě řízení přístupu k databázi, patří i tvorba SQL dotazů pro zpracování v databázi či řízení transakcí v průběhu připojení.

Pokud chceme v databázi vykonat nějaký dotaz, musíme nejprve vytvořit tzv. objekt příkazu. To provedeme pomocí metody `createStatement()` obsaženou v rozhraní `Connection` jež vytvoří objekt příkazu, který je typu `Statement`, jehož hlavní funkcí je vykonávání příkazů SQL. Existuje několik možností jak požadovaný dotaz provést. Pokud chceme vykonat dotazy, po nichž nám nejsou vráceny žádné výsledky (UPDATE, INSERT, COMMIT, CREATE TABLE, DELETE), můžeme využít metody `executeUpdate()`. Pokud ale chceme vykonat dotaz (SELECT), použijeme metodu `executeQuery()`, jenž vrací výsledkovou množinu (`ResultSet`).

4.3.2 Získání dat z databázového systému

Pro prácí s výsledky nám JDBC nabízí objekt množiny výsledků `ResultSet` [6]. Instance třídy `ResultSet` je výsledkem úspěšného volání metody `executeQuery()`. Ve třídě `ResultSet` jsou ukryta data, které nám databázový systém vrátil jako odpověď na náš dotaz. Třída nám umožňuje zpracovávat výsledky po **jednotlivých řádcích (záznamech)**. Abychom ale mohli zpracovat data z konkrétního řádku, je nutné se na konkrétní záznam přesunout. O tom, se kterým záznamem můžeme pracovat, nás informuje **aktuální řádek (current row)**, někdy též nazývaný **aktuální záznam (current record)**. V **množině řádků (záznamů)** smí být aktivní pouze vždy

pouze jeden řádek. Jakmile je vytvořena instance třídy `ResultSet`, je kurzor nastaven na první záznam v množině. Viz jednoduchý přiklad:

```
try {
      //Zavedení ovladače
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      //Adresa umistění DB systému
      String url = " jdbc.odbc.Ukazka";

      //vytvoření připojení
      Connection con = DriverManager.getConnection(url, "dba", "sql" );

      //tvorba objektu příkazu
      Statement stmt = con.createStatement();

      //SQL dotaz
      String sql = "SELECT * FROM KLIENTI";

      //vykonání dotazu a uložení výsledků do objektu množiny výsledků
      ResultSet set = stmt.executeQuery(sql);

} catch(Exception e) {}
```

Velmi zajímavé je, že data vrácená databázovým systémem po vykonání dotazu zpravidla nebývají fyzicky uloženy v instanci typu `ResultSet`. Data se obvykle přenáší až v okamžiku, kdy se z daného řádku v množině stane aktivní záznam. Záznam je potom přesunut do **lokální mezipaměti** objektu `ResultSet`. Díky tomuto mechanismu jsme schopní vykonávat dotazy vracející větší objemy dat než by byla lokální mezipaměť schopna obsáhnout.

4.3.3 Třída PreparedStatement

Pokud chceme v programu vykonat jakýkoliv SQL dotaz, musí ho ovladač JDBC zkompilovat a potom ho odeslat databázovému systému. Nezřídka se stává, že potřebujeme vykonat sérii velmi podobných SQL příkazů, jež se například liší jen v hodnotách argumentů. Viz příklad:

```
Statement stmt = conn.createStatement();

stmt.executeUpdate(
"UPDATE HRACI SET GOLY=24 WHERE JMENO='Jan Kolar'");

stmt.executeUpdate(
"UPDATE HRACI SET GOLY=12 WHERE JMENO='Jan Stary'");

stmt.executeUpdate(
"UPDATE HRACI SET GOLY=8 WHERE JMENO='Ales Pisa'");
```

Pokud budeme takových dotazů vykonávat příliš mnoho, znatelně tím snížíme výkon naší aplikace. Právě pro tento případ obsahuje JDBC třídu `PreparedStatement` [6], jenž je potomkem třídy `Statement`. Díky této třídě bude příkaz přeložen jen jednou a poté již budeme používat jen tzv. **substituční parametry (substitution parameters)**. Pro výše uvedený příklad by se instance třídy `PreparedStatement` a následné odeslání příkazů mohlo zapsat takto:

```
PreparedStatement pstmt = conn.prepareStatement(
  "UPDATE HRACI SET GOLY = ? WHERE JMENO= ? ");

pstmt.setInt(1,24);
pstmt.setString(2,"Jan Kolar");
pstmt.executeUpdate();

pstmt.setInt(1,12);
pstmt.setString(2,"Jan Stary");
pstmt.executeUpdate();

pstmt.setInt(1,8);
pstmt.setString(2,"Ales Pisa");
pstmt.executeUpdate();
```

Otazníky uvedené v příkazu při vytváření instance třídy `PreparedStatement` jsou právě zmiňované substituční parametry. Pro úpravu substitučních parametrů se používají metody `setXXX()` (kde xxx je datový typ parametru) ze třídy `PreparedStatement`. Metoda `setXXX()` má dva parametry. Prvním je celočíselná hodnota, která reprezentuje pořadí substitučního parametru (otazníku) v dotazu (čísluje se od jedničky). Druhým je samotná hodnota. Zde je důležité, abychom správně zvolili datový typ parametru a tím i správnou metodu `setXXX()`. Pokud jsme nastavily již

všechny potřebné parametry, můžeme příkaz odeslat databázovému systému metodou executeUpdate, nebo executeQuery, v závislosti na tom, zda očekáváme nějaký výsledek.

4.3.4 Práce s třídou ResultSet

O třídě ResultSet jsem se již zmínil. Nyní ji ale proberu více do hloubky. Instanci třídy ResultSet získáme od metody executeQuery() a je v ní obsažen výsledek našeho dotazu.

K procházení mezi jednotlivými záznamy nám slouží metoda next(). Metoda nepřímá žádné argumenty a vrátí nám booleovskou hodnotu, která nám indikuje, zda metoda našla další záznam. Pokud obdržíme hodnotu false, znamená to, že kurzor se nastavil až za poslední záznam v instanci třídy ResultSet. Malá ukázka jak projít všechny záznamy načtené v instanci třídy ResultSet:

```
ResultSet rset = stmt.executeQuery("SELECT * FROM TABULKA");
while(rset.next()) {
   // zde bude kód pro obsluhu jednotlivých záznamů
}
```

Pro načtení dat z aktuálního záznamu nám slouží funkce getXXX(), která je definována ve třídě ResultSet. Pro každý datový typ je zde tedy speciální metoda. Takže například metoda getString() nám vrací instanci třídy String. Těchto metod využíváme, pokud již v předstihu známe datový typ sloupce. Pokud jej neznáme, nebo si nejsme jisti, můžeme využít metody getObject(), jež nám vrací instanci typu Object, jenž je vhodná pro všechny typy dat ukládaných v databázových systémech.

Každá metoda getXXX() se ve třídě ResultSet se nachází ve dvou verzích. První z nich přijímá jako argument celé číslo, jenž vyjadřuje index sloupce. Druhá přijímá argument typu String, jenž vyjadřuje název sloupce uloženého v instanci ResultSet. Proto:

```
ResultSet rset;
int id1 = rset.getInt(1);
int id2 = rset.getInt("ID");
```

Vrátí stejnou hodnotu. Tedy za předpokladu, že sloupec s indexem 1 se jmenuje ID.

4.3.5 Metadata

Metadata [5][6]se používají pro zjišťování informací o databázovém systému či ke zkoumání obsahu databáze. V JDBC jsou definovány dvě rozhraní pracující s metadaty a to `DatabaseMetadata` a `ResultSetMetaData`.

`DatabaseMetadata` se používá k popisu databázového systému. Metadata získáme tím, že zavoláme metodu `getMetaData()` definovanou ve třídě `Connection`. `DatabaseMetadata` pak obsahuje mnoho metod pro získávání jednotlivých dílčích informací o databázovém systému. Nejběžněji se používají následující metody:

- `getDatabaseProductName()` vrátí název databáze.
- `getUserName()` vrací uživatelské jméno přihlášeného uživatele.
- `getURL()` vrací adresu databáze.
- `getSchemas()` vrací názvy všech databázových schémat, která jsou v databázi použitelné.
- `getTables()` vrací názvy všech tabulek v databázi.

Druhým zástupcem metadat jsou metadata vrácena spolu s výsledkem dotazu do databáze. Abychom k nim získali přístup, musíme je získat pomocí metody `getMetaData()`, která je definována ve třídě `ResultSet`. Tato metoda nám vrátí instanci třídy `ResultSetMetaData` jež také obsahuje metody pro získání jednotlivých informací. Nejčastěji se využívají:

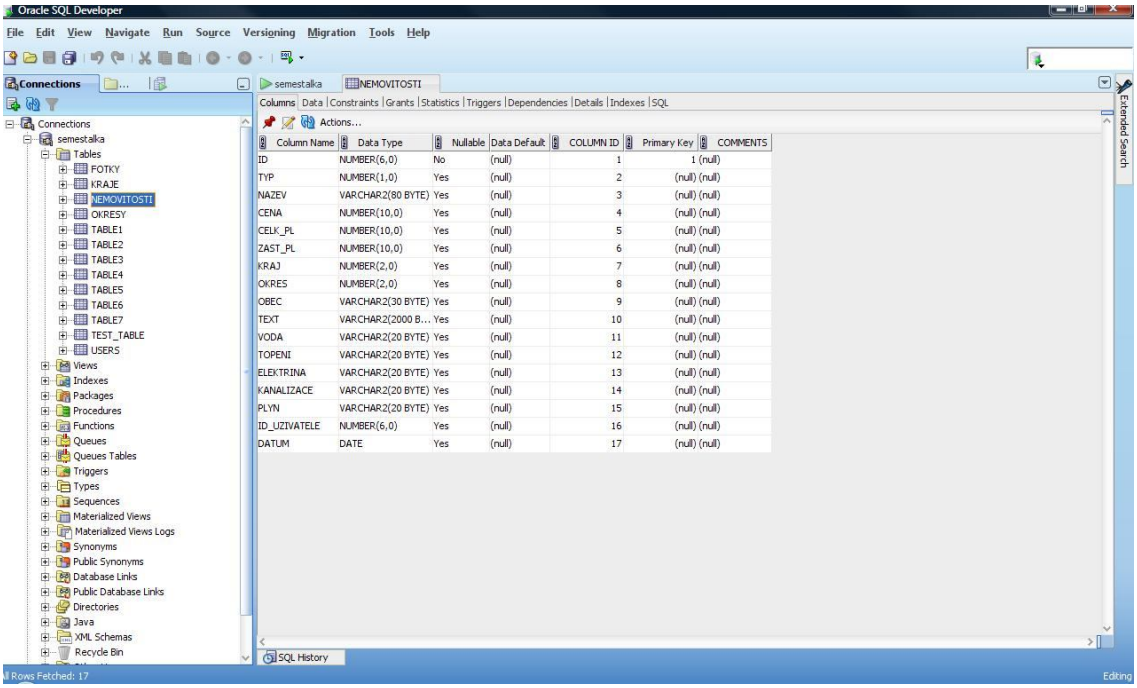
- `getColumnCount()` vrátí informaci o počtu sloupců ve výsledné sadě záznamů.

- `getColumnName(int index)` vrátí název sloupce odpovídajícího zadanému indexu
- `getColumnType(int index)` vrátí jakého datového typu je sloupec se specifikovaným indexem

5 Aplikace pro správu databází

5.1 Oracle SQL Developer

SQL Developer (obrázek 3) je grafický nástroj od společnosti Oracle pro správu a vývoj databázových aplikací v databázích společnosti Oracle. Umožňuje nám vytvářet různé databázové objekty a umožňuje nám spouštět databázové příkazy a skripty. SQL Developer je kompletně napsán v Javě, tudíž je spustitelný všude tam, kde se nachází JVM. Navíc je k dispozici úplně zdarma.

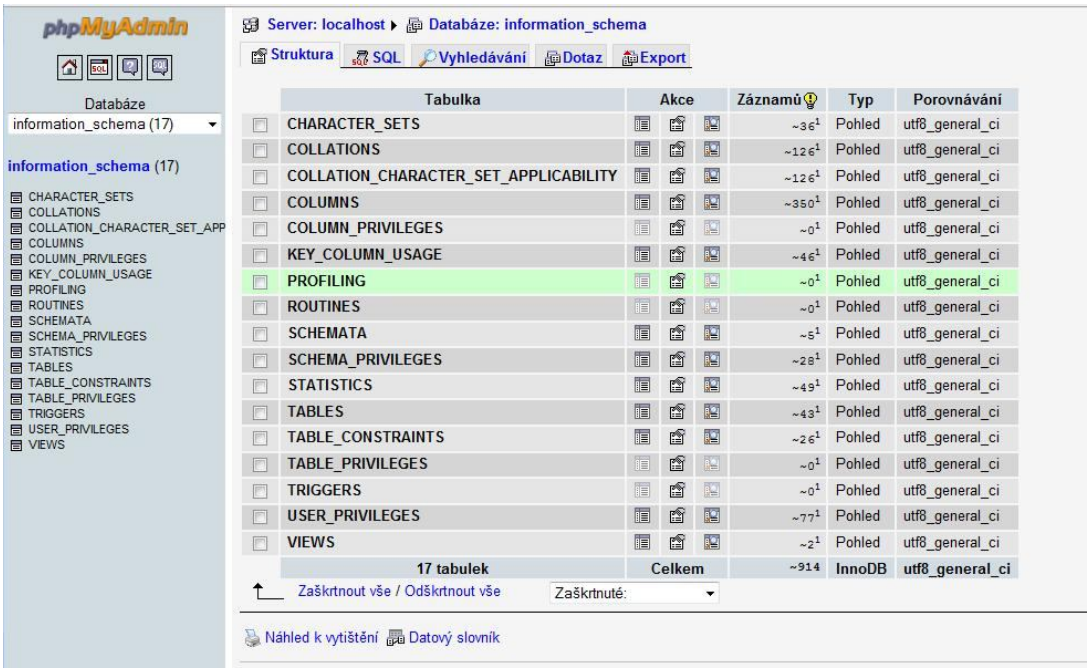


Obrázek 3 Oracle SQL Developer. Zdroj vlastní.

Oracle SQL Developer se může připojit do libovolné databáze společnosti Oracle, ovšem ve verzi 9.2.0.1 a vyšší běžící na platformách Windows, Linux nebo Mac.

5.2 PHPMyAdmin

PHPMyAdmin (obrázek 4) je nástroj kompletně napsaný v jazyce PHP (tudíž je k jeho provozu potřeba webserveru s podporou PHP), který umožňuje komplexní a intuitivní správu obsahu databáze. Pro MySQL se jedná asi o nejpopulárnější nástroj. PHPMyAdmin je dostupný v 52 jazycích.

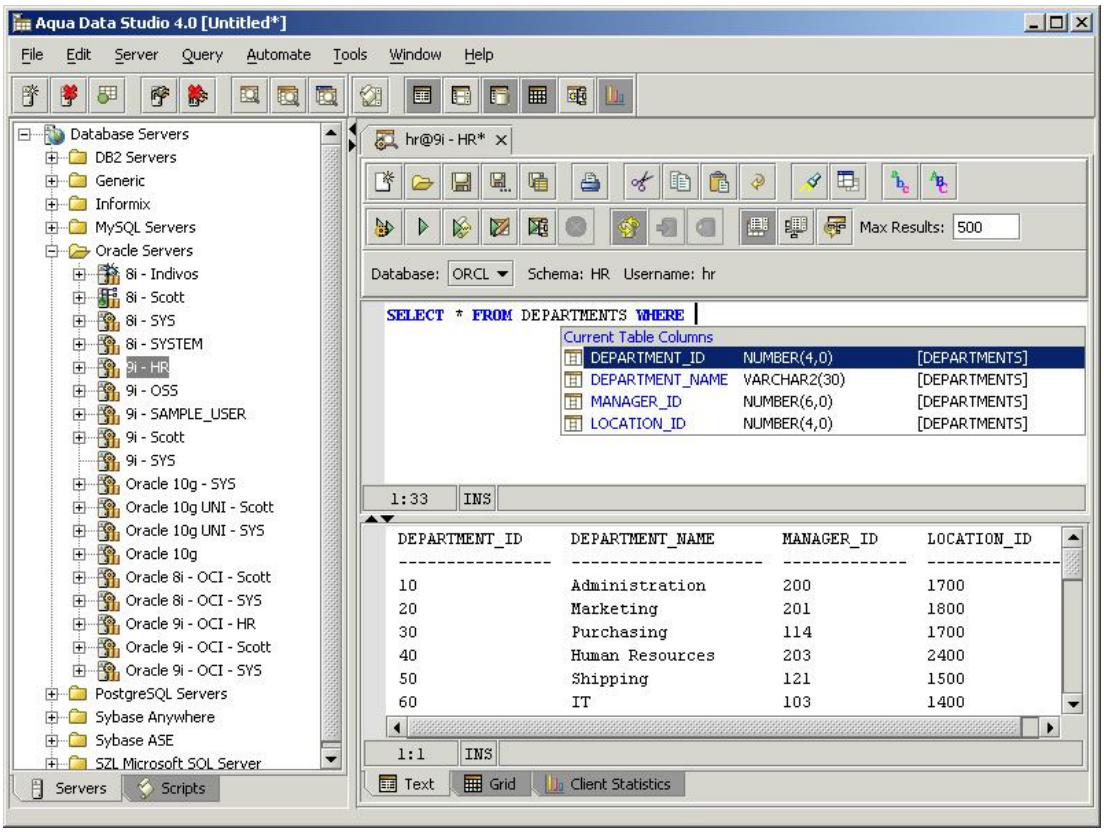


Obrázek 4 PHPMyAdmin. Zdroj vlastní.

PhpMyAdmin existuje i ve verzích pro jiné databázové systémy. PhpPgAdmin je určen pro správu databáze typu PostgreSQL. Zpočátku byl vyvíjen jako věrná kopie phpMyAdminu, nyní jde však kompletně o samostatný produkt. Nástroj phpMSAdmin je nástroj určený pro správu databázového systému Microsoft SQL Server, který má ovšem s phpMyAdminem společný pouze vzhled, jedná se totiž o kompletně od základů napsaný program.

5.3 Aqua Data Studio

Aqua Data Studio [12](obrázek 5) je kompletní vývojové prostředí pro vývoj databázových aplikací. Mezi velké výhody tohoto nástroje patří podpora velkého počtu relačních databází. Jen namátkou: Databáze Oracle, DB2 od IBM, MS SQL Server, Sybase nebo MySQL. Aqua Data Studio je vyvíjen společností AquaFold a je dispozici pro velké množství platforem (od Windows až po MacOS).



Obrázek 5 Aqua Data Studio. Zdroj [12].

Mezi další přednosti programu patří jednoduché a přehledné rozhraní pro vkládání dotazů, doplňování názvů tabulek a sloupců v průběhu tvorby skriptu, generování ER diagramů, podpora verzovacích systémů (CVS a Subversion) nebo to že Personal verze je k dispozici zdarma bez dalších omezení.

6 SQL Manažer

Praktická část bakalářské práce je zaměřena na tvorbu aplikace, schopnou se připojit do databáze, procházet její obsah, upravovat ho a vykonávat základní úkony spojené se správou databáze. Aplikace dostala název SQL Manažer. Aplikace je postavena na platformě J2SE a jedná se o standardní GUI aplikaci. Aplikace slouží jako demonstrace možností rozhraní JDBC. Ukazuje jak rozhraní JDBC usnadňuje tvorbu databázově orientovaných aplikací.

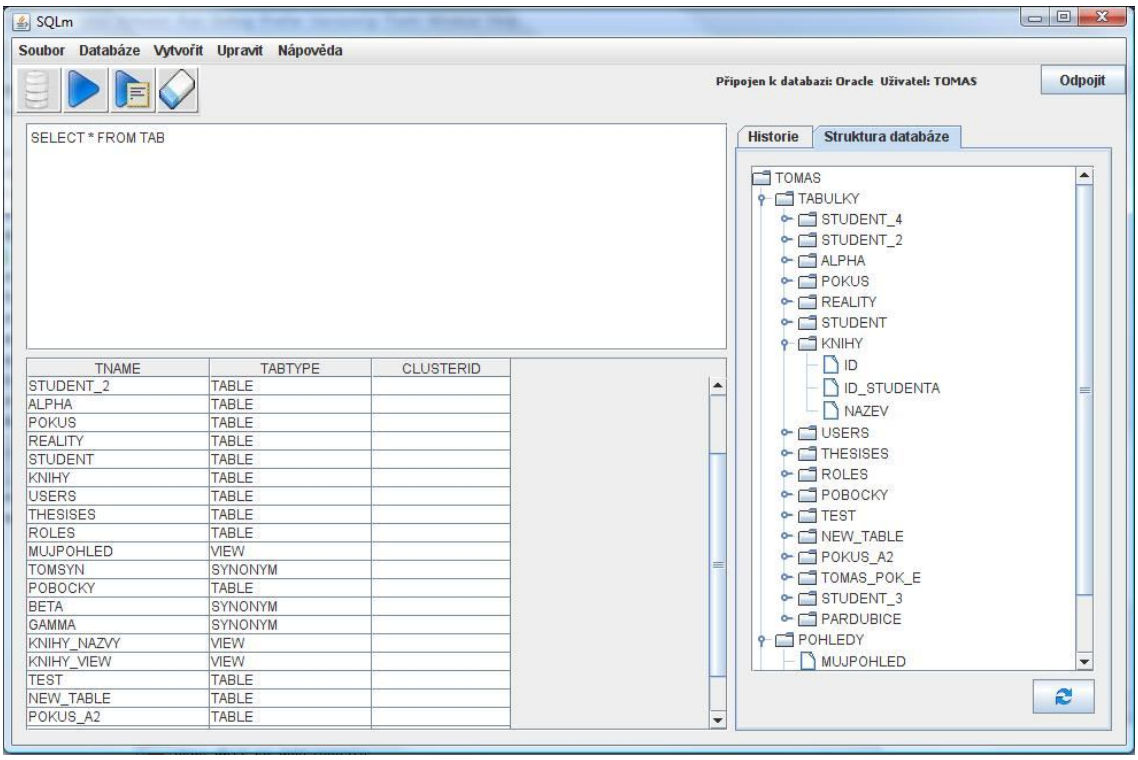
6.1 Funkce a možnosti aplikace

Aplikace SQL Manažer byla vyvíjena především pro účely této práce. Proto se soustředí na prezentaci jen některých možností jazyka Java a rozhraní JDBC. Některé možnosti musely být, ať už z důvodu časových, či z důvodu velmi náročné implementace, vynechány. Aplikace byla primárně určena pro správu databáze Oracle. Ovšem podporuje i přístup do databáze MySQL. I když v případě MySQL nenabízí takové možnosti správy, jako je tomu v případě Oracle. To je způsobeno tím, že ač Oracle i MySQL jsou postaveny na standardu SQL, obě ho neimplementují doslovně. To ovšem potom způsobuje, že některé funkce určené pro databázi Oracle, jsou nekompatibilní s funkcemi databáze MySQL. Z tohoto důvodu jsou některé funkce (např. tvorba sekvencí) aplikace dostupné pouze při připojení do databáze Oracle.

6.1.1 Základní uživatelské rozhraní

Základním prvkem uživatelského rozhraní (obrázek 6) je textové pole pro zadávání SQL příkazů. Do pole můžeme uvádět jednotlivé dotazy či celé skripty. U skriptů je důležité, aby jednotlivé příkazy skriptu byly od sebe odděleny středníkem. Aby uživatel nemusel psát stále stejné příkazy (SELECT * FROM, INSERT INTO, DELETE FROM…), umožňuje aplikace napsáním počátečního písmena a stisku klávesy ALT požadovaný kód doplnit. Takže po napsání „s“ a stisku klávesy ALT se nám „s“ rozvine na SELECT * FROM. Příkaz (příkazy) v textovém poli můžeme vykonávat různě. Můžeme buď vykonat pouze jediný, samostatně napsaný příkaz, nebo můžeme

z více označených příkazů nechat vykonat pouze ten označený. Třetí možností je nechat vykonat více příkazů najednou jako skript.



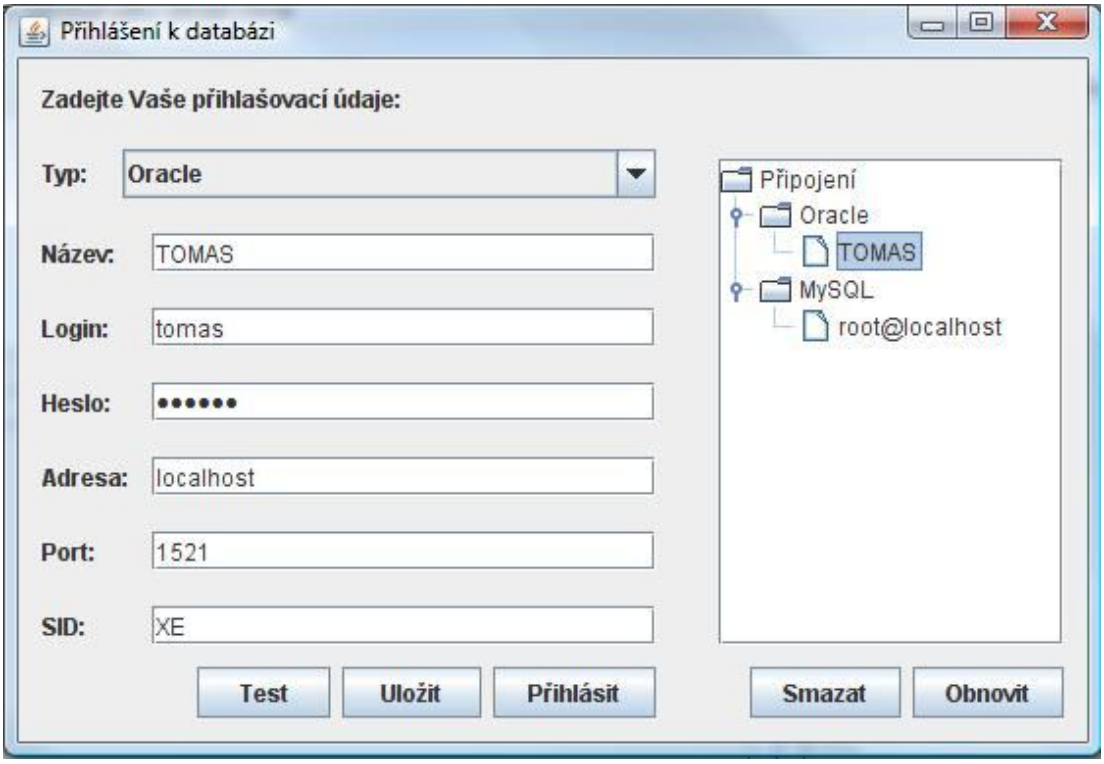
Obrázek 6 SQL Manažer - Hlavní okno aplikace. Zdroj vlastní.

Výsledky dotazů, tedy pokud SQL dotaz nějaký výsledek vrátí, jsou zobrazeny v tabulce nacházející se pod textovým polem pro zadávání SQL příkazů. V pravé části aplikace se nachází přepínatelný panel, obsahující položky Historie a Struktura databáze. Do historie se ukládají úspěšné provedené SQL příkazy s informací o datu a času jejich vykonání. Je také možné historii příkazů vyexportovat jako SQL skript. Druhá položka, struktura databáze, obsahuje strom přehledně zobrazující strukturu databáze, ke které jsme právě připojeni. Ve struktuře vidí uživatel všechny jemu přístupné tabulky, pohledy či jeho synonyma.

6.1.2 Přihlášení k databázi

Velmi důležitou součásti aplikace je formulář pro přihlášení k databázi (obrázek 7). Tento formulář najdeme v menu Databáze, položka Připojit k databázi. Formulář nám umožňuje zvolit typ databáze (Oracle nebo MySQL) a vyplnit potřebné přihlašovací údaje. Následně se může uživatel buďto rovnou přihlásit nebo může přihlašo-

vací údaje uložit pro pozdější využití. V obou případech uživatel může využít služeb tlačítka Test, jež mu oznámí, zda se lze k požadované databázi s požadovanými přihlašovacími údaji přihlásit.

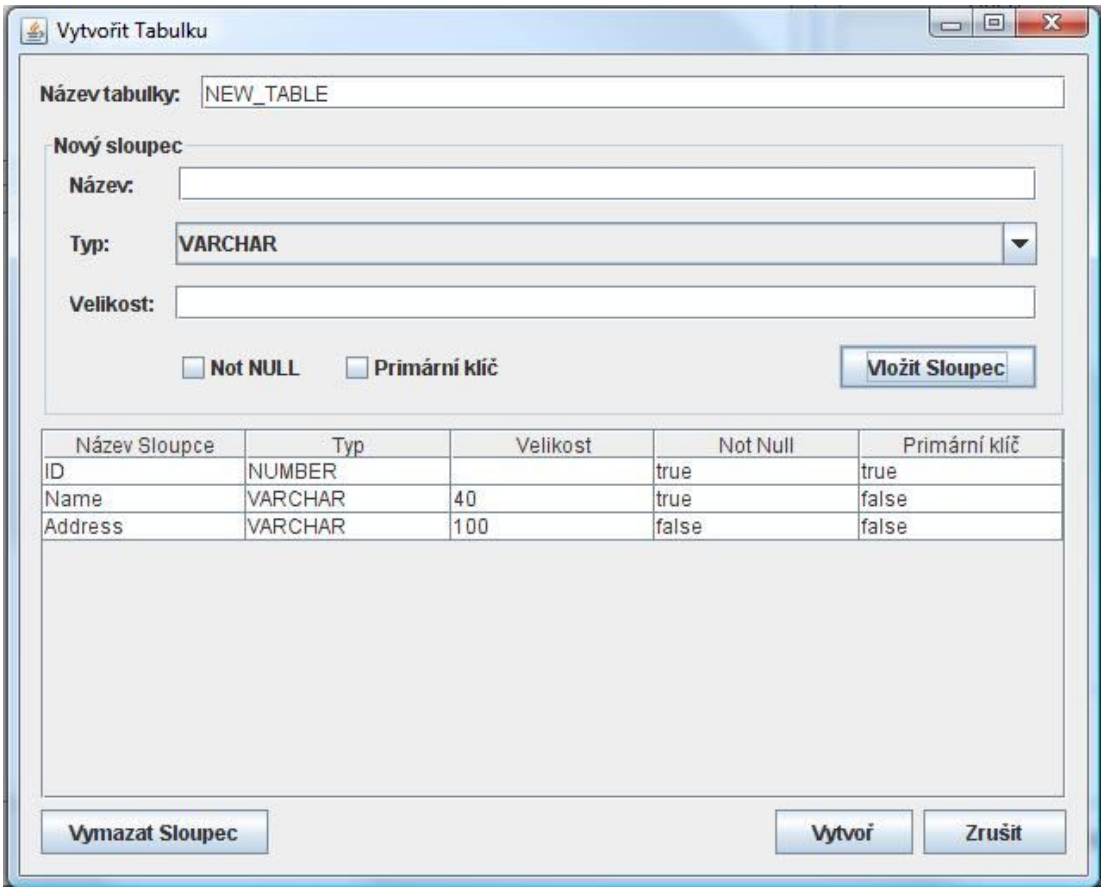


Obrázek 7 SQL Manažer - Přihlášení k databázi. Zdroj vlastní.

V pravé části přihlašovacího okna máme strom s dříve uloženými připojeními. Uživatel pak prostým kliknutím může zobrazit přihlašovací informace v přihlašovacím formuláři a poté se přihlásit. Uživatel může též staré a nepotřebné uložené přihlašovací údaje vymazat.

6.1.3 Vytvoření nové tabulky

Pokud bude chtít uživatel ve svém schématu vytvořit novou tabulku, má v SQL Manažeru v zásadě dvě možnosti. Buď může novou tabulku zapsat ve formě SQL příkazu do pole pro zadávání SQL příkazů, nebo může využít služeb průvodce pro vytvoření nové tabulky (obrázek 8). Ten se nachází v menu databáze pod položkou nová tabulka.

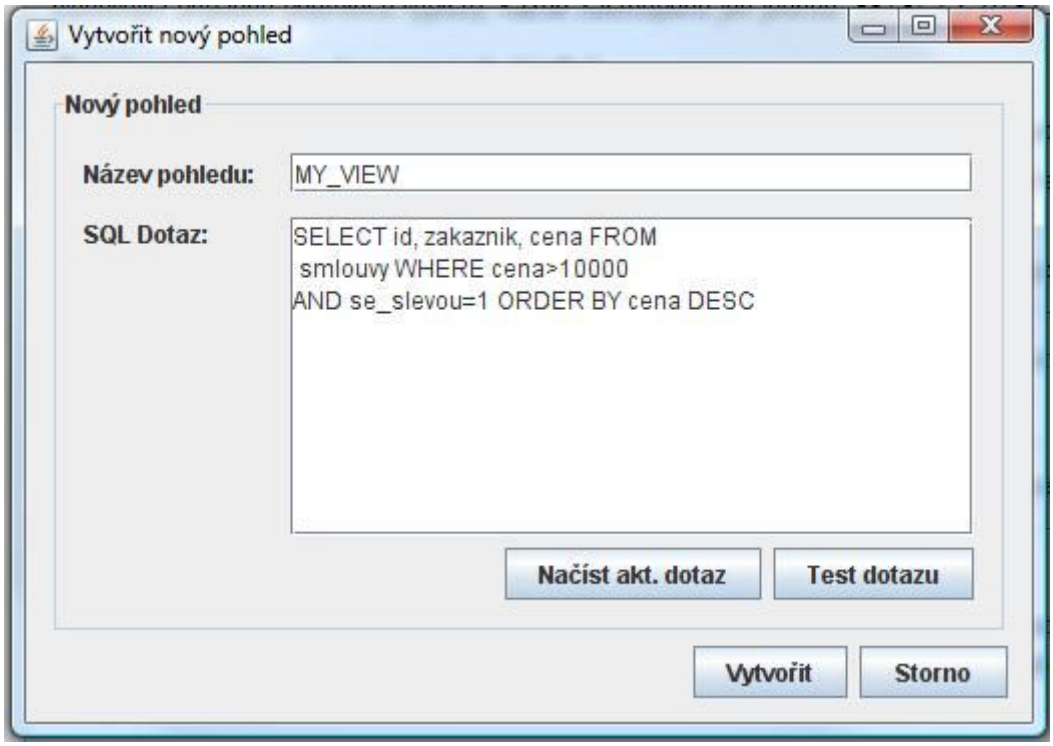


Obrázek 8 SQL Manažer - Vytvoření nové tabulky. Zdroj vlastní.

Uživatel se zde nemusí obtěžovat se psaním SQL příkazu pro vytvoření tabulky, ale stačí mu zadat jméno nové tabulky a poté vytvořit libovolný počet sloupců s požadovanými atributy a tabulku vytvořit.

6.1.4 Tvorba dalších databázových objektů

SQL Manažer neumožňuje pouze tvorbu nových tabulek. Mezi jeho další možnosti patří tvorba pohledu (obrázek 9), sekvence či synonym. Například vytvořit pohled je velmi jednoduché. Stačí z menu Databáze vybrat položku Vytvořit pohled, poté vyplnit název pohledu a zadat SQL dotaz, který bude pohled reprezentovat. Uživatel může před vytvořením pohledu otestovat správnost syntaxe SQL dotazu. SQL dotaz uživatel ani nemusí psát, stačí stisknout tlačítko načíst aktuální dotaz a do pole SQL dotazu mu bude přenesen dotaz, který je právě napsán v dotazovém poli hlavního okna.



Obrázek 9 SQL Manažer - Tvorba pohledu. Zdroj vlastní.

6.1.5 Úprava tabulky a editace dat

V menu *Upravit* se nacházejí dvě položky. První z nich, *Editovat tabulku*, umožňuje uživateli změnit parametry již vytvořené tabulky. Uživatel může změnit název tabulky, editovat názvy sloupců nebo sloupce mazat či přidávat. Druhou položkou jsou *Data tabulky,* jež uživateli nabízejí pohodlnou možnost editace dat přímo v tabulce. Uživatel může měnit jednotlivé buňky tabulky, může smazat celý řádek či nový řádek do tabulky přidat.

6.2 Implementace aplikace

Aplikaci jsem se rozhodl implementovat jako klasickou GUI desktopovou aplikaci. Aplikace byla kompletně vyvíjena v NetBeans (konkrétně verze 6.5), její grafické uživatelské rozhraní, které využívá komponent grafické knihovny Swing, bylo vytvořeno za pomoci integrovaného GUI Builderu a celá aplikace byla odladěna na Javě verze 6 (1.6.0_07). Pro připojení do databáze Oracle byl použit Oracle JDBC Driver verze 10.2.0.1.0 a pro připojení k databázi MySQL byl použit ovladač MySQL-AB

JDBC Driver (verze 5.0.8). Výhodu tohoto způsobu implementace je, že výsledný program je distribuovaný ve formě jediného JAR souboru, který bude spustitelný všude tam, kde bude k dispozici JVM. Navíc díky technologii Java Web Start (JWS), se distribuce GUI aplikací stává velmi snadnou. Díky JWS odpadá uživatelům nutnost aplikaci stahovat a instalovat, stačí pouze kliknout na odkaz a JVM (Java verze 1.4 a vyšší) se o stažení a spuštění aplikace postará sama. Za drobnou nevýhodu lze považovat, třeba například oproti čistě webovým aplikacím, které vyžadují pouze internetový prohlížeč, že je nutné zajistit na hostitelské platformě běh JVM.

6.3 Struktura aplikace

Aplikace se rozkládá v několika balíčcích. V podstatě každé nové aplikační okno (třeba jako okno přihlášení) sídlí ve svém vlastním balíčku spolu se svými pomocnými třídami. Jedinou výjimkou je třída `MainAppWindow`**,** která není součástí žádného balíčku a v souborové hierarchii stojí nejvýše ze všech tříd. Ve třídě `MainAppWindow` se nachází jádro celé aplikace, zodpovědné za vykreslení základního grafického uživatelského rozhraní a za obsluhu základních uživatelských požadavků. Pro provoz této třídy jsou nezbytné třídy `ResultTableData` a `QuerryHistoryTable`, které se nacházejí v balíčku `utils`. Obě dvě třídy v sobě implementují metody ze třídy `AbstractTableModel`**,** a tudíž jsou zodpovědné za udržování datových modelů pro instance tříd `JTable` (pro tabulku výsledků a tabulku historie) uvnitř hlavního aplikačního okna.

Třída `ResultTableData` vytváří datový model z výsledku dotazu do databáze. Dokáže tedy z výsledku dotazu do databáze (instance třídy `ResultSet`) vytvořit tabulku s výsledky. Tato třída je tedy velmi užitečná, navíc je její kód velmi univerzální a tudíž snadno znovupoužitelný. Viz kód:

```
public class ResultTableData extends AbstractTableModel{

    protected Vector collumHeadrs;
    protected Vector tableData;

    public ResultTableData(ResultSet rset) throws SQLException {
        Vector rowData;
        ResultSetMetaData rsmd = rset.getMetaData();
        int count = rsmd.getColumnCount();
        collumHeadrs = new Vector(count);
        tableData = new Vector();
        for (int i = 1; i <= count; i++) {
            collumHeadrs.addElement(rsmd.getColumnName(i));
        }
        while (rset.next()) {
            rowData = new Vector(count);
            for (int i = 1; i <= count; i++) {
                rowData.addElement(rset.getObject(i));
            }
            tableData.addElement(rowData);
        }
    }

    public int getRowCount() {
        return tableData.size();
    }

    public int getColumnCount() {
        return collumHeadrs.size();
    }

    public Object getValueAt(int rowIndex, int columnIndex) {
        Vector rowData = (Vector) tableData.elementAt(rowIndex);
        return rowData.elementAt(columnIndex);
    }

    @Override
    public String getColumnName(int column) {
        return (String) collumHeadrs.elementAt(column);
    }
}
```

6.3.1 Balíček dblogin

V tomto balíčku se nacházejí třídy zodpovědné za přihlášení do databáze. Balíček obsahuje třídu `dbLoginForm`, jenž obsahuje formulář pro přihlašovací údaje a je zodpovědná za připojení k databázi, a třídu `DBUser`, obsahující přihlašovací informace o uživateli. Tato třída implementuje rozhraní `Serailizable`, jelikož je využívána při ukládaní do objektového souboru `Users.dat`, který obsahuje uložená připojení všech uživatelů využívajících SQL Manažer.

6.3.2 Balíček info

Obsahuje velmi jednoduchou třídu `InfoFrame`, která slouží pro zobrazení okna s některými základními informacemi o aktuálním připojení.

6.3.3 Balíček alter

Balíček obsahuje rozhraní pro úpravu parametrů (název tabulky, názvy sloupců, …) tabulek. Toto rozhraní ke své funkci vyžaduje třídu `TableColumnsTableData`, obsaženou v tomto balíčku, jenž implementuje abstraktní třídu `AbstarctTableModel` a obsahuje názvy sloupců právě editované tabulky.

6.3.4 Balíček structTreeModel

V tomto balíčku nalezneme dvě třídy. A to třídu `mySQLDBTree` a třídu `OracleDBTree`. Obě třídy slouží k vytvoření datového modelu pro instanci třídy `JTree` v hlavním aplikačním okně. Obsahem stromu je struktura databáze tzn., že uživateli se struktura databáze zobrazí v podobě přehledné stromové hierarchie. Obě třídy slouží k tomu samému, s tím rozdílem, že každá je určena pro vytvoření datového modelu pro jiný druh databáze (z důvodu rozdílnosti systémových katalogů jednotlivých databázových systémů).

6.3.5 Balíček utils

O tomto balíčku jsem se zmínil již v úvodu této kapitoly, především tedy o třídě `ResultTableData`. Třída ovšem ještě obsahuje třídu `QuerryHistoryTable`, která vytváří datový model pro tabulku historie úspěšně provedených dotazů.

6.3.6 Balíček newtable

V tomto balíčku se nacházejí třídy pro vytvoření nové tabulky v databázi. Obsahuje čtyři třídy, z nichž ta nejdůležitější je `NewTableFrame`, jenž implementuje základní uživatelské rozhraní pro tvorbu nové tabulky. Tato třída se ale neobejde bez třídy `TableColumn`, jejíž instance nesou informace o jednotlivých sloupcích nové tabulky (např. název sloupce, datový typ, velikost atd.). Další důležitou třídou je třída `CreateStatement`, jenž obsahuje jedinou statickou metodu, a to `createStatement`, jenž vrací řetězec obsahující SQL příkaz pro vytvoření nové tabulky:

```
public static String createStatement(String
tableName,Vector<TableColumn> columnsVector) {
        String createString = "CREATE TABLE ";
        String nn = "NOT NULL ";
        String pk = "PRIMARY KEY ";
        String sep = ", ";

        createString += tableName + "( ";

        for(int i = 0; i < columnsVector.size();i++) {
            TableColumn tc = columnsVector.get(i);

            createString += tc.getColumnName() + " ";
            createString += tc.getType();
            if(!tc.getSize().isEmpty()) {
                createString += "(" + tc.getSize() + ") ";
            } else {
                createString += " ";
            }
            if(tc.isNotNull()) {
                createString += nn + " ";
            }
            if(tc.isPrimaryKey()) {
                createString += pk + " ";
            }


            if(i != (columnsVector.size()-1)) {
                createString += sep;
            }

        }

        createString += ")";

        return createString;
    }
```

Jak vidíte, metoda `createStatement` přímá dva argumenty. Prvním je název nové tabulky, druhým je vektor instancí třídy `TableColumn`. Vektor tedy obsahuje informace o všech zamýšlených sloupcích nové tabulky. Tento balíček ještě navíc obsahuje třídu `NewTableModel`, obsahující datový model pro tabulku právě přidávaných sloupců.

6.3.7 Balíček pass

Obsahuje jednoduché rozhraní pro změnu přístupového hesla do databáze.

6.3.8 Balíček sequence

Opět velmi jednoduché rozhraní pro tvorbu databázových sekvencí.

6.3.9 Balíček synonyms

Obsahuje velmi jednoduché rozhraní pro tvorbu databázových synonym.

6.3.10 Balíček views

Obsahuje třídu `CreateViewFrame`, jenž poskytuje základní rozhraní pro tvorbu databázových pohledů.

6.3.11 Balíček edittable

Tento balíček obsahuje třídy pro editaci dat v jednotlivých databázových tabulkách. V rámci editace je možné editovat stávající řádky, mazat je nebo přidávat nové.

7 Závěr

Databáze dnes tvoří nedílnou součást internetových aplikací. Databáze využívá, ať už vědomě či nevědomě, denně každý z nás. Dnešní internetové aplikace by se bez databází a jazyka SQL asi tak snadno nerozšířily a těžko by se tak rychle rozvíjely. Se stále rostoucím zájmem o databázové systémy logicky roste i zájem o nástroje pro správu databází, především pak o ty s přehledným grafickým rozhraním.

Účelem této práce bylo vytvořit aplikaci demonstrující možnosti jazyka Java (konkrétně J2SE), jako nástroje pro tvorbu databázově orientovaných aplikací. Při návrhu aplikace jsem byl postaven před dva zásadní úkoly. Prvním bylo pochopení a implementace aplikačního rozhraní JDBC a pochopení způsobu jeho spolupráce s databází. Druhým úkolem, který mi při zpětném hodnocení projektu připadá implementačně náročnější, bylo navrhnout kvalitní a uživatelsky přívětivé prostředí. Co se uživatelského rozhraní týče, za nejnáročnější považuji implementaci takových komponent uživatelského rozhraní, jako jsou tabulka (JTable) a strom (JTree).

Aplikace byla vyvíjena výhradně pro účely této práce, proto ji jenom těžko můžeme srovnávat s podobnými nástroji, jako je třeba Oracle SQL Developer. Aplikace má spíše demonstrativní charakter, ale i přes to je dobře použitelná pro základní databázové operace (vytvoření nové tabulky, editace existující tabulky, tvorba pohledu, editace dat v tabulce).

V aplikaci jsem se snažil implementovat veškeré funkce ze zadání práce, což se v případě připojení k databázi Oracle bezezbytku podařilo. V případě připojení k databázi MySQL se mi z časových, a zároveň i implementačních, důvodů nepodařilo implementovat funkčnost ve stejném rozsahu, jako je tomu v případě připojení k databázi Oracle. I přesto věřím, že aplikace, a především pak její zdrojové kódy, mohou posloužit všem zájemcům o pochopení této problematiky, jako vhodný zdroj inspirace a příkladů.

SEZNAM POUŽITÝCH ZDROJŮ A LITERATURY

[1] GROFF, James R., WEINBERG, Paul N. *SQL Kompletní průvodce.* Brno: CP Books, 2005. 936 s. ISBN 80-251-0369-2

[2] *Oracle Database* [online]. Wikipedia, the free encyclopedia, [2001], this page was last modified on 29 April 2009, at 03:31 (UTC) [cit. 2009-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Oracle_Database>.

[3] *MySQL* [online]. Wikipedia, the free encyclopedia, [2001], this page was last modified on 29 April 2009, at 05:11 (UTC) [cit. 2009-04-29]. Dostupný z WWW: < http://en.wikipedia.org/wiki/MySQL>.

[4] *SQL* [online]. Wikipedia, 2007, stránka byla naposledy editována 16. 1. 2009 v 19:09 [cit. 2009-04-21]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/SQL>.

[5] KEOGH, James. *Java: bez předchozích znalostí.* Petr Baláš; Ivo Magera; Ivo Fořt; Martin Sodomka. 1. vyd. Brno: CP Books a.s., 2005. 275 s. ISBN 80-251-0839-2.

[6] SPELL, Brett. *Java*: Programujeme *profesionálně*. 1. vyd. Praha: Computer Press, 2002. 1022 s. ISBN 80-7226-667-5.

[7] FILTH, Oli. *Java (programming language)* [online]. Wikipedia, the free encyclopedia, [2001], this page was last modified on 28 April 2009, at 18:26 (UTC) [cit. 2009-04-29]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Java_(programming_language)>.

[8] CHAPMAN, Stephen J. *Začínáme programovat v jazyce Java.* [s.l.] : Computer Press, 2003. 318 s. ISBN 80-7226-472-9.

[9] *The Java Tutorials: How to Use Tables* [online]. Sun Microsystems, c1995-2008 , Last Updated 2/14/2008 [cit. 2009-04-20]. Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/uiswing/components/table.html> .

[10] *The Java Tutorials:* How to Use Trees [online]. Sun Microsystems, c1995-2008 , Last Updated 2/14/2008 [cit. 2009-04-20]. Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/uiswing/components/tree.html>.

[11] ŠEDA, Jan. *Úvod do JDBC* [online]. Interval.cz, [2003], 4. 3. 2003 [cit. 2009-04-25]. Dostupný z WWW: <http://interval.cz/clanky/uvod-do-jdbc/>. ISSN 1212-8651

[12] *Nástroj pro dotazování databází – aplikace Aqua Data Studio* [online]. aquafold.com, c2001-2009, 1. 4. 2009 [cit. 2009-04-30]. Dostupný z WWW: <http://www.aquafold.com/cs/index.html>.