VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

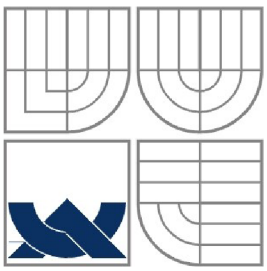
VERZOVÁNÍ DATABÁZE PŘI VÝVOJI APLIKACÍ V ECLIPSE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

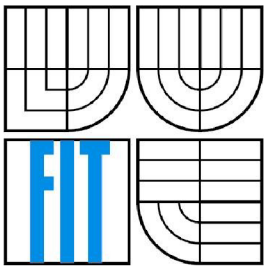
AUTOR PRÁCE
AUTHOR

VÍT PALARCZYK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VERZOVÁNÍ DATABÁZE PŘI VÝVOJI APLIKACÍ V ECLIPSE

DATABASE VERSION MANAGEMENT FOR APPLICATION DEVELOPMENT IN ECLIPSE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÍT PALARCZYK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zabývá verzováním databáze při vývoji aplikací v Eclipse. V teoretické části jsou obsaženy informace o samotném prostředí aplikace Eclipse a také o rozšiřujících modulech (pluginech), které jsou nedílnou součástí Eclipse. Dále je zde popis systémů pro správu verzí s podrobnějším zaměřením na systém Subversion a popis aplikačního rozhraní JDBC, které se používá pro připojení k databázi u Java aplikací. Další část práce se zabývá návrhem nástroje pro verzování databáze, který je implementován jako plugin pro Eclipse. V této části je popis implementace i instalace a použití tohoto nástroje. V závěru práce jsou popsány provedené testy a jejich výsledky.

Abstract

This thesis deals with database version control for application development in Eclipse. In the theoretical part, it provides information about the Eclipse environment and its extension modules (plugins) that are an inseparable part of Eclipse. There is also a description of version control systems with a detailed focus on the Subversion system and a description of the JDBC application interface that is used for connecting the database in Java applications. The next part deals with the design of a tool for database version control, which is implemented as a plugin for Eclipse. In this section, there is a description of the implementation and a description of the installation and use of this tool. In the final part, there is a description of the performed tests and their results.

Klíčová slova

Subversion, SVN, JDBC, Java, Eclipse, plugin, databáze, systém řízení báze dat, SŘBD, PostgreSQL, MySQL, verzování databáze, export databáze, import databáze

Keywords

Subversion, SVN, JDBC, Java, Eclipse, plugin, database, database management system, DBMS, PostgreSQL, MySQL, database versioning, export of database, import of database

Citace

Vít Palarczyk: Verzování databáze při vývoji aplikací v Eclipse, bakalářská práce, Brno, FIT VUT v Brně, 2012

Verzování databáze při vývoji aplikací v Eclipse

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Vít Palarczyk
16. května 2012

Poděkování

Děkuji vedoucímu své bakalářské práce panu Ing. Radku Burgetovi, Ph.D. za odbornou pomoc a cenné rady při konzultacích.

Obsah

1 Úvod....2
2 Eclipse....3
2.1 Eclipse plugin....4
2.2 Vytvoření pluginu....4
2.3 PHP v Eclipse....5
3 Systém pro správu verzí....6
3.1 Subversion....8
3.2 Správa verzí v Eclipse....8
4 Připojení k databázi....11
4.1 JDBC....11
4.2 Nástroje pro správu databází v Eclipse....12
5 Analýza a návrh verzování....13
5.1 Databázové objekty....14
6 Implementace....16
6.1 Export databáze....17
6.2 Import databáze....19
6.3 Popis architektury pluginu....20
7 Instalace a použití....21
8 Testování....22
8.1 Souběžný vývoj....23
8.2 Transakční zpracování....25
9 Další možná rozšíření....26
9.1 Verzování aktualizačních skriptů....26
10 Závěr....28

1 Úvod

Verzování, neboli uchovávání historie změn, zdrojového kódu je při vývoji aplikací běžné. Protože se zdrojový kódy píší do obyčejných textových souborů, které lze snadno verzovat, existuje k tomuto účelu spousta nástrojů. Verzování databáze je však stejně důležité jako verzování zdrojového kódu. Bohužel to již není tak jednoduché a nástrojů, které by toho byly schopny, je opravdu velmi málo.

Tato práce se tedy zabývá vytvořením nástroje, který dokáže verzovat strukturu a obsah relační databáze. Zároveň je tento nástroj navržen jako rozšiřující modul (plugin) pro Eclipse a je tedy napsán čistě v jazyce Java. Jelikož nástroje pro správu verzí jsou již součástí aplikace Eclipse, je vyvíjený nástroj zaměřen především na vytvoření souborů reprezentujících strukturu a obsah databáze. K samotnému verzování tak lez použít i jinou samostatnou aplikaci, vice v kapitole 5.

Kapitola 2 se zabývá popisem aplikace Eclipse, rozšiřujícími moduly a jejich vytvářením. Vytvořený nástroj je použitelný, nejen při vývojí Java aplikací, ale také PHP aplikací. Stručný popis vývoje PHP aplikací v prostředí Eclipse je uveden v kapitole 2.3.

V zadání této práce není specifikován systém řízení báze dat (SŘBD), se kterým by měl vytvořený nástroj pracovat. Přestože vedoucí mé práce kladl důraz především na systém MySQL, rozhodl jsem se pro obecný přístup k databázím, aby bylo možné použít jakýkoli databázový systém. K tomuto účelu jsem využil aplikační rozhraní JDBC, popsané v kapitole 4.1. Pro konkrétní testování aplikace jsem vybral již zmiňovaný MySQL a také PostgreSQL, což je jeden z nejpoužívanějších systému při vývoji aplikací v jazyce Java.

Kapitola 3 popisuje systém pro správu verzí obecně a kapitola 3.1 je zaměřena konkrétně na systém Subversion. Správa verzí v Eclipse a existující pluginy pro tento účel jsou popsány v kapitole 3.2. Kapitola 5 se zabývá možnostmi verzování a popisem řešení, které spočívá v importu a exportu databáze. Podrobný popis, jak vytvořit jednotlivé databázové objekty, je součástí kapitoly 5.1. Architektura implementované aplikace a konkrétní popis implementace importu a exportu databáze je v kapitole 6. Jak daný plugin nainstalovat, jak se pomocí něj správně připojit k databázi a jak s jeho pomocí verzovat databázi, je součástí kapitoly 7.

V kapitole 8 jsou popsány provedené testy a jejich výsledky. Test souběžného vývoje je v jednotlivých krocích, detailně popsán v kapitole 8.1. V kapitole 9 jsou stručně popsána další možná rozšíření a návrhy na změnu samotného verzování.

2 Eclipse

Eclipse je rozšiřitelná open source platforma pro vývoj aplikací v jazyce Java. V listopadu 2001 byla vydána první verze (verze 1.0) aplikace [1]. Nejnovější verze, která byla vydána v roce 2011, má označení Indigo. Eclipse se skládá z jádra SDK (Software Development Kit) a dvou základních částí. JDT (Java Development Tools) pro psaní a ladění Java aplikací a PDE (Plug-in Development Environment) pro rozšiřování funkčnosti vývojového prostředí. Dále obsahuje tzv. „plugin loader“, přes který je připojeno několik stovek až tisíc malých Java aplikací. Každá tato aplikace (neboli plugin) rozšiřuje funkcionalitu samotné aplikace Eclipse. Pluginy se načítají dynamicky za běhu aplikace až při jejich prvním použití a mohou rozšířit buď samotné jádro SDK, nebo nějaký další plugin. Např. plugin pro textový editor můžeme dále rozšířit a vytvořit tak XML editor [2].

Přestože se Eclipse využívá převážně pro vývoj aplikací v jazyce Java, byly vyvinuty i nástroje, které umožňují vývoj např. v jazyce PHP, C++, Cobol nebo C# [1]. Při vývoji aplikace je možné mít v jednom pracovním prostředí (neboli workspace) více projektů, které lze nezávisle na sobě spouštět a ladit. Samozřejmě je možné mít i více oddělených pracovních prostředí a v každém mít rozdílné nastavení editorů, rozmístění jednotlivých oken, nebo např. klávesových zkratek. Pro přepnutí do jiného pracovního prostředí je však potřeba Eclipse restartovat a zadat cestu ke složce, která dané prostředí obsahuje.

Standardní editor pro psaní zdrojového kódu obsahuje zvýraznění a obarvení syntaxe, což čtení kódu velice usnadňuje. Také obsahuje automatické formátování, které kód správně odsadí, zalomí příliš dlouhé řádky a upraví tak soubor do přehledné a čitelné podoby. Navíc je možné vše nastavit dle potřeb a zvyklostí uživatele.

Při psaní zdrojových kódů samozřejmě funguje i historie, a to jak obsahu všech otevřených souborů, tak celých projektů i nastavení prostředí. Je tedy např. možné obnovit smazaný soubor nebo znovu otevřít právě zavřenou záložku.

Při vývoji je k dispozici mnoho dalších užitečných funkcí. Za zmínku určitě stojí automatické napovídání a doplňování zdrojového kódu, možnost automaticky vytvořit standardní metody pro přístup k atributům dané třídy (anglicky *Getters* a *Setters),* nebo např. automatické přejmenování a následné nahrazení všech výskytů dané třídy nebo metody (anglicky *Refactoring).*

Grafické uživatelské rozhraní Eclipse je postaveno na knihovnách SWT (Standard Widget Toolkit) a JFace [1]. Hlavním rysem uživatelského rozhraní je možnost měnit perspektivy, které obsahují různé pohledy a editory. Lze tak jednoduše přepínat např. s Java perspektivy do Debug perspektivy, která je lépe uzpůsobena k ladění programů. SWT i JFace obsahují základní ovládací prvky (tabulky, dialogy, popisky, tlačítka atd.) pro snadné vytvoření grafického uživatelského rozhraní, a to jak u Java aplikací, tak u pluginů pro Eclipse.

2.1 Eclipse plugin

Eclipse má rozšiřitelnou architekturu, která dovoluje komukoli přidat nový zásuvný modul (plugin) a rozšířit tak funkčnost aplikace. Díky těmto pluginům je tak možné přidat do prostředí jakoukoli novou komponentu. Plugin je vlastně nejmenší rozšiřitelná jednotka. Každý plugin dokáže s jinými pluginy spolupracovat. Možnost rozšířit plugin o další funkčnost je dána tzv. „extension point“ (rozšiřitelným bodem). Tento bod určuje obecné rozhraní pro další moduly, které můžou přidat funkčnost vytvořením tzv. „extension“ (rozšíření). Každý extension point i extension jsou specifikovány v souboru *plugin.xml* [2].

Ve složce, kde je umístěn samotný program Eclipse, je podsložka *plugins*, ve které jsou všechny dostupné pluginy ve formátu JAR archivu. Každý tento soubor má plně kvalifikované jméno pluginu následované podtržítkem a číslem verze. Jeho obsah je tvořen několika složkami a soubory, které tvoří základ pluginu. Nejdůležitější soubor je tzv. *manifest*, který obsahuje především definici rozšíření a také akcí, které se spustí při použití pluginu. Tento soubor najdeme v kořenové složce pod názvem *plugin.xml*. K tomuto souboru jsou k dispozici ještě dva dodatečné soubory. Jméno a verze daného pluginu je v souboru *MANIFEST.MF* a informace potřebné pro sestavení jsou v souboru *build.properties*.

Aplikace při startu prochází složku *plugins* a zjišťuje, které pluginy jsou definovány. Pokud pro jeden plugin najde více verzí, je použita ta nejnovější. Samozřejmě se pluginy nenačítají hned při startu aplikace, ale až při jejich prvním použití, což start Eclipse dosti urychluje.

2.2 Vytvoření pluginu

Plugin lze vytvořit ručně, ale data v manifestu mohou být někdy velice rozsáhlá a jejich ruční správa a dodržování jmenných konvencí může být velice náročné. Proto je k dispozici vývojové prostředí PDE (Plugin Development Environment), které usnadňuje vytváření, správu a publikování pluginů. Pro vytvoření pluginu lze použít průvodce, který v několika krocích vygeneruje ze zadaných údajů základní zdrojový kód a nemusíme jej tak psát ručně.

Při vytváření pluginu pomocí průvodce stačí v podstatě zadat pouze název projektu. Doporučuje se zadat tento název jako plně kvalifikované jméno včetně všech balíčků, kterých je součástí. Je to hlavně z toho důvodu, aby bylo jméno unikátní a nedocházelo tak ke kolizím s jinými pluginy. V průvodci také máme možnost automaticky vytvořit Java třídu *Activator*, která je důležitá pokud požadujeme provedení nějaké akce při spuštění nebo ukončení pluginu [2]. Nemusí to ale nutně znamenat, že se tyto akce spustí při startu Eclipse, protože se plugin spouští až ve chvíli potřeby.

ID daného pluginu je automaticky vyplněno podle zadaného jména projektu. Stejně tak i verze a název samotného pluginu se doplní automaticky. Samozřejmě je možné údaje změnit, ale identifikátor (ID) by měl vždy obsahovat unikátní plně kvalifikované jméno. Pro rychlé vytvoření běžných pluginů jsou v Eclipse přichystány šablony, které mají základní zdrojový kód již předpřipravený [1].

Po vytvoření pluginu je nutné jej exportovat do formátu, který slouží pro jednoduchou instalaci pluginu na jiném počítači. Exportovaný soubor je ve formátu JAR archivu, který lze jednoduše nakopírovat do složky s ostatními Eclipse pluginy. Součástí tohoto archivu jsou zkompilované Java soubory s příponou *class*. V dialogu pro export je však možné nastavit, že chceme exportovat i zdrojový kód. Můžeme tak mít dva oddělené JAR archivy, jeden se zkompilovanými soubory (**.class*) a druhý s originálními soubory (**.java*) obsahující zdrojový kód. Druhou možností je pak provést export zdrojového kódu společně se zkompilovanými soubory do jediného archivu. V tom případě jsou soubory se zdrojovým kódem umístěny ve složce *src*.

Pokud bychom pro ladění a testování pluginu měli vždy provést export a nakopírovat archiv do složky s ostatními pluginy, bylo by navíc nutné Eclipse po každém exportu restartovat. Toto řešení je ale velice zdlouhavé a neefektivní, proto vývojové prostředí PDE nabízí možnost vytvoření dočasné instance celé aplikace Eclipse, kde lze daný plugin vyzkoušet a ladit.

Pro správu základního nastavení pluginu slouží *manifest editor*, který je rozdělen do několika záložek:

- **Overview** - obsahuje základní informace o pluginu jako je identifikátor, název, verze atd. Pro testování je možné odsud spustit samostatnou Eclipse aplikaci, která již bude obsahovat daný plugin. Samozřejmě je zde i možnost spustit aplikaci v Debug módu, kde lze vyvíjený plugin ladit a upravovat zdrojový kód za běhu.
- **Dependencies** - obsahuje seznam všech pluginů, na kterých je daný plugin závislý a které jsou potřeba pro jeho sestavení.
- **Runtime** - na této záložce můžeme přidat balíčky (packages), které budou viditelné i mimo tento plugin, tedy pro rozšiřující pluginy. Dále zde můžeme přidat důležité knihovny a složky do tzv. „classpath“. Classpath musí obsahovat všechny externí JAR soubory, které jsou v pluginu použity.
- **Extensions** - pomáhá rychle a jednoduše vytvořit rozšíření, které se připojí na rozšiřitelný bod jiného pluginu.
- **Extension Points** - zde můžeme přidat vlastní rozšiřitelné body, ke kterým se budou moci jiné pluginy připojit.
- **Build** - záložka pro určení složek a souborů potřebných pro zkompilování a sestavení aplikace. Pokud jsou součástí naší aplikace (pluginu) např. obrázky, je nutné tyto soubory označit v sekci *Binary Build*.

Poslední tři záložky (MANIFEST.MF, plugin.xml a build.properties) obsahují pouze textovou reprezentaci předchozích záložek.

2.3 PHP v Eclipse

Pomocí aplikace Eclipse lze vyvíjet programy i v jiných programovací jazycích, než je Java. A právě pro vývoj v jazyce PHP je k dispozici nástroj PDT (PHP Development Tools), který je k dispozici na oficiálních stránkách Eclipse (*www.eclipse.org/pdt/*). Lze jej tedy do Eclipse nainstalovat samostatně, nebo můžeme stáhnout Eclipse balíček, který již v sobě obsahuje PDT

i debugger potřebný k ladění. Tento balíček je dostupný na webových stránkách společnosti Zend, která se zabývá vývojem aplikací v PHP (*www.zend.com/en/community/pdt/*).

Pro správný chod webových aplikací je nutné na systém doinstalovat samotné PHP, webový server a také databázový server MySQL. Nejjednodušší cestou je nainstalovat aplikaci WampServer (*www.wampserver.com*), což je webový server, který obsahuje všechny potřebné nástroje. Standardně se aplikace nainstaluje do složky *wamp* na disku *C*. Pro usnadnění práce je vhodné umístit pracovní prostředí (neboli workspace) Eclipse právě do této složky, konkrétně do složky *c:\wamp\www\workspace*. Díky tomu lze potom jednoduše spustit daný soubor např. pod adresou *http://localhost/workspace/projekt/index.php*.

Po nainstalování aplikace WampServer je nutné upravit nastavení Eclipse pro spouštění PHP skriptů. V hlavním menu zvolíme *Windows- > Preferences -> PHP*, kde nastavíme nejprve P*HP Executables*. Pomocí tlačítka *Add* přidáme novou definici, která může vypadat následovně:

Name: xdebug
Executable Path: C:\wamp\bin\php\php5.3.8\php-cgi.exe
PHP ini file: C:\wamp\bin\apache\Apache2.2.21\bin\php.ini
SAPI Type: CGI
PHP debugger: XDebug

Poté přejdeme na *Windows- > Preferences -> PHP -> Debug* a nastavíme *Default Settings*:

PHP Debugger: XDebug
Server: Default PHP Web Server
PHP Executable: xdebug

Pro vytvoření nového projektu zvolíme v hlavním menu *File -> New -> PHP project* a zadáme jméno projektu. Pokud chceme importovat do projektu již hotové soubory, stačí je zabalit do archivu a zvolit *File -> Import -> PHP -> Phar File*. Poté už lze jednoduše spouštět jednotlivé soubory pravým kliknutím myši a spuštěním *Run As* nebo *Debug As -> PHP script* nebo *PHP Web Page* v případě webové aplikace. Pokud zvolíme *PHP Web Page* objeví se dialogové okno pro zadání URL adresy. Adresa je už předvyplněná, *http://localhost/* je cesta do složky *c:\wamp\www*, poté musí následovat jméno pracovního prostředí tedy */workspace/* a nakonec název projektu a souboru. Celá adresa by měla vypadat např. takto *http://localhost/workspace/projekt/index.php*.

3 Systém pro správu verzí

Systém pro správu verzí (verzovací systém) je systém, který dokáže zaznamenat změny v souborech a adresářích a postupně tyto změny ukládat v jednotlivých verzích (revizích). Také dovoluje tyto změny prohlížet a navrátit soubor do stavu v jakém se nacházel v určité revizi.

Jádro verzovacího systému tvoří repozitář, což je místo kde se ukládají veškeré změny a další informace potřebné pro práci se spravovanými soubory. Ve většině případů jsou data v repozitáři uložena ve formě stromu souborového systému, tedy jako hierarchie souborů a adresářů. K tomuto

repozitáři, který je většinou na serveru, se jako klienti můžeme připojit a číst nebo zapisovat data. Zápisem dáváme data přístupná dalším uživatelům, čtením naopak přijímáme data od ostatních.

Dalo by se tedy říct, že se repozitář chová jako obyčejný server, ale odlišuje se právě tím, že si uchovává všechny verze souborů. Klient běžně vidí pouze poslední verzi stromu z repozitáře, ale je schopný požádat repozitář o předchozí stav (revizi). Stejně tak si může zažádat i o informace o dané revizi, tedy datum a čas změny nebo jméno osoby, která změnu provedla [3].

Systém pro správu verzí může být centralizovaný nebo distribuovaný. U centralizovaného systému je repozitář pouze jeden a uživatelé mají své vlastní lokální kopie obsahu repozitáře v pracovních adresářích. Distribuované systémy nemají pouze jeden repozitář společný pro všechny, ale každý uživatel má u sebe kopii celého repozitáře. Mohou tak verzovat soubory lokálně bez připojení na server [4].

Způsob jakým jsou soubory uloženy v repozitáři je pro většinu běžných aplikací nesrozumitelný. Nejsou zde totiž uloženy celé soubory, ale pouze jejich změny a často v binární podobě. Obyčejné programy neumí s těmito verzemi souboru pracovat a potřebují každý soubor jako celek. K tomuto účelu slouží pracovní adresář (working copy).

Pracovní adresář obsahuje lokální kopii určité verze dat, kterou může uživatel měnit aniž by ovlivnil data v repozitáři. Všechny programy ho vidí jako obyčejný adresář obsahující soubory, se kterými umí běžně pracovat. Komunikaci s repozitářem a správu verzí těchto souborů obstarává pouze softwarový klient verzovacího systému. Pro nahrání nových změn z pracovního adresáře do repozitáře slouží akce *commit.* S uložením nového stavu souboru se uloží i datum a čas změny a informace o autorovi dané změny. Spolu s tím se v repozitáři vytvoří i nová revize [4].

Při běžném používání verzovacího systému může dojít k situaci, kdy chce více uživatelů editovat jeden soubor zároveň. Některé systémy k řešení této situace využívají tzv. zámky. Pokud chce uživatel měnit soubor, uzamkne ho a ostatní k němu nemají přístup. Až ve chvíli, kdy uživatel nahraje své změny do repozitáře, je soubor odemčen a zpřístupněn tak ostatním.

Používání zámků není vždy to nejvhodnější řešení, protože často dochází k vzájemné blokaci uživatelů. Proto většina verzovacích systémů používá tzv. *„copy-modify-merge“* řešení, které umožňuje více lidem pracovat na jednom souboru současně a jejich změny automaticky sloučit při nahrání do repozitáře.

Ne vždy je však systém schopen sloučit dvě různé změny jednoho souboru, v takovém případě nastává konflikt a uživatel musí tento konflikt vyřešit tím, že sloučení (*merge*) provede ručně. Ke konfliktu může dojít např. pokud dva uživatelé změní jeden soubor na stejném řádku nebo řádcích. Samozřejmě toto *„copy-modify-merge“* řešení je použitelné pouze u souborů, které lze jednoduše sloučit, jako jsou textové soubory. U obrázků a jiných binárních souborů se opět musíme uchýlit k řešení s použitím zámků [3].

Mezi systémy pro správu verzí patří např. CVS (Concurrent Versions System) nebo Subversion, což jsou centralizované systémy s jedním centrálním repozitářem. Za zmínku určitě stojí i distribuované systémy jako Git nebo Mercurial, které mohou být pro začátečníky o něco obtížnější, za to mají mnoho užitečných funkcí, které bychom u CVS hledali marně.

3.1 Subversion

Subversion (SVN) je centralizovaný open source systém pro správu verzí. Je to následník systému CVS, který se používal mnoho let, ale obsahoval spoustu nedostatků. Hlavním cílem Subversion tedy bylo odstranit tyto nedostatky a zároveň zachovat stejný přístup k systému, na který jsou uživatelé zvyklí.

Samotné ukládání revizí se oproti CVS hodně změnilo. Čísla revizí se u CVS vztahovala k jednotlivým souborům. Každá verze souboru měla unikátní číslo revize. U Subversion je repozitář spíše souborový systém, kde každá nová revize představuje nový strom tohoto souborové systému. Každý takovýto strom má pak jedno číslo revize. Pokud např. mluvíme o revizi číslo 32, není to verze jednoho souboru, ale celého stromu, který obsahuje všechny soubory nahrané do systému do této doby. Když tedy máme od revize 10 nový soubor, který byl naposledy změněn v revizi 18, je tento soubor i součástí revize 32.

Do Subversion byla přidána i schopnost verzovat stromovou strukturu adresářů, čehož systém CVS nebyl schopný, protože verzování bylo založeno pouze na obsahu daných souborů. V Subversion jsou tedy adresáře, stejně jako soubory, součástí stromu souborového systému dané revize. Pro správu verzí je použit binární algoritmus pro zjišťování změn. Ať už se tedy jedná o textové nebo binární soubory, vždy je použit tento algoritmus a všechny soubory mohou být ukládány v repozitáři stejným způsobem. To znamená, že je vždy uložena pouze změna souboru oproti předchozí revizi a ne celý soubor. To je další změna oproti CVS, kde byly binární soubory (např. obrázky) vždy uloženy do repozitáře jako celek a to při každé změně (revizi) [3].

Pokud uživatel změní soubor, který pár minut před ním změnil někdo jiný a dojde tak ke konfliktu, systém nedovolí uživateli tento soubor nahrát do systému. Skutečnost, že je některý soubor v konfliktním stavu, si systém poznamená a zamezí tak uživateli, aby tento konfliktní soubor nahrál do repozitáře. Toto zabezpečení u CVS chybělo a označení konfliktního místa bylo lehce přehlédnutelné. Subversion v takovémto souboru zaznačí lokální změny i změny z repozitáře a nechá řešení na uživateli. Poté co uživatel konflikt manuálně odstraní a označí soubor jako vyřešený, povolí systém nahrání tohoto souboru do repozitáře.

Jednou z dalších výhod Subversion je také ukládání informací o souboru lokálně u klienta v adresáři *.svn*. Díky tomu není potřeba připojovat se k serveru, pokud chceme např. zjistit, které soubory jsme změnili, přidali, popř. smazali. Stejně tak je možné zobrazit řádky, které se změnily v jednom určitém souboru, nebo odstranit všechny lokální změny a navrátit tak soubor do původního stavu (*revert*). Všechny tyto operace lze provést, aniž bychom zatěžovali síťové spojení a server samotný [3].

3.2 Správa verzí v Eclipse

Aplikace Eclipse má již v základu nástroj na správu verzí pomocí systému CVS. Pro použití SVN je potřeba doinstalovat některý z pluginů. Dva nejpoužívanější pluginy jsou *Subversive* a *Subclipse*. Rozdíl mezi nimi je minimální. U obou jsou příkazy pro správu Subversion umístěny

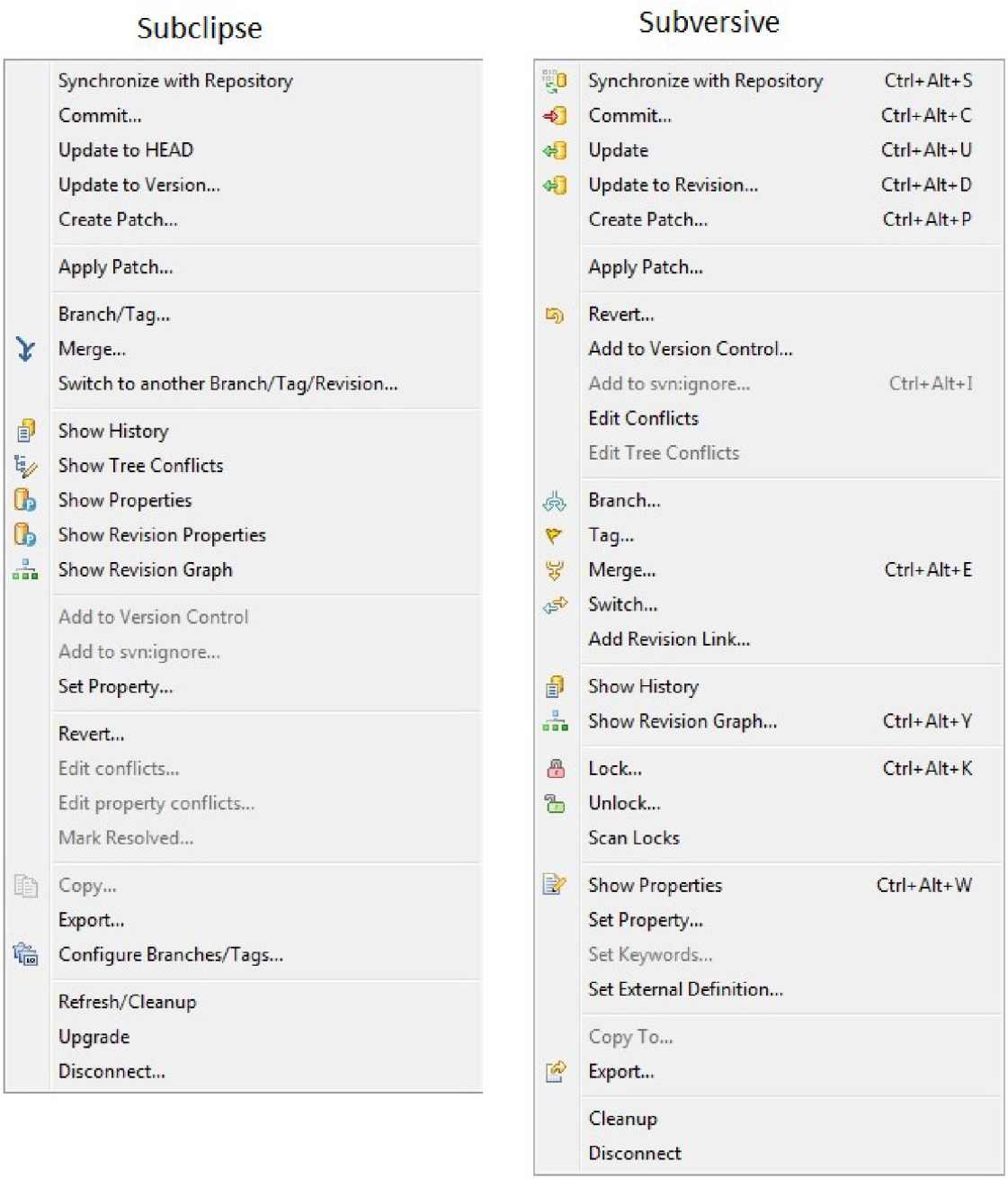
v kontextovém menu pod položkou *Team.* Základní akce jako jsou *Update, Commit, Show History* apod. jsou téměř identické v jednom i druhém pluginu. Liší se snad jen pořadí položek v menu a vzhled některých dialogů, jako např. commit dialog pro nahrání změn do repozitáře.

Po instalaci pluginu se objeví nová perspektiva „SVN Repository Exploring“, která je opět pro oba pluginy velice podobná. Tato perspektiva umožňuje prohlížet a spravovat daný repozitář, zobrazovat např. historii nebo graf revizí.

Pokud již máme na serveru vytvořený repozitář, připojíme se k němu jednoduše pomocí kontextového menu *Team->Share Project*, které vyvoláme kliknutím pravým tlačítkem na daný projekt. Zde zvolíme možnost SVN a zadáme URL adresu daného repozitáře. Pokud je repozitář umístěn na lokálním disku, musí adresa začínat klíčovým slovem file, následovaným dvojtečkou a třemi lomítky (*file:///*). Poté už stačí připsat absolutní cestu k adresáři obsahující repozitář.

Pokud náš projekt ještě v repozitáři není umístěn, provede se první inicializační nahrání (*commit*) projektu do repozitáře. V případě, že už je projekt v repozitáři obsažen, provede se pouze aktualizace *(checkout)* nejnovější verze z repozitáře do pracovního adresáře.

Po úspěšném připojení k repozitáři se v kontextovém menu *Team* zobrazí další položky pro práci v Subversion. Rozdíl kontextového menu pro *Subclipse* a *Subversive* je na obrázku 1.



Obrázek 1: Porovnání kontextového menu pro Subclipse a Subversive

Je vidět, že základní příkazy pro práci s repozitářem jsou stejné, Subversive má navíc ještě klávesové zkratky, což práci dosti urychluje.

Instalace probíhá stejně jako u běžných pluginů. V hlavním menu vybereme *Help->Install New Software* a zadáme příslušnou URL adresu, kterou lze získat na oficiálních stránkách (http://subclipse.tigris.org/ a http://www.eclipse.org/subversive/). Na těchto stránkách lze také najít podrobný návod na instalaci.

Po nainstalování Subversive je potřeba ještě doinstalovat některý z SVN konektorů. Dialogové okno s nabídkou se objeví buď po restartu Eclipse nebo při vytváření repozitáře. Po výběru se Eclipse připojí na patřičné URL a konektor stáhne a nainstaluje. Máme na výběr ze dvou konektorů. JavaHL, který je vyvíjen společně se Subversion jako základní Java konektor k SVN. Jeho výhodou je, že nová verze je k dispozici ihned s každou novou verzí Subversion. Nevýhodou je jeho nativní implementace závislá na Eclipse. Pokud tedy dojde k pádu JavaHL, přestane pracovat i celá aplikace Eclipse. Druhou možností je konektor SVNKit, což je samostatný Java klient, který může běžet na jakémkoli operačním systému. Oproti JavaHL tedy nezpůsobí pád Eclipse, ale nevýhodou je zase zpožděné vydání nové verze oproti verzi Subversion [5].

4 Připojení k databázi

Tato kapitola popisuje možnosti připojení k databázi v jazyce Java pomocí JDBC a pluginy pro Eclipse, které poskytují grafické uživatelské rozhraní pro připojení k databázi a její správu.

4.1 JDBC

JDBC (Java Database Connectivity) je aplikační rozhraní (API) napsané v jazyce Java a slouží pro přístup k databázi a obecně jakémukoli druhu tabulkových dat. Díky JDBC API můžeme napsat aplikaci, která dokáže přistupovat k jakémukoli systému řízení báze dat (SŘBD). Není tedy potřeba psát jednu aplikaci pro přístup k MySQL databázi a jinou pro přístup k Oracle databázi. JDBC API nám umožní napsat jediný program, který bude schopný pomocí SQL příkazů přistoupit k příslušnému zdroji dat.

JDBC vychází z ODBC (Open Database Connectivity), které se používá pro přístup a správu databáze obecně. Pomocí ODBC se lze připojit skoro k jakékoli databázi. Nelze ho však použít přímo z Javy, protože používá rozhraní jazyka C. Navíc i použití jednoduchého dotazu nad databází vyžaduje hlubší znalosti ODBC. Pro zjednodušení a hlavně umožnění přístupu k databázi při programování v jazyce Java, bylo vytvořeno právě rozhraní JDBC API.

JDBC API však nebylo navrženo pouze pro přímé volání SQL příkazů, ale i jako základ pro vytvoření dalších nástrojů, které jsou pro uživatelé mnohem přívětivější. Patří mezi ně např. JDO (Java Data Objects), což je technologie schopná mapovat Java objekty na relační databázi. Využívá JDBC k mapování Java tříd na tabulky databáze, kde každý řádek tabulky představuje instanci dané třídy a každý sloupec tabulky je jeden atribut (field). Uživatel tak nemusí psát přímo SQL kód, ale pracuje s těmito JDO objekty a píše kód čistě v jazyce Java [6].

Samotné připojení k dané databázi zajišťuje JDBC ovladač, který se stará o zasílání dotazů na konkrétní databázi a o zpracování výsledků. Ke každému databázovému systému je potřeba jiný JDBC ovladač. Každý ovladač musí být zaregistrován tzv. správcem ovladačů, což je Java třída (*DriverManager*), která se stará o připojení aplikace ke správnému JDBC ovladači.

JDBC ovladače se dělí na čtyři základní typy:

1) U prvního typu se využívá tzv. „JDBC-ODBC Bridge“, který slouží jako most mezi JDBC a ODBC. Používá se hlavně tam, kde k dané databázi není JDBC ovladač k dispozici. Kromě JDBC je tedy zapotřebí i ODBC ovladač, který musí být nainstalován u klienta.
2) Druhý typ rovnou překládá volání metod JDBC API na volání metod aplikačního rozhraní dané databáze. Toto rozhraní musí být na klientovi nainstalováno v nativní podobě.
3) Třetí typ ovladače využívá síťový protokol, pomocí kterého komunikuje se serverem. Server pak tento síťový protokol překládá do konkrétního protokolu dané databáze.
4) U posledního typu není mezi ovladačem a databází již žádný server. Ovladač je napsán čistě v jazyce Java a překládá volání JDBC přímo do síťového protokolu dané databáze.

Pro připojení k databázi pomocí JDBC, je potřeba nejprve zaregistrovat JDBC ovladač příkazem *Class.forName(JDBC_ovladac)*. Dalším krokem je vytvoření spojení s databází. K tomu slouží metoda *getConnection(URL, uzivatelske_jmeno, heslo)* ve třídě *DriverManager*. Tato metoda nám vrátí objekt typu *Connection*, díky kterému můžeme vytvářet SQL příkazy nebo nahlížet do metadat databáze. Připojení k MySQL databázi by mohlo vypadat např. takto:

```
Class.forName("com.mysql.jdbc.Driver");
DriverManager.getConnection("jdbc:mysql://localhost/db", "root", "root");
```

Třída *Connection* obsahuje metodu *createStatement()*, která vrací objekt typu *Statement*. Tento objekt již umožňuje provádět dotazy nad databází. Slouží k tomu metody *execute()*, *executeUpdate()* a *executeQuery()*. Pro získání metadat je ve třídě *Connection* k dispozici metoda *getMetadata()*. Ta vrací objekt *DatabaseMetaData*, ze kterého lze získat např. informace o tabulkách metodou *getTables()*.

Výsledek SQL dotazu nebo získaných metadat je v podobě objektu *ResultSet*, přes který lze iterovat a postupně tak získat např. jednotlivé řádky dotazu *SELECT* nad danou tabulkou. Pro iteraci slouží metoda *next()*, která ve výsledku posune pozici kurzoru na další řádek. Konec iterace lze poznat z návratové hodnoty. Metoda vrací *false* v případě, že už nejsou ve výsledku k dispozici další řádky.

4.2 Nástroje pro správu databází v Eclipse

Pro Eclipse existuje několik nástrojů (pluginů), které umožňují připojit se v podstatě k jakékoli databázi a pomocí grafického uživatelského rozhraní prohlížet a měnit obsah této databáze. Všechny tyto nástroje pracují na podobném principu. Pro připojení k databázi je potřeba příslušný JDBC ovladač a samozřejmě URL adresa, uživatelské jméno a heslo k dané databázi. JDBC ovladač musí být ve formátu JAR souboru. Po úspěšném připojení je zpravidla obsah databáze zobrazen ve stromové struktuře. Lze tak snadno prohlížet jednotlivé tabulky, definice sloupců, primárních a cizích klíčů apod.

Pro editaci jednotlivých záznamů v tabulce slouží tabulkový procesor, podobný jiným běžným aplikacím pro správu tabulkových dat (např. MS Excel). Při editaci jednotlivých hodnot v tabulce se vygeneruje příslušný SQL příkaz, který provede aktualizaci dané tabulky. Mezi další funkce většiny nástrojů pro správu databází patří i možnost exportu schématu databáze do souboru obsahujícího SQL příkazy pro vytvoření tabulek a dalších objektů databáze. Samozřejmě i obsah tabulek je možné uložit do souboru a to např. ve formátu CSV. U většiny nástrojů je součástí i SQL editor pro psání SQL příkazů, které lze nad danou databází provádět. Tento editor barevně odlišuje klíčová slova a používá automatické doplňování kódu, což psání SQL příkazů výrazně zjednodušuje.

Mezi nejpoužívanější pluginy patří DTP (Data Tools Platform), který je k dispozici na oficiálních stránkách Eclipse (http://www.eclipse.org/datatools/). Velice podobným nástrojem je QuantumDB (http://quantum.sourceforge.net/), který se od DTP liší snad jen v grafických detailech. Oba dva nástroje zobrazují obsah databáze ve stromové struktuře a v základní funkčnosti se téměř neliší. Po instalaci každý přidá novou samostatnou perspektivu a je tak možné mít nainstalovány oba nástroje zároveň.

5 Analýza a návrh verzování

Abychom mohli databázi verzovat, je potřeba provést export její struktury a obsahu do souboru, který bude možné nahrát do repozitáře. Některé nástroje určené pro zálohování, exportují databázi do binárního souboru. Jiné zase používají obyčejný textový soubor, který obsahuje SQL příkazy pro znovu vytvoření struktury a obsahu databáze. SQL příkazy pro vytvoření jednotlivých databázových objektů jsou podrobněji popsány v kapitole 5.1.

Pro verzování databáze je potřeba mít její exportovanou podobu v textovém formátu, aby se daly jednotlivé verze snadno porovnávat. Proto jsem zvolil právě způsob, který spočívá v exportu databáze do souborů obsahující SQL příkazy. Tyto obyčejné textové soubory lze snadno verzovat a rozdíly mezi jednotlivými revizemi tak jednoduše zobrazit, stejně jako při verzování souborů se zdrojovým kódem.

Tyto SQL soubory už pak lze jednoduše nahrát (*commit*) do repozitáře. Samozřejmě systém Subversion sám zajistí porovnání s předchozí verzí a nahraje do repozitáře pouze změny v souboru a ne kompletní soubor.

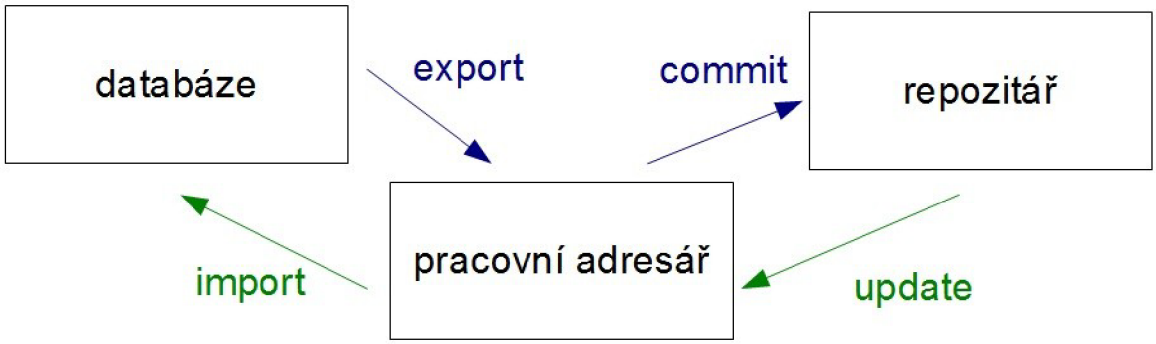
Při aktualizaci (*update*) databáze z repozitáře proběhne aktualizace SQL souborů do pracovního adresáře. Změny, které byly v pracovním adresáři provedeny a zatím nejsou v repozitáři, jsou automaticky sloučeny se změnami z nové revize. Toto opět řeší systém Subversion automaticky. Pokud však není automatické sloučení změn možné, je zahlášen konflikt, který musí uživatel vyřešit sám. Druhým krokem při aktualizaci (*update*) je import aktualizovaných souborů do databáze. Jelikož tyto soubory obsahují SQL příkazy, jejich vykonáním se do databáze promítnou změny z nové revize v repozitáři.

V obou situacích (*commit* i *update*) je tedy vždy potřeba provést dva kroky. Jednou možností by bylo tyto kroky sloučit a provádět export a import automaticky. Znamenalo by to tedy provést

při každé aktualizaci z repozitáře (*update*) automaticky i import SQL souborů do databáze. Obdobně by se před nahráním (*commit*) nové revize do repozitáře provedl automaticky export databáze do SQL souborů. To by ale znamenalo, že by se export a import prováděl i v případech, kdy by nebyl vůbec potřeba. Navíc by byl uživatel závislý na jednom konkrétním nástroji pro práci s repozitářem.

Proto jsem se rozhodl, že budou akce pro práci s repozitářem (*update*, *commit*) a akce pro práci s databází (*export*, *import)* odděleny. Uživatel tak může použít pro správu verzí jakýkoli plugin v Eclipse. Stejně tak může použít i jakoukoli samostatnou aplikaci (např. TortoiseSVN) a samotné verzování tak provádět mimo Eclipse. Samozřejmě může použít i úplně jiný verzovací systém než Subversion, jako je např. Git.

Postup akcí, které je nutné provést při nahrání nové revize nebo aktualizaci z repozitáře, je znázorněn na obrázku 2.



Obrázek 2: Schéma operací při verzování databáze

Většina nástrojů pro exportování databáze vytvoří jeden soubor, tzv. SQL dump, který obsahuje příkazy pro vytvoření struktury i obsahu databáze. Pokud bychom pro verzování použili pouze tento soubor, bylo by sledování změn a samotné verzování velice komplikované a nepřehledné. Jestliže by dva uživatelé provedli zároveň změnu dvou na sobě nezávislých tabulek, v repozitáři by se vždy jednalo o změnu jediného souboru a mohlo by dojít ke konfliktu, který by jeden z uživatelů musel ručně vyřešit. Z toho důvodu jsem zvolil řešení, kde je export každé tabulky v samostatném souboru a navíc jsou data oddělena od schématu.

5.1 Databázové objekty

Pro verzování databáze je potřeba provést export objektů databáze do souboru, který bude obsahovat SQL příkazy pro opětovné vytvoření těchto objektů v rámci importu do databáze. Tato kapitola ve zkratce popisuje SQL příkazy potřebné pro vytvoření daných databázových objektů. Podrobnější informace a příklady užití lze nalézt v [7] a [8].

Jádrem každé databáze jsou tabulky. Pro jejich vytvoření slouží příkaz *CREATE TABLE* následovaný výčtem sloupců, které daná tabulka obsahuje. Definice každého sloupce se skládá z jeho názvu a typu. Dále mohou být použity následující klíčová slova:

- *NOT NULL* pro zajištění toho, že hodnoty v daném sloupci nebudou prázdné
- *AUTO_INCREMENT* slouží pro automatické zvyšování hodnoty v daném sloupci při vložení nového záznamu (řádku)
- *CHECK* obsahující podmínku (uzavřenou v závorkách), která může udávat např. rozmezí hodnot, které lze v daném sloupci uchovávat
- UNIQUE zaručuje, že hodnoty v daném sloupci budou unikátní a na žádném řádku se nebudou opakovat
- *PRIMARY KEY* pro označení sloupce jako primárního klíče

Příkaz pro vytvoření tabulky může na konci obsahovat ještě definici primárního klíče a cizích klíčů. Definice primárního klíče až za definicí sloupců se používá hlavně v případě, kdy je primární klíč složený z více sloupců. Klíč se definuje pomocí klíčového slova *CONSTRAINT* a názvem daného klíče. Dále je nutné uvést typ klíče, tedy *PRIMARY KEY*. A nakonec do závorek sloupec (nebo sloupce), ke kterému se klíč vztahuje. Celý příkaz má tedy tuto podobu:

```
CONSTRAINT název_primárního_klíče PRIMARY KEY (sloupec1, sloupec2, ...)
```

Definice cizího klíče vypadá podobně. Obsahuje ale navíc ještě klíčové slovo *REFERENCES* spolu s názvem tabulky a jejího sloupce (sloupců), na který se cizí klíč odkazuje. Volitelně lze k definici dopsat akce, které se mají provést při aktualizaci (*ON UPDATE*) nebo smazání (*ON DELETE*) záznamu z tabulky. Lze tak například nadefinovat, aby se při smazání záznamu z tabulky s primárním klíčem smazaly i záznamy, které se na tento primární klíč odkazují. Mezi akce, které lze použít při aktualizaci nebo smazání patří *CASCADE*, *RESTRICT*, *SET NULL*, *NO ACTION* a *SET DEFAULT*.

```
CONSTRAINT název_cizího_klíče FOREIGN KEY (sloupec1, sloupec2, ...) REFERENCES
odkazovaná_tabulka (sloupec1, sloupec2, ...) ON UPDATE akce ON DELETE akce
```

Primární i cizí klíč lze do tabulky přidat i po jejím vytvoření a to příkazem *ALTER TABLE*. Příkaz zůstává stejný, jako když definujeme klíč při vytváření tabulky, pouze na začátek připíšeme:

```
ALTER TABLE jméno_tabulky ADD příkaz_pro_definici_klíče
```

Primární i cizí klíč se řadí do skupiny databázových indexů. Existují i indexy, které na primární klíč nemají žádnou vazbu a slouží hlavně pro rychlejší vyhledávání záznamů v tabulce. Využijí se tedy hlavně v případech, kdy tabulka obsahuje velké množství záznamů. Každý index se vztahuje ke konkrétnímu sloupci (sloupcům) dané tabulky. Index vytvoříme příkazem *CREATE INDEX*

nebo *CREATE UNIQUE INDEX*, který zajistí, že ve sloupci, ke kterému se index vztahuje, nebudou duplicitní data. Dále následuje jméno indexu, klíčové slovo *ON* a nakonec jméno tabulky a sloupce (nebo sloupců), nad kterým je index vytvářen.

```
CREATE [UNIQUE] INDEX jméno_indexu ON jméno_tabulky (sloupec1, sloupec2, ....)
```

Pro zajištění unikátních hodnot ve sloupci je, kromě vytvoření unikátního indexu, možno i přidat omezení (constraint) na daný sloupec. Význam je stejný, jako když označíme sloupec jako UNIQUE při vytváření tabulky. Zde použijeme příkaz *ALTER TABLE* pro přidání omezení k již existující tabulce.

```
ALTER TABLE jméno_tabulky ADD CONSTRAINT název UNIQUE
(sloupec1, sloupec2, ...)
```

Další objekty, se kterými je nutné při exportu databáze počítat jsou pohledy (view), což jsou pouze virtuální tabulky. Pohled neobsahuje žádná data a je tak závislý na jiné tabulce, ze které data získává. Jeho součástí je tak příkaz (většinou příkaz *SELECT*), který poskytne náhled na data jedné nebo více tabulek. Pro vytvoření pohledu slouží příkaz *CREATE VIEW*, který musí obsahovat jméno pohledu. Za tímto jménem lze libovolně uvést jména sloupců uzavřených do závorky. Na závěr je nutno uvést klíčové slovo *AS* a samotný příkaz *SELECT*.

```
CREATE VIEW jméno_pohledu [(sloupec1, sloupec2, ....)]  AS SELECT...
```

Důležitou součástí každé databáze je i její obsah, přesněji obsah všech tabulek. Data, která mají být v tabulce obsažena, vložíme do tabulky příkazem *INSERT INTO*, za kterým následuje jméno dané tabulky. Dále je nutné uvést klíčové slovo *VALUES* a v závorce jednotlivé hodnoty odděleny čárkou. Každý příkaz vloží do tabulky jeden řádek, neboli jeden nový záznam. Hodnoty za klíčovým slovem *VALUES,* musí být v pořadí, v jakém byly uvedeny sloupce tabulky při jejím vytvoření.

```
INSERT INTO jméno_tabulky VALUES (hodnota1, hodnota2...)
```

6 Implementace

Aplikace je implementovaná jako zásuvný modul (plugin) pro Eclipse a je tedy kompletně napsaná v jazyce Java. Aplikaci jsem vyvíjel v prostředí Eclipse, konkrétně ve verzi Indigo s již nainstalovanými nástroji pro usnadnění vývoje pluginů. Tato verze má označení Eclipse for RCP and RAP Developers. Pro návrh grafického uživatelského rozhraní (GUI) jsem využil plugin WindowBuilder, který obsahuje WYSIWYG editor pro snadné vytvoření GUI aplikace. Jednoduchým přetažením myši tak lze na formulář umístit základní komponenty knihovny SWT

jako tlačítka, popisky, apod. Zdrojový kód je nástrojem WindowBuilder generován automaticky, ale nedochází k přepsání celého souboru, takže je možné upravit vygenerovaný zdrojový kód i ručně.

Pro práci s databází jsem použil balíček *java.sql*, který obsahuje základní třídy a rozhraní pro připojení k databázi pomocí JDBC. Díky tomu je možné se připojit k jakémukoli databázovému systému, pouze je potřeba mít k dispozici příslušný JDBC ovladač v podobě zkompilovaného JAR archivu.

6.1 Export databáze

Po úspěšném načtení JDBC ovladače a připojení k databázi jsou v rámci exportu vytvořeny soubory obsahující SQL příkazy pro vytvoření schématu a obsahu databáze. Všechny soubory mají příponu *sql*, jsou umístěny ve složce *sql_export* a dále se dělí do podsložek. Pokud byly ve složkách staré soubory z předešlého exportu, jsou nejdříve všechny smazány a následně znovu vytvořeny.

Informace o objektech databáze, potřebné pro vytvoření SQL příkazů obsažených v souborech, jsou získávány z metadat, konkrétně ze třídy *DatabaseMetaData*. Pro získání dat z tabulek je použita třída *Statement*, která obsahuje metody pro vykonání dotazu nad databází. Jednoduše tak lze pomocí příkazu *SELECT* získat všechny řádky dané tabulky.

Export tabulek

Soubory pro vytvoření tabulek jsou ve složce *sql_export/tables*. Každý soubor obsahuje SQL příkaz *CREATE TABLE* pro vytvoření jedné tabulky. Název takového souboru je shodný s názvem tabulky. Pro získání názvů všech tabulek slouží metoda *getTables()* třídy *DatabaseMetaData.* Výsledek této metody je objekt *ResultSet,* který obsahuje metodu *next()*. Tato metoda vždy posune pozici kurzoru na další řádek výsledku. Postupně tak lze získat názvy tabulek a to pomocí metody *getString("TABLE_NAME")*. Další metoda *getColumns()* vrací informace o sloupcích dané tabulky, jejíž název je potřeba metodě předat jako argument. Výsledek je opět v podobě objektu *ResultSet* a stejně tak s pomocí metody *getString()* můžeme zjistit název sloupce, jeho typ atd.

Export primárních a cizích klíčů

Soubory pro vytvoření primárních klíčů jsou ve složce *sql_export/primaryKeys*. Každý soubor je pojmenován podle tabulky, ke kterému se daný klíč vztahuje a obsahuje příkaz *ALTER TABLE* pro přidání primárního klíče do tabulky. Stejným způsobem jsou vytvořeny i soubory pro cizí klíče, jsou ale umístěny ve složce *sql_export/foreignKeys*. Metadata primárních a cizích klíčů můžeme získat podobně jako u tabulek. Ve třídě *DatabaseMetaData* jsou k dispozici metody *getPrimaryKeys()* pro získání primárních klíčů a *getImportedKeys()* pro získání cizích klíčů. Tyto metody, podle názvu tabulky předaného v argumentu, vrátí potřebné informace o daném klíči.

Export pohledů

Ve složce *sql_export/views* jsou soubory obsahující příkaz *CREATE VIEW* pro vytvoření pohledu. Každý soubor je pojmenován podle daného pohledu. Názvy všech pohledů databáze lze získat stejným způsobem jako tabulky. Rozdíl je pouze v tom, že je metodě *getTables()* předán, jako poslední argument, řetězec *VIEW* a ne *TABLE*, jak je tomu u získávání tabulek. Stejným způsobem jako u tabulek, tedy metodou *getColumns()*, můžeme získat názvy sloupců. Tyto názvy lze použít pro pojmenování sloupců ve výsledku dotazu, který daný pohled obsahuje. Pro získání tohoto dotazu, neboli definice pohledu, nemá JDBC rozhraní k dispozici žádné metody. Jinou možností je tedy využití informací z tzv. informačního schématu (*information_schema*), což je v podstatě kolekce tabulek, nebo spíše pohledů, které jsou určeny pouze pro čtení. Informační schéma je součástí databáze a obsahuje metadata o tabulkách a jiných objektech dané databáze. Pro získání definice pohledu tedy použijeme příkaz *SELECT* nad tabulkou *information_schema.views*. Výsledek omezíme pouze pro daný pohled. Název pohledu se nachází ve sloupci *table_name*.

```
SELECT * FROM information_schema.views WHERE table_name='název_pohledu'
```

Pro vykonání tohoto dotazu použijeme třídu *Statement* a její metodu *executeQuery()*, která vrací objekt *ResultSet*. Z tohoto výsledku lze pak získat definici pohledu metodou *getString("view_definition")*.

Export dat

Data obsažená v tabulkách jsou umístěna ve složce *sql_export/data*. Opět je pro každou tabulku samostatný soubor pojmenovaný podle dané tabulky. V souboru jsou příkazy *INSERT INTO* pro vložení dat. Data získáme z tabulky jednoduše pomocí dotazu *SELECT*. Stačí znát pouze název tabulky.

```
SELECT * FROM název_tabulky
```

Stejně jako v předchozím případě provedeme daný dotaz pomocí metody *executeQuery()*. Vrácený výsledek v objektu *ResultSet* bude obsahovat všechny řádky z dané tabulky. Abychom získali počet sloupců, je potřeba nahlédnout do metadat výsledku, které získáme metodou *getMetaData()*. Tato metoda vrací objekt *ResultSetMetaData*, nad kterým stačí, pro získání počtu sloupců, zavolat metodu *getColumnCount()*. Počet řádků znát nemusíme, stačí v cyklu metodou *next()* posouvat pozici kurzoru na další řádek, do chvíle než metoda vrátí *false*, to znamená, že už ve výsledku další řádky nejsou. Na každém řádku můžeme postupně získat hodnoty jednotlivých sloupců metodou *getString(i)*, kde *i* je index sloupce (číslováno od jedničky). Indexy sloupců známe právě díky získanému počtu sloupců.

Export indexů

Pro získání indexů databáze slouží metoda *getIndexInfo()* ve třídě *DatabaseMetaData*. Tato metoda vrací jméno indexu, jméno tabulky i konkrétního sloupce, ke kterému se index vztahuje, a informaci o tom jestli je index unikátní. Bohužel ve výsledku této metody není rozlišeno jestli se jedná o index, který by se zpětně vytvořil příkazem *CREATE INDEX*, nebo jestli se jedná o sloupec, který byl pouze označen jako unikátní. Navíc je součástí výsledku metody *getIndexInfo()* ještě primární klíč dané tabulky, což je ve své podstatě také index, ale z daného výsledku ho nelze nijak od ostatních rozlišit. Tyto tři objekty (indexy, unikátní sloupce a primární klíč) jsou tedy ve výsledku pomíchané a není možnost jak je od sebe odlišit. Bohužel jsem nenašel ani jiný způsob, jak tyto informace o indexech získat.

6.2 Import databáze

Při importu databáze ze souborů dojde v databázi nejprve k vymazání všech pohledů a tabulek. Pro smazání pohledu i tabulky slouží příkaz *DROP*. Přidáním klíčového slova *CASCADE* na konec příkazu, docílíme odstranění cizích klíčů, které se odkazují na danou tabulku.

```
DROP VIEW název_pohledu CASCADE
DROP TABLE název_tabulky CASCADE
```

Bohužel klíčové slovo *CASCADE* v MySQL nefunguje. Je tedy potřeba nejprve smazat cizí klíče ze všech tabulek. Pro smazání cizího klíče pomocí *DROP FOREIGN KEY* použijeme příkaz *ALTER TABLE*. U některých databázových systémů, jako např. PostgreSQL, není klíčové slovo *FOREIGN KEY* podporováno a je potřeba místo něj použít klíčové slovo *CONSTRAINT*.

```
ALTER TABLE název_tabulky DROP FOREIGN KEY název_cizího_klíče
```

Po úspěšném smazání všech tabulek se postupně provedou SQL příkazy obsažené v daných souborech. Při importu je nutné dodržet určité pořadí vykonávaných příkazů. Jako první je potřeba vytvořit tabulky. Pokud by bylo součástí příkazu *CREATE TABLE* i vytvoření primárních a cizích klíčů, bylo by nutné vzít v úvahu jejich vzájemnou závislost a zajistit tak vytvoření tabulek ve správném pořadí. Právě proto jsou primární a cizí klíče v samostatných souborech, aby tak bylo možné nejprve vytvořit tabulky a teprve potom k nim přidat klíče. Samozřejmě je potřeba nejdříve vytvořit primární klíče pro všechny tabulky a až poté cizí klíče, které se na ně odkazují.

Další problém nastává při vkládání dat do tabulek pomocí příkazu *INSERT INTO*. Vložení dat není možné, pokud je již v tabulkách zajištěna referenční integrita pomocí primárních a cizích klíčů. Pokud je tabulka s primárním klíčem stále prázdná, nelze do tabulky s cizím klíčem vložit nové řádky. Primární klíč, na který se ten cizí odkazuje, v tu chvíli totiž ještě neexistuje. Aby nebylo potřeba složitě určovat pořadí v jakém vkládat data do tabulek, jsou nejprve do tabulek

vložena data a potom přidány primární a cizí klíče. Při vytváření pohledů je pouze potřeba, aby již v databázi existovaly tabulky, ke kterým se daný pohled vztahuje.

Import všech SQL souborů je uzavřen do transakce. Před samotným zásahem do databáze je zrušeno nastavení, které automaticky promítne, pomocí příkazu *commit,* každý SQL příkaz do databáze. Slouží k tomu metoda *setAutoCommit()* ve třídě *Connection.* Pokud proběhne import do databáze bez problému, jsou potvrzeny všechny změny dané transakce v databázi pomocí metody *commit()*. Při jakékoli chybě dojde k navracení změn, které byly v transakci provedeny. Opět je k tomuto účelu ve třídě *Connection* metoda *rollback()*.

6.3 Popis architektury pluginu

Nejdůležitější Java třída pluginu je třída *Activator*, která byla automaticky vytvořena pomocí průvodce. Tato třída dědí ze třídy *org.eclipse.ui.plugin.AbstractUIPlugin* a stará se o životní cyklus pluginu. Obsahuje metodu *start(),* která se spustí při načtení pluginu, což nemusí být při spuštění Eclipse, ale většinou to bývá při prvním použití pluginu. Dále obsahuje metodu *stop()*, která slouží k zastavení pluginu a uvolnění všech zdrojů. Většinou se tato metoda volá spolu s ukončením aplikace Eclipse.

Druhou důležitou součástí definice mého pluginu je Java třída *ActionDelegate*, jejíž základ byl také vytvořen pomocí průvodce. Tato třída implementuje rozhraní (interface) *org.eclipse.ui.IObjectActionDelegate* a obsahuje metodu *run()*, která otevírá hlavní dialog pro import a export databáze.

Hlavní dialog pluginu je definován ve třídě *DatabaseConnectionDialog*, která má jedinou veřejnou metodu *open()*. Tato metoda vytvoří všechny prvky v dialogu, umístí je na definované souřadnice, tlačítkům přiřadí akce, které se mají provést při jejich stisknutí, načte předchozí nastavení ze souboru a konečně otevře dialog.

Pro připojení k databázi slouží třída *DatabaseConnection*, která v sobě drží informace potřebné k připojení a obsahuje metodu *connect()* pro připojení k databázi. Zadané informace jsou uloženy do souboru pomocí třídy *Settings,* která obsahuje metody *load()* a *save().*

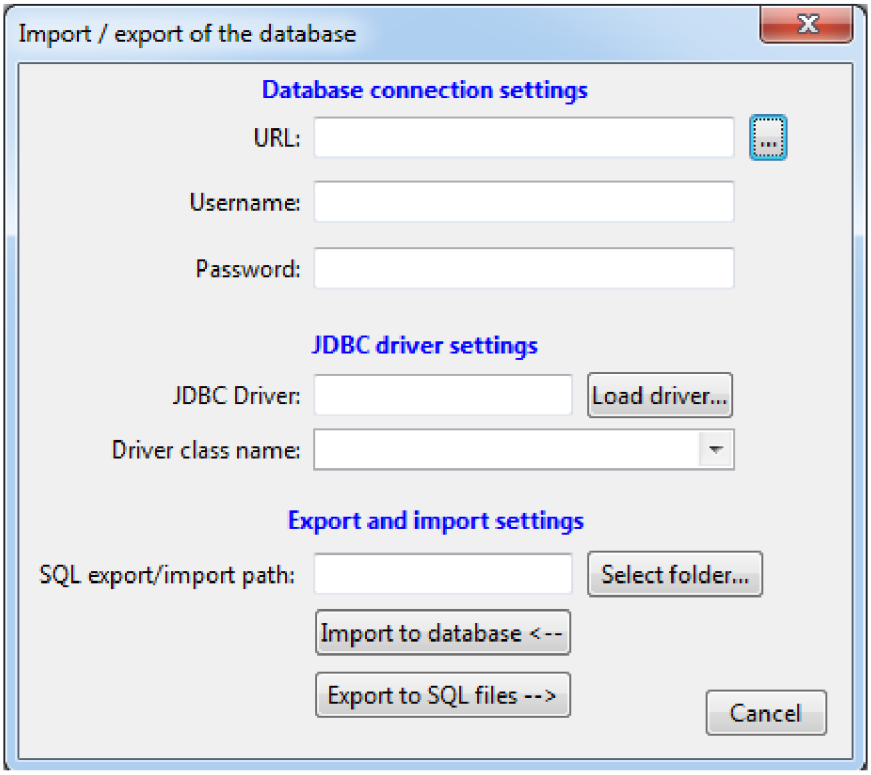
O import a export databáze se stará třída *SqlExporter*, která má dvě veřejné metody, *executeImport()* a *executeExport()*. Tyto metody jsou volány ze třídy *DatabaseConnectionDialog* po kliknutí na tlačítka *Import* a *Export*.

Definice toho, jak je vyvíjený plugin zapojen do grafického uživatelského rozhraní Eclipse, je definována v manifestu pluginu, tedy v soubor s názvem *plugin.xml*, jehož obsah je součástí přílohy A. Je zde nadefinované rozšíření (*extension*), které udává, že bude přidána nová položka do kontextového (neboli popup) menu. Dále je zde definice toho, že se nová položka objeví v kontextovém menu pouze po kliknutí pravým tlačítkem na projekt. Jako poslední je definice akce, tedy Java třídy *ActionDelegate*, která obsahuje metodu *run()*. Tato metoda se spustí po kliknutí na danou položku v kontextovém menu. V definici této akce je také uveden popisek a ikona položky. I přesné umístění položky menu je součástí definice akce, konkrétně se jedná o vlastnost *menubarPath*, která udává konkrétní cestu.

7 Instalace a použití

Plugin je i se zdrojovými kódy zkompilován do jednoho JAR archivu, který má název *cz.vut.fit.xpalar01.dbversioning_1.0.0.201205070921.jar*. Tento soubor stačí zkopírovat do složky *plugins* ve složce se samotnou aplikací Eclipse. Po restartu aplikace se v kontextovém menu projektu objeví nová položka *Team -> Database import/export*. Pokud chceme databázi verzovat přímo z prostředí Eclipse, musíme doinstalovat jeden z pluginů popsaných v kapitole 3.2.

Projekty jsou zobrazeny v okně, resp. v pohledu (view) *Project Explorer*, který je většinou v levé části aplikace. Import a export databáze lze provádět pouze nad celým projektem, je tedy potřeba, pro zobrazení kontextového menu, kliknout pravým tlačítkem na daném projektu. Po vybrání položky *Team -> Database import/export* se zobrazí dialogové okno. V tomto dialogu (na obrázku 3) je potřeba nastavit připojení k databázi, JDBC ovladač a cestu pro export.



Obrázek 3: Dialog pro import a export databáze

V nastavení připojení k databázi je nutné uvést URL, uživatelské jméno a heslo k databázi. Pro snadnější nastavení URL, slouží dialog, ve kterém stačí zadat typ databáze, jméno hostitele neboli host name (většinou localhost) a název databáze. Z těchto informací se pak automaticky vytvoří správné URL pro připojení přes JDBC.

Další dvě nastavení se týkají JDBC ovladače. Jako první je potřeba načíst samotný JDBC ovladač v podobě JAR archivu, nebo zadat cestu k tomuto souboru ručně. Každý JDBC ovladač obsahuje Java třídu, která implementuje rozhraní *java.sql.Driver*. Tato třída se poté zaregistruje ve

správci ovladačů *(DriverManager).* A právě název této třídy je druhý údaj, který je nutno zadat. Pro ovladače pro PostgreSQL a MySQL jsou názvy tříd již předdefinovány.

Poslední pole slouží pro zadání cesty ke složce, ve které budou uloženy exportované SQL soubory. Stejně tak se budou z této složky načítat soubory při importu do databáze. Při prvním otevření dialogu je tato cesta automaticky nastavena na cestu, ve které je umístěn samotný projekt.

Poté už stačí použít tlačítko pro import do databáze nebo export do SQL souborů. Samozřejmě je před importem a po exportu provedena aktualizace (*refresh*) SQL souborů v pracovním prostředí (workspace). Důvod je ten, že Eclipse ve standardním nastavení neprovádí aktualizaci souborů a při následném nahrání do repozitáře (*commit*), by nebyly detekovány žádné změny.

Při importování nebo exportování databáze je zadané nastavení uloženo do souboru *database_connection.ini*, který je automaticky uložen do kořenové složky projektu. Při dalším použití je dané nastavení z tohoto souboru načteno a není ho tedy potřeba vyplňovat znovu.

Po úspěšném exportu databáze je dialog uzavřen a exportované soubory můžeme nahrát (*commit*) do repozitáře. Pokud máme nainstalován *Subversive* nebo *Subclipse* plugin, klikneme pravým tlačítkem na daný projekt a zvolíme *Team -> Commit*. Otevře se dialog, ve kterém vidíme nové a změněné soubory připravené pro nahrání do repozitáře. Aktualizaci z repozitáře (*update*) provedeme obdobným způsobem, tedy *Team -> Update*.

8 Testování

Testy implementované aplikace (pluginu) jsem prováděl na 64bitovém systému Windows 7 Professional. Stejně jako pro vývoj jsem pro testování použil Eclipse verzi Indigo a navíc ještě verzi Helios, konkrétně Eclipse for PHP Developers. Pro systém Subversion jsem nainstaloval plugin *Subclipse*, popsaný v kapitole 3.2. Jako testovací databázové systémy jsem vybral dva nejpoužívanější: MySQL a PostgreSQL. Jelikož ale aplikace využívá rozhraní JDBC, neměl by být problém připojit se k jakémukoli jinému databázovému systému. Stačí pouze správně zadat URL k databázi a mít k dispozici příslušný JDBC ovladač.

Aplikaci jsem testoval v průběhu celého vývoje, postupně odstraňoval nalezené chyby a snažil se vyřešit rozdíly mezi systémy MySQL a PostgreSQL jako např. nefunkčnost klíčového slova *CASCADE* v příkazu *DROP TABLE* u MySQL, nebo odstraňování cizích klíčů. U MySQL bylo potřeba použít klíčové slovo *FOREIGN KEY*, zatímco u PostgreSQL klíčové slovo *CONSTRAINT*.

Další problém, na který jsem narazil, byl při exportu primárních klíčů. Pro získání názvu primárního klíče slouží metoda *getString("PK_NAME")*. U databáze PostgreSQL tato metoda správně vrátila název primárního klíče dané tabulky. U databáze MySQL však tato metoda vždy vrátila řetězec PRIMARY a požadovaný název nebylo možné získat. Databázi jsem vytvářel pomocí nástroje *phpMyAdmin,* ve kterém je vidět, že název primárního klíče je PRIMARY. Tato informace je tam i přesto, že jsem při vytváření primárního klíče zadal konkrétní název. Samozřejmě název PRIMARY nemůže být názvem primárního klíče, protože je to vyhrazené klíčové slovo. Znamená to tedy, že MySQL neumí uchovat název primárního klíče, přestože byl specificky zadán. Z toho důvodu chybí v exportovaných souborech, které obsahují primární klíč,

název daného klíče. Při následném importu tohoto bezejmenného klíče do databáze, proběhne import bez problému. Avšak pokud bychom se pokusili soubory exportované z MySQL importovat do PostgreSQL, nastane chyba právě v neuvedení názvu primárního klíče.

Testování transakčního zpracování a nalezené problémy jsem podrobněji popsal v kapitole 8.2. Největší část testování je věnována souběžnému vývoji, tedy situacím které mohou nastat pokud se na vývoji podílí více lidí a používají systém Subversion pro verzování. Nejčastějším problémem je úprava jednoho souboru více uživateli a právě tento případ je podrobně popsán v kapitole 8.1.

8.1 Souběžný vývoj

Souběžný vývoj aplikací s použitím verzovacího systému Subversion je velice snadný. Jednotliví uživatelé mohou nahrávat do systému nové změny, aniž by museli mít ve svém pracovním adresáři nejnovější verzi z repozitáře.

Jediný problém nastává v případě, kdy má konkrétní upravovaný soubor v repozitáři novou revizi. Při pokusu o nahrání *(commit)* tohoto souboru do repozitáře, dojde k chybě a uživatel je nucen provést aktualizaci (*update*) tohoto souboru. V tu chvíli mohou nastat dvě situace. V tom lepším případě jsou jednotlivé změny od obou uživatelů na různých místech souboru a systém Subversion je schopen tyto změny sloučit (*merge*). V tom horším případě dojde ke konfliktu a uživatel musí sloučení souborů provést ručně. A právě otestování současné změny stejného souboru více uživateli je součástí této podkapitoly.

Abych simuloval více uživatelů, vytvořil jsem v Eclipse dvě pracovní prostředí (workspace), zároveň jsem vytvořil dvě databáze v systému MySQL. Dále v textu budu používat pojmenování uživatel A pro první pracovní prostředí a databázi, stejně tak uživatel B pro druhé pracovní prostředí a databázi.

Vytvořil jsem v databázi několik tabulek s primárními i cizími klíči a naplnil je vzorky dat. Jednotlivé kroky, které jsem provedl pod oběma uživateli, jsou dále sepsány v bodech. Nahrání nové změny bylo u uživatele A otázkou několika základních kroků. U uživatele B už bylo potřeba, v rámci řešení konfliktu, provést kroků více. Kroky 1 až 3 byly shodné, s tím, že třetí krok nebylo možné dokončit a řešení tohoto problému je popsáno v dalších krocích.

Postup uživatele A:

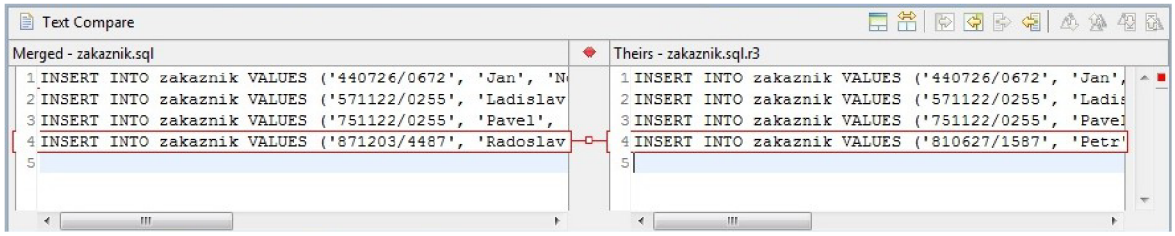
1. Změna v tabulce – v databázi jsem do tabulky s názvem *zakaznik* přidal jeden záznam (řádek dat).
2. Export do SQL souborů – v Eclipse jsem provedl export databáze do SQL souborů, pomocí kontextového menu *Team -> Database import/export*. Došlo ke změně souboru *zakaznik.sql*.
3. Commit – nad celým projektem jsem, pomocí kontextového menu *Team -> Commit,* otevřel dialog, ve kterém systém Subversion správně detekoval změnu v souboru *zakaznik.sql*. Potvrzením dialogu jsem tento souboru nahrál do repozitáře.

Postup uživatele B:

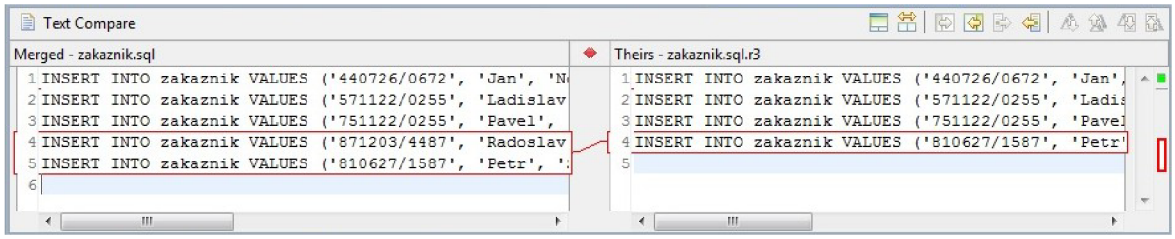
0. Neprovedl jsem aktualizaci – změny, které provedl uživatel A a nahrál do repozitáře, jsem neaktualizoval *(update)* do pracovního adresáře. Uživatel B tak měl stále starou verzi.
1. Změna v tabulce – v databázi jsem do tabulky s názvem *zakaznik* přidal jeden záznam (řádek dat).
2. Export do SQL souborů – v Eclipse jsem provedl export databáze do SQL souborů, pomocí kontextového menu *Team -> Database import/export*. Došlo ke změně souboru *zakaznik.sql*.
3. Commit – pokusil jsem se změněný soubor *zakaznik.sql* nahrát do repozitáře pomocí kontextového menu *Team -> Commit*. Jelikož soubor nebyl aktualizován z repozitáře, systém Subversion zahlásil chybu. Objevil se dialog upozorňující, že operaci nelze provést, protože daný soubor má v repozitáři novou revizi. V angličtině je tato chyba označována jako *„file is out of date"*.
4. Update – provedl jsem tedy aktualizaci z repozitáře pomocí kontextového menu *Team -> Update*. Systém Subversion se pokusil sloučit obě změny, ale jelikož byly tyto změny na stejném řádku, došlo ke konfliktu.
5. Commit - hlášení o tom, že nastal konflikt bylo vypsáno pouze do konzole a bylo lehce přehlédnutelné. Jakožto nepozorný uživatel jsem se tedy pokusil znovu o nahrání změn do repozitáře. Systém Subversion opět zahlásil chybu. Objevil se dialog s upozorněním, že nastal problém, protože je soubor *zakaznik.sql* v konfliktním stavu.
6. Nalezení konfliktního souboru – pro vyřešení konfliktu je potřeba najít v projektu složku *sql_export* a v ní daný konfliktní soubor, v našem případě *zakaznik.sql*. Vedle tohoto souboru se po aktualizaci objevili další tři soubory. Jsou to jednotlivé verze daného souboru a v názvu mají navíc právě číslo verze. Jedná se o následující verze. Verze souboru se změnou od uživatele A (tato verze je již v repozitáři), verze souboru se změnou od uživatele B (ta je zatím v pracovním adresáři) a předchozí verze souboru, vůči které byly obě změny provedeny. Samozřejmě je zde stále soubor s původním názvem, který je v konfliktním stavu a obsahuje označené změny od obou uživatelů.
7. Zobrazení konfliktu – nalezený konfliktní soubor jsem zobrazil v editoru pro řešení konfliktů a to pomocí kontextového menu *Team -> Edit conflicts*. Porovnání obou změn v editoru je na obrázku 4. V levé části je změna uživatele B, kterou je potřeba nahrát do repozitáře. V pravé části je změna od uživatele A, kterou je potřeba zachovat. Je vidět, že uživatel B opravdu změnil stejný řádek (řádek č.4) jako uživatel A.
8. Vyřešení konfliktu – vyřešení tohoto konfliktu bylo velice snadné. Pomocí tlačítka *Copy current changes from right to left* (šesté tlačítko zleva) jsem sloučil obě změny dohromady. Výsledek je znázorněn na obrázku 5.
9. Označení za vyřešený – po ruční úpravě konfliktního souboru zůstaly v pracovním adresáři ještě dočasné soubory, obsahující jednotlivé verze konfliktního souboru. Navíc byl daný soubor stále v konfliktním stavu. Před nahráním souboru do repozitáře je tedy nutné označit daný soubor jako vyřešený. Stačí kliknout pravým tlačítkem na daný soubor

a ve vyvolaném kontextovém menu zvolit *Team -> Mark Resolved*. Po zvolení této akce, se objevil dialog s nabídkou na konečné vyřešení konfliktu. Zvolil jsem tedy první možnost, která říká, že konflikt byl vyřešen v daném souboru.

10. Commit – poté co byl konflikt vyřešen a dočasné soubory smazány, byl soubor připraven pro nahrání do repozitáře. Stejným způsobem jako vždy, tedy pomocí kontextového menu *Team -> Commit*, jsem nové změny souboru *zakaznik.sql* bez problémů nahrál do repozitáře a změny, které provedl uživatel A, byly zachovány.



Obrázek 4: Porovnání dvou verzí konfliktního souboru



Obrázek 5: Vyřešení konfliktního souboru

8.2 Transakční zpracování

Jak již bylo zmíněno v kapitole 6.2, import do databáze je uzavřen do transakce. Následně jsem tedy toto transakční zpracování otestoval v databázích MySQL a PostgreSQL.

U databázového systému PostgreSQL bylo použití transakcí bezproblémové. Pokud došlo při importu do databáze k jakékoli chybě, vždy se provedené změny vrátily zpět pomocí operace *rollback* a tabulky, které byly na začátku importu smazány, tak byly opět obnoveny.

U systému MySQL funguje transakční zpracování jen částečně a navíc je potřeba nastavit správný typ tabulky. Tento typ, anglicky také nazývaný *„storage engine“*, určuje jakým způsobem jsou databázová data uložena na disku (nebo v paměti). Každý typ se liší ve způsobu indexování dat, v implementaci transakcí a mnoha jiných funkcích. Mezi tyto typy patří např. MyISAM, InnoDB aj. Aby se příkazy vykonávaly transakčně, je nutné zvolit patřičný typ tabulky. Podpora transakcí v jednotlivých typech je znázorněna v tabulce 1. Více podrobností o těchto typech

tabulek lze nalézt v [9]. Pro nastavení konkrétního typu použijeme konfigurační soubor *my.ini*. Do tohoto souboru připíšeme následující řádek.

```
--default-storage-engine=konkrétní_typ_tabulky
```

	MyISAM	InnoDB	Memory	Merge	NDB	Archive	Federated
Podpora transakcí	Ne	Ano	Ne	Ne	Ano	Ne	Ne

Tabulka 1: Podpora transakcí v různých typech tabulek MySQL

Bohužel ani při použití typu, který transakce podporuje, nejsou všechny vykonávané příkazy uzavřeny do transakce. Konkrétně se jedná o příkazy pro definici dat (DDL), tedy příkazy pro vytvoření, změnu a smazání objektů databáze. Tím se stává použití transakcí v MySQL pro mou aplikaci nepoužitelné. Pokud dojde při importu k chybě, není možné smazání tabulek vrátit zpět.

9 Další možná rozšíření

Jak jsem již zmínil v kapitole 5, samotné verzování by bylo možné zjednodušit tím, že by byly kroky pro práci s databází a kroky pro práci s repozitářem sloučené. Při každé aktualizaci by bylo nutné nějakým způsobem zjistit, jestli došlo ke změně SQL souborů a případně provést import do databáze automaticky. Stejným způsobem by se před nahráním do repozitáře prováděl automaticky export do SQL souborů. Samozřejmě zjistit, jestli došlo v databázi k nějaké změně a zda je tedy export nutný či nikoli, by bylo mnohem obtížnější. Nevýhodou tohoto řešení by byla samotná skutečnost, že by práce s repozitářem byla součástí aplikace. Uživatel, který již používá jinou aplikace pro verzování, by tak měl zbytečně druhou a navíc by byl omezen použitím systému Subversion nebo CVS.

Kromě samotné změny verzování, popsané v kapitole 9.1, by také bylo vhodné otestovat aplikaci i za použití jiných databázových systému (Oracle, SQL Server atd.) a popř. do implementace doplnit funkčnost, která by řešila problémy vzniklé kvůli rozdílům v jednotlivých systémech.

9.1 Verzování aktualizačních skriptů

Další možné rozšíření, nebo spíše změna konceptu verzování, by mohlo být verzování aktualizačních skriptů. Namísto verzování SQL souborů obsahující kompletní schéma a obsah databáze, by se do repozitáře nahrávaly pouze skripty, které by obsahovaly SQL příkazy pro aktualizaci databáze. Řešení spočívá ve vytvoření aktualizačního skriptu při každé změně v databázi a následné nahrání tohoto souboru do repozitáře. Takovýto soubor již pak není možné

měnit a při další změně databáze je vytvořen nový soubor. Soubory jsou postupně číslovány, aby bylo jasné pořadí jejich spuštění.

Problém je však v tom, že uživatel musí psát tyto aktualizační skripty ručně. Pokud bychom je chtěli vytvářet automaticky, potřebovali bychom vyvinout nástroj, který dokáže porovnat dvě databáze a ze získaného výsledku vytvořit aktualizační skript, po jehož spuštění bude stará databáze stejná jako ta nová. Vytvořit takový nástroj je však velice obtížné.

Velká výhoda tohoto řešení je ta, že při aktualizaci z repozitáře stačí pouze spustit skript, který např. přidá do tabulky jeden sloupec. Je to tedy velice rychlé a hlavně není potřeba mazat celou databázi a znovu ji vytvářet.

Problém nastává pouze ve chvíli, kdy se někdo rozhodne provést aktualizaci na předchozí verzi. V takovém případě by bylo nutné celou databázi smazat a spustit všechny aktualizační skripty od první revize.

10 Závěr

Cílem této práce bylo vytvořit nástroj, který umožní verzovat strukturu a obsah databáze při vývoji aplikací v prostředí Eclipse. Nástroj, který jsem vytvořil provádí export databáze do SQL souborů. Tyto soubory jsou logicky rozčleněny tak, aby bylo jejich verzování snadné a následný import do databáze možný pouhým spuštěním SQL příkazů, které tyto soubory obsahují. Díky rozdělení souborů dle jednotlivých objektů a tabulek, ke kterým objekty náleží, je řešení případných konfliktů jednodušší, než kdyby byl export proveden do jediného souboru.

Samotné verzování těchto SQL souborů jsem ponechal na nástrojích, které jsou již součástí Eclipse, nebo je lze doinstalovat. Díky tomu není uživatel závislý na mé aplikaci a může pro verzování použít svůj oblíbený Eclipse plugin, nebo úplně samostatnou aplikaci. Pokud se rozhodne verzovat databázi přímo v prostředí Eclipse, všechny položky pro práci s repozitářem i má přidaná položka pro práci s databází jsou v kontextovém menu na stejném místě.

Snažil jsem se nástroj vytvořit tak, aby bylo možné připojit se k jakémukoli databázovému systému. Dle specifikace aplikačního rozhraní, které jsem použil (JDBC), by k tomu měl být zapotřebí pouze ovladač k příslušné databázi. Aplikaci jsem však testoval pouze na systémech MySQL a PostgreSQL. Už u těchto dvou systémů byly značné rozdíly mezi získanými metadaty, pravidly pro vytváření a mazání databázových objektů apod. Věřím proto, že i u ostatních databázových systémů tyto rozdíly budou a bylo by tedy potřeba každý systém otestovat a vypořádat se s těmito rozdíly v implementaci.

Můj cíl, vytvořit nástroj, který dokáže verzovat obecně jakoukoli databázi, se mi tedy nejspíš splnit nepodařilo. Dle mého názoru, ani tohoto cíle obecnou cestou dosáhnout nelze. Bylo by nutné implementovat import a export databáze pro každý systém zvlášť a poté vždy vybrat konkrétní implementaci dle použitého databázového systému.

Seznam tabulek

Tabulka 1: Podpora transakcí v různých typech tabulek MySQL....26

Seznam obrázků

Obrázek 1: Porovnání kontextového menu pro Subclipse a Subversive....10
Obrázek 2: Schéma operací při verzování databáze....14
Obrázek 3: Dialog pro import a export databáze....21
Obrázek 4: Porovnání dvou verzí konfliktního souboru....25
Obrázek 5: Vyřešení konfliktního souboru....25

Literatura

[1] Gallardo, D.; Burnette, E.; McGovern, R.: Eclipse in Action: A Guide for the Java Developer. Greenwich, CT: Manning Publications, 2003, ISBN 1-930110-96-0

[2] Eclipse Plugin Development Tutorial [online]. 2008 [cit. 2012-01-23]. Dostupné z WWW: <http://www.eclipsepluginsite.com/index.html>

[3] Collins-Sussman, B.; Fitzpatrick, B.; Pilato, C.: Version Control with Subversion. O'Reilly Media, 2011. Dostupné z WWW: <http://svnbook.red-bean.com/>

[4] Jirkovský, L.: Systémy pro správu verzí. 2010. Dostupné z WWW: <http://stativ.kx.cz/src/index.php?text_id=15>

[5] Polarion Software: What are the differences between SVN Connectors and which one to choose? [online]. 2012 [cit. 2012-01-27]. Dostupné z WWW: <http://www.polarion.com/products/svn/subversive/download.php>

[6] Fisher, M.; Ellis, J.; Bruce, J.: JDBC API Tutorial and Reference, Third Edition. Addison Wesley, 2003, ISBN 0-321-17384-8

[7] Groff, J.; Weinberg, P.; Oppel, A.: SQL The Complete Reference, Third Edition. McGraw-Hill Osborne Media, 2009, ISBN 0-07-159255-5

[8] Kriegel, A.; Trukhnov, B.: SQL Bible. Indianapolis, Indiana: Wiley Publishing, 2003, ISBN 0-7645-2584-0

[9] Pachev, S.: Understanding MySQL Internals. USA: O'Reilly Media, 2007, ISBN 0-596-00957-7

Seznam příloh

Příloha A: Manifest (plugin.xml) aplikace

Příloha B: Instalace prostředí

Obsah CD

- Zdrojové kódy programu a kompletní projekt vyvíjený v Eclipse: adresář *src/*
- Text práce v elektronické podobě: adresář *text/*
- Instalační soubory nástrojů potřebných pro běh aplikace: adresář *install/*
- Zkompilovaná aplikace (plugin) pro verzování databáze: adresář *app/*
- Testovací prostředí a výsledky testů: adresář *test/*
  - Balíček Eclipse použitý pro testování: adresář *eclipse/*
  - JDBC ovladače pro MySQL a PostgreSQL: adresář *jdbc_drivers/*
  - Repozitář použitý pro testování: adresář *subversion_repository/*
  - Snímky obrazovky s postupem testu souběžného vývoje: adresář *test_db/*

Příloha A

Manifest (plugin.xml) aplikace

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
    <!-- definice umístění nové položky – v kontextovém (popup) menu -->
   <extension point="org.eclipse.ui.popupMenus">
       <!-- definice objektu, jehož kontextové menu bude obsahovat novou položku – pouze
kontextové menu projektu -->
      <objectContribution
            objectClass="org.eclipse.core.resources.IProject"
            adaptable="true"
            id="cz.vut.fit.xpalar01.dbversioning.contribution1"
      >
         <!-- definice třídy reprezentující akci, která se provede po kliknutí na položku -->
         <!-- definice ikony nové položky -->
         <!-- definice přesného umístění (cesty) nové položky - Team -> skupina 9 -->
         <action
               label="Database import/export..."
               class="cz.vut.fit.xpalar01.dbversioning.actions.ActionDelegate"
               icon="icons/database.gif"
               menubarPath="team.main/group9"
               enablesFor="1"
               id="cz.vut.fit.xpalar01.dbversioning.actions.ActionDelegate">
         </action>
      </objectContribution>
   </extension>
</plugin>
```

Příloha B

Instalace prostředí

V této příloze je návod na instalaci nástrojů, které jsou potřeba pro správný běh aplikace Eclipse. Dále je zde popis instalace databázového serveru (MySQL a PostgreSQL), popis připojení k lokálnímu repozitáři za pomocí Subclipse pluginu a konečně popis instalace a použití pluginu pro verzování databáze. Všechny instalační soubory, na které se odkazuji, jsou součástí přiloženého CD. Stejně tak jsou zde umístěny JDBC ovladače pro oba dva použité databázové systémy. Tyto ovladače lze stáhnout z oficiálních stránek. Pro PostgreSQL je to http://jdbc.postgresql.org/download.html a http://www.mysql.com/downloads/connector/j/ pro MySQL.

Instalace Eclipse

Instalační balíček lze stáhnout z oficiálních stránek: http://www.eclipse.org/downloads/. Po stažení balíčku Eclipse, jej není potřeba instalovat, stačí rozbalit archiv a spustit soubor *eclipse.exe*. Balíček, na kterém jsem prováděl testování, je součástí přiloženého CD.

Ke správnému chodu aplikace Eclipse je potřeba doinstalovat běhové prostředí JRE (Java Runtime Environment) nebo JDK (Java Development Kit). Pokud budeme používat Eclipse pro vývoj aplikací, musíme nainstalovat JDK, jinak stačí JRE. Instalační soubor prostředí JDK lze stáhnout například z http://www.oracle.com/technetwork/java/javase/downloads/index.html. Podrobný návod pro instalaci aplikace Eclipse a běhového prostředí je na těchto stránkách: http://wiki.eclipse.org/Eclipse/Installation.

Instalace nástroje pro práci se Subversion v Eclipse

Na výběr máme ze dvou pluginů. Instalace probíhá stejně jako u jakéhokoli jiného pluginu V hlavním menu Eclipse vybereme *Help->Install New Software* a zadáme URL adresu, kterou lze získat na oficiálních stránkách. Pro plugin Subclipse jsou informace k dispozici na http://subclipse.tigris.org/ a pro plugin Subversive na http://www.eclipse.org/subversive/. V balíčku Eclipse, který je součástí CD, je nainstalován plugin Subclipse.

Instalace MySQL

Abychom mohli vytvořit databázi v systému MySQL, je potřeba nainstalovat databázový server. Nejjednodušší je nainstalovat Wamp server, jehož součástí jsou i další nástroje pro správu databáze a pro vývoj aplikací v PHP. Wamp server lze stáhnout z oficiálních stránek: http://www.wampserver.com/en/. V průběhu instalace stačí ponechat umístění c:\wamp a instalaci dokončit.

Po instalaci je potřeba spustit aplikaci Wamp server, u systémových ikon na hlavním panelu se objeví ikona, na kterou stačí kliknout a zvolit *phpMyAdmin.* Prostředí se otevře ve webovém prohlížeči, kde je možné vytvářet a spravovat databáze.

Instalace PostgreSQL

Pro vytváření databází v PostgreSQL musíme nainstalovat databázový server, který se po instalaci zaregistruje jako služba systému Windows. Spouští se tedy automaticky a není již potřeba spouštět nic ručně. Instalační soubor lze nalézt na: http://www.postgresql.org/download/windows/. Při instalaci zvolíme umístění a potom heslo pro připojení k databázi. Také je potřeba zadat port, na kterém bude server naslouchat, stačí ponechat hodnotu 5432. Spolu s PostgreSQL se nainstaluje i *pgAdmin*, což je grafické prostředí, ve kterém můžeme vytvářet a spravovat databáze.

Instalace a použití pluginu pro verzování databáze

Plugin má název *cz.vut.fit.xpalar01.dbversioning_1.0.0.201205070921.jar* a je součástí přiloženého CD. Tento soubor stačí nakopírovat do složky plugins, která je součástí složky, ve které je samotná aplikace Eclipse. V Eclipse balíčku, který je součástí CD, je daný plugin již nainstalován.

Po spuštění tohoto balíčku, uvidíme projekt s názvem *testProject,* po kliknutí pravým tlačítkem na tento projekt se objeví kontextové menu. Na přiloženém CD je složka s názvem *subversion_repository*, která obsahuje repozitář, který jsem použil pro testování. Pro připojení k tomuto repozitáři stačí v otevřeném kontextovém menu vybrat *Team -> Share Project -> SVN -> Create a new repository location* -> vybrat složku *subversion_repository* -> a dokončit průvodce.

Poté se již v kontextovém menu *Team* objeví ostatní položky pro práci s repozitářem. Položka pro import a export databáze je také v kontextovém menu *Team*, v předposlední skupině, přibližně čtvrtá položka odspodu. Po kliknutí na tuto položku se objeví dialog, který je nutné vyplnit. V projektu jsou exportované soubory z MySQL databáze. Stačí tedy pomocí *phpMyAdmin* vytvořit databázi a její název poté vyplnit v dialogu. Dále je potřeba zadat uživatelské jméno a heslo pro přístup k databázi. V nastavení pro JDBC zadáme cestu k MySQL JDBC ovladači, který je umístěn na CD a také vybereme předvyplněnou Java třídu ovladače *com.mysql.jdbc.Driver*. Cestu ke složce pro export můžeme ponechat výchozí. Nyní již můžeme provést import SQL souborů do databáze.