Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Rozšíření nástroje pro synchronizaci databázových modelů

Adam Kugler

Vedoucí práce: Ing. Jiří Mlejnek

12. května 2015

Poděkování

Děkuji vedoucímu mé práce panu Ing. Jiřímu Mlejnkovi za vedení a pomoc při řešení této práce a navádění správným směrem. Díky jeho radám a připomínkám je výsledek mé práce jistě lepší a kvalitnější. Dále děkuji RNDr. Evě Kuglerové a Martinovi Ďáskovi za jazykovou korekturu textu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Kugler, Adam. *Rozšíření nástroje pro synchronizaci databázových modelů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá porovnáváním a synchronizací databázových modelů. Cílem této práce je navázat na práce kolegů z minulých let a rozšířit jimi vytvořený nástroj, který umí již pracovat s Enterprise Architektem a Oracle databází, o možnost pracovat s modely z databázových systémů MySQL a PostgreSQL. Součástí práce je analýza požadavků na toto rozšíření, jeho následný návrh a implementace. Analýza se zabývá hlavně vzájemnými převody datových typů v různých typech databází. Implementace se týká především datové vrstvy. Implementačním jazykem je Java. Nástroj by měl sloužit pro porovnávání databázových modelů nejen ze stejných databázových systémů, ale i z různých systémů, což by v důsledku umožnovalo přenést databázové schéma na jiný systém.

Klíčová slova Databázový model, porovnávání, synchronizace, datové typy, MySQL, PostgreSQL, Oracle, Enterprise Architect.

Abstract

This bachelor thesis is about comparison and synchronization of database models. Main goal is to continue in my colleagues work from previous years and extend their tool that can work with Enterprise Architect and Oracle database. Extension includes MySQL and PostgreSQL as new data sources for this tool. This thesis consists of extension requirements analysis, extension design and implementation. Analysis is mostly about data type transformation for different types of databases. Implementation relates mainly to data layer. Java is implementation language. This tool should help user with database model comparison. Models don't have to be from same database system so it is able to transfer database schema to different system.

Keywords Database model, comparison, synchronization, data types, MySQL, PostgreSQL, Oracle, Enterprise Architect.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Pokud jste vývojář a pracoval jste někdy na nějakém větším projektu, jistě jste se během vývoje setkal s problémem, že to, co jste si namodeloval během návrhu, neodpovídá výslednému řešení a že je nutné model a skutečnou implementaci udržovat v konzistentním stavu, jinak to působí problémy ostatním členům týmu a v budoucnu bude jistě i vám. Udržování modelů v konzistentním stavu se nazývá synchronizace modelů. V mé bakalářské práci se budu zabývat právě synchronizací modelů, ale pouze těch databázových. Synchronizace databázových modelů je potřeba především u větších, neustále se rozvíjejících projektů využívajících databázi a mnoho různých prostředí (vývojové, testovací, produkční,...), protože každé prostředí využívá svou vlastní databázi a nachází se v jiném stádiu vývoje aplikace. Manuální porovnávání návrhu a modelu na běžící databázi je velmi náročné, nákladné a především časté. Také je možné, že člověk udělá chybu, proto je vhodné toto porovnání dělat automatizovaně.

Když už je řeč o porovnávání databázových modelů nemusíme se omezovat pouze na porovnávání návrhu a běžící databáze, můžeme také porovnávat návrhy nebo databáze mezi sebou. Návrhy můžeme porovnávat například, když potřebujeme zjistit, v které části vývoje byla do produktu zavlečena změna a s ní i možná chyba. Na trhu existuje více databází od různých výrobců a jistě by bylo hezké, mít možnost porovnat dva databázové modely běžící na databázových strojích od různých výrobců a zjistit zda jsou tyto modely ekvivalentní. V kombinaci se synchronizací by toto umožňovalo přesunout databázový model například z MySQL na Oracle databázi.

Mým úkolem je tedy rozšířit aplikaci, která byla již dříve vyvinuta na této fakultě a která je schopná pracovat s databázovými modely z nástroje Enterpise Architect a Oracle databáze, o nové datové zdroje. Tímto rozšířením najde nástroj širší uplatnění, protože spousta uživatelů používá jiné databázové systémy než ty od společnosti Oracle. S vedoucím práce jsme vybrali databáze MySQL a PostgreSQL jako nové datové zdroje, neboť jsme přesvědčeni, že jsou mezi uživateli velmi rozšířené.

Toto téma jsem si vybral proto, že si chci prohloubit své znalosti v oblasti databází. Doposud jsem se setkal pouze s Oracle databází v rámci studia na Fakultě informačních technologií. Také se chci zdokonalit v programování v Javě. Věřím, že problémy, které budu muset řešit, mi v tomto pomohou.

Tato práce se zaměřuje na relační databáze, pokud nebude řečeno jinak, slovem databáze je tedy myšlena právě relační databáze. Dále zde typ databáze znamená typ databázového stroje (Oracle, MySQL, PostgreSQL,...).

Cíle práce

Cílem této práce je seznámit se s nástrojem dbCompare a navrhnout a implementovat rozšíření tohoto programu o nové datové zdroje, kterými jsou relační databáze MySQL a PostgreSQL.

Přidání nových datových zdrojů s sebou nese spoustu problémů, které je potřeba vyřešit. Původní aplikace totiž pracuje výhradně s modely Oracle a předpokládá, že EA modely jsou také Oracle. V nástroji Enterprise Architect si uživatel totiž může vytvořit databázový model pro mnoho typů databázových systémů. Původní program tedy neřeší porovnávaní modelů různých typů. Já bych rád zachoval možnost libovolného porovnání datových zdrojů mezi sebou tak, aby bylo smysluplné a vygenerované SQL scripty by se daly přímo aplikovat.

Teoretická část se bude zabývat především analýzou datových typů vybraných databází a jejich vzájemnými převody.

V praktické části je potřeba nejdříve implementovat datovou vrstvu pro nové typy databází a udělat drobné změny ve zbylých vrstvách. Převod datových typů bude pravděpodobně vyžadovat zásah do všech tří vrstev.

Kapitola 1

Analýza

Tato kapitola se zabývá analýzou původního nástroje vzhledem k požadovanému rozšíření. Zjišťuje se, jaké důsledky má rozšíření na chování nástroje a shromažďují se zde informace potřebné k návrhu tohoto rozšíření.

1.1 Nástroj dbCompare

Nástroj pro synchronizaci databázových modelů s názvem dbCompare je aplikace vyvinutá mými třemi kolegy v rámci jejich bakalářských prací [1, 2, 3]. Tento nástroj dovede nahrávat a porovnávat modely z Oracle databáze a nástroje Enterprise Architect. Porovnává tabulky, sloupce, komentáře, integritní omezení a v omezené míře také pohledy. Nástroj také dokáže zanášet změny do obou datových zdrojů a provádět tím synchronizaci modelů dle přání uživatele, nebo pouze vygeneruje změnové SQL scripty, které uživatel může později použít. Jedná se o třívrstvou aplikaci napsanou v jazyce Java. Aplikace se skládá z datové, business a prezentační vrstvy, každá z těchto vrstev je popsána v jedné bakalářské práci [1, 2, 3].

1.1.1 Datová vrstva

„Jedná se o vrstvu, která obstarává samotná data a převádí je do vnitřní reprezentace a naopak z této reprezentace generuje skripty a provádí požadované změny. Tato vrstva může přidávat další a další vstupy i výstupy.“ [2]

Touto vrstvou se hlouběji zabývá práce [1].

1.1.2 Business vrstva

„Business vrstva implementuje veškerou logiku aplikace, to znamená, že na základě vstupu prezentační vrstvy získává určitá data od datové vrstvy, poskytuje je dále prezentační vrstvě a provádí nad nimi operace (například po-

rovnávání). Výsledky opět poskytuje prezentační vrstvě, případně je pošle do vrstvy datové k uložení/provedení.“ [2]

Touto vrstvou se důkladně zabývá práce [2].

1.1.3 Prezentační vrstva

„Prezentuje data z nižších vrstev uživateli. Zároveň přeposílá vstupy od uživatele do nižších vrstev, které podle těchto vstupů načítají data, zpracovávají je a následně je posílají zase zpět do prezentační vrstvy, která je prezentuje uživateli. K získání těchto dat používá rozhraní, které poskytuje business vrstva.“ [2]

Touto vrstvou se podrobněji zabývá práce [3]. Následující kapitola naznačuje, že zvolené řešení této vrstvy není úplně ideální.

1.1.3.1 Nešťastná volba GUI

Můj předchůdce se ve své práci [3] rozhodl pro vytvoření grafického uživatelského rozhraní použít technologii SWT (Standard Widget Toolkit). Tato volba je dle mého názoru velmi nešťastná, protože kvůli využívání nativních knihoven operačních systémů se tato aplikace stává platformě závislou a pro každou platformu je vyžadována jiná verze knihovny, což je nepříjemné jak pro vývojáře, kteří musí vydávat verze speciálně pro každou platformu, tak pro uživatele, kteří musí sehnat správné verze pro svá zařízení [4]. Toto rozhodnutí také zpochybňuje využití Javy jako programovacího jazyka, protože neumožňuje využít její hlavní výhodu, kterou je právě platformní nezávislost.

Argumentace v jeho práci [3], že se takto rozhodl kvůli rychlému vykreslování a přirozenému vzhledu, je podle mě velmi chabá, protože aplikace není kriticky závislá na rychlosti vykreslování GUI a na javovská okna je většina uživatelů počítače zvyklá.

1.1.4 Problém datových typů

Nástroj porovnává databázové modely na úrovni business vrstvy za pomoci abstraktního modelu, na který jsou převedeny načtené modely z datové vrstvy. Více se o porovnávání dozvíte v práci o business vrstvě [2].

Problémem ovšem je, že původní aplikace je navržena tak, že ji je sice možné rozšířit o nové zdroje, ale porovnávání datových typů se provádí podle shody názvů. Vzhledem k tomu, že různé zdroje mají také různé datové typy, které se mnohdy liší nejen v názvu, bylo by porovnávání dvou modelů z různých typů databází výše uvedeným způsobem značně omezené. Uživatel by musel manuálně kontrolovat datové typy a upravovat nefungující změnové SQL scripty.

1.2 Důsledky rozšíření

V této části jsou shrnuty hlavní dopady, které má rozšíření na původní aplikaci.

1.2.1 Možnosti porovnávání

Rozšířený nástroj uživateli v podstatě umožňuje provádět 4 typy porovnávání. Nástroj před rozšířením podporoval pouze první dvě. Následující možnosti porovnávání jsou seřazeny podle důležitosti využití v praxi.

1.2.1.1 Databáze vs. stejný typ databáze

Jedná se o porovnávání dvou skutečných databází stejného typu. Například mohu porovnat Oracle databázi ve vývojovém prostředí s Oracle databází v testovacím prostředí. Nároky na porovnání jsou zřejmé, požaduje se absolutní shoda databází.

1.2.1.2 Model v EA vs. databáze

Tato možnost nabízí porovnat reálnou databázi s databázovým modelem v EA. Opět se jedná o databáze stejného typu. V praxi tuto možnost využijeme, když vyvíjíme nějaký software používající databázi a potřebujeme udržovat model a reálnou databázi v konsistentním stavu. Za tímto účelem také tento nástroj původně vznikl. Porovnání zde probíhá velmi podobným způsobem jako u předchozí možnosti, ale je možné tolerovat jemné odchylky. Například MySQL neumožnuje pojmenovat primární klíč jinak něž PRIMARY, model nám to však umožní, proto je potřeba tuto situaci nějakým způsobem vyřešit.

1.2.1.3 Databáze vs. databáze jiného typu

S touto situací se nesetkáme tak často jako s přechozími a většinou se jedná o jednorázovou záležitost. Situace nastává hlavně v případě, že potřebujeme převést databázi na jiný typ databáze. Porovnání se proto většinou provádí proti prázdnému databázovému schématu. Zde využíváme schopnosti nástroje vytvořit změnový SQL script, který následně aplikujeme. Porovnávání se zde nemůže provádět na přesnou shodu, především kvůli různým datovým typům. Proto se musí uvažovat ekvivalentní datové typy různých databází, může také nastat situace, že ekvivalentní datový typ neexistuje. Návrhová část 2 se zabývá především tím, jak se vypořádat právě s tímto typem porovnávání, který vzniknul přidáním nových datových zdrojů.

1.2.1.4 Model v EA vs. databáze jiného typu

Tento případ nastává velmi ojediněle a nejspíše nemá praktické využití. Napadá mě pouze situace, kdy uživatel nemá k dispozici reálnou databázi (je

komerční, drahá), ale má model pro danou databázi a chce si tuto databázi rozjet na databázovém stroji jiného typu (opensource). Porovnávání je velmi podobné přechozí možnosti. Opět zde mohou navíc nastat odchylky jako v druhé možnosti.

1.2.2 Rozlišování typů databáze

V původní aplikaci nebylo potřeba rozlišovat typ databáze pro porovnávání, protože se předpokládalo, že všechny modely jsou pro databázi Oracle. Ale rozšíření potřebuje znát typ databáze, ze které byl model nahrán, především kvůli určení způsobu porovnávání, viz 1.2.1, a kvůli práci s datovými typy.

V původní aplikaci se sice rozlišovaly datové zdroje, což se zdá být stejné jako rozlišování typů databází, ale musíme si uvědomit, že EA není žádným typem reálné databáze a může obsahovat modely, všech možných typů databází, proto je nutné při nahrávání modelu z EA zjistit, pro jakou reálnou databázi byl vytvořen a s tím později pracovat.

1.2.3 Porovnávání a převody datových typů

K vyřešení problému s datovými typy 1.1.4, který je spojen se všemi důsledky rozšíření, bude třeba navrhnout nový způsob porovnávání a také převody datových typů. Proto se dále budu zabývat analýzou datových typů u původního zdroje (Oracle) a u dvou mnou přidávaných zdrojů (MySQL, PostgreSQL).

1.3 Analýza datových typů

Pro vyřešení porovnávání a převodů bylo potřeba vědět, jaké datové typy používají dotyčné databáze a jak se s nimi pracuje, proto byla tato oblast zanalyzována. Nezáleží tolik na konkrétním uložení jednotlivých datových typů, jde hlavně o jejich použití a rozsah hodnot, kterých mohou nabývat.

Podtrženy jsou hlavní názvy datových typů, následují čárkami odděleny jejich synonyma. Ve starších verzích tato synonyma mohla představovat vlastní datový typ, ale tato analýza pramenila z nejnovějších specifikací dotyčných databází.

1.3.1 Oracle

Informace byly čerpány z oficiální dokumentace Oracle pro verzi 11.1 [5, Oracle Data Types], tato verze byla v době vzniku této práce aktuální.

1.3.1.1 Znakové datové typy

Oracle může používat různé znakové sady a podporuje jednobytové a vícebytové kódování.

CHAR
Uchovává znakové řetězce fixní délky. Musí být specifikována délka od 1 až do 2000 bytů. Výchozí délka je 1. Kratší řetězce se doplní bílými znaky na plnou délku.

VARCHAR2, VARCHAR
Uchovává znakové řetězce proměnné délky. Musí být specifikována maximální délka ve znacích nebo bytech od 1 až do 4000 bytů.

NCHAR
Uchovává znakové řetězce fixní délky v Unicode. Maximum 2000 bytů. Obdoba CHAR.

NVARCHAR2
Uchovává znakové řetězce proměnné délky v Unicode. Maximum 4000 bytů. Obdoba VARCHAR2.

CLOB
Uchovává znakové řetězce proměnné délky až do 128 terabytů.

NCLOB
Uchovává znakové řetězce v Unicode. Obdoba CLOB.

LONG
Uchovává znakové řetězce proměnné délky. Maximum 2 gigabyty. Tento typ je podporován kvůli zpětné kompatibilitě. Oracle důrazně doporučuje používat místo tohoto typu CLOB nebo NCLOB.

Délka řetězce může být zadána napevno v bytech (VARCHAR2(20 BYTE)), nebo ve znacích, kde se liší podle použité znakové sady (VARCHAR2(10 CHAR)). Implicitně se používají byty.

1.3.1.2 Číselné datové typy

Uchovávají kladná, záporná, celá i desetinná čísla, nulu, nekonečno a hodnotu pro nedefinovanou operaci (not a number).

NUMBER
Číslo s pevnou čárkou nebo plovoucí čárkou. Rozsah od $-9.99...99 * 10^{125}$ do $-1 * 10^{-130}$, 0 a od $1 * 10^{-130}$ do $9.99...99 * 10^{125}$ včetně plus a mínus nekonečna. Přesnost na 38 míst. Nastavuje se celkový počet číslic a počet desetinných míst.

BINARY_FLOAT
Používá binární přesnost. Jedná se o 32-bitové číslo s plovoucí čárkou. Potřebuje 5 bytů včetně bytu pro uložení délky.

BINARY_DOUBLE
Používá binární přesnost. Jde o 64-bitové číslo s plovoucí čárkou. Potřebuje 9 bytů včetně bytu pro uložení délky.

1.3.1.3 Datum a čas

DATE
Uchovává datum a čas mezi 1.1.4712 př. n. l. a 31.12.4712 n. l.. To znamená rok, měsíc, den, hodiny, minuty a sekundy. Data se dají různě formátovat.

TIMESTAMP
Stejné jako DATE, navíc ukládá zlomky sekund. Lze zde nastavit časové pásmo (TIMESTAMP WITH (LOCAL) TIME ZONE).

1.3.1.4 Rozsáhlé datové typy

Umožňují manipulaci s velkými bloky nestrukturovaných dat v binární nebo znakové podobě. Patří sem také CLOB a NCLOB uvedené výše.

BLOB
Uchovává až 128 terabytů binárních dat.

BFILE
Data jsou uložena v souborech mimo databázi a nelze s nimy manipulovat.

RAW
Datový typ s proměnnou délkou podobně jako VARCHAR2. Uchovává binární data.

1.3.2 MySQL

Informace jsem čerpal z oficiální dokumentace MySQL pro verzi 5.7 [6, Data Types], která byla aktuální v době vzniku této práce.

MySQL nabízí vesměs základní datové typy používané většinou databázových systémů. Snaží se šetřit místem, a proto nabízí datové typy s různým rozsahem. Jiné databázové protějšky většinou nenabízí tak širokou škálu datových typů podle rozsahu.

1.3.2.1 Řetězcové datové typy

Je nutné zvolit znakovou sadu.

CHAR
Uchovává znakové řetězce fixní délky. Musí být specifikována délka od 0

až do 255. Výchozí délka je 1. Kratší řetězce se doplní bílými znaky na plnou délku.

VARCHAR
Uchovává znakové řetězce proměnné délky. Musí být specifikována maximální délka od 0 až do 65,535. Výchozí délka je 1.

BINARY
Obdoba CHAR, ale ukládá binární řetězce. Délka v bytech.

VARBINARY
Obdoba VARCHAR, ale ukládá binární řetězce. Délka v bytech.

TINYBLOB
Binární data maximální délky 255 bytů.

TINYTEXT
Znaková data maximální délky 255 bytů.

BLOB
Binární data maximální délky $(2^{16} - 1)$ bytů.

TEXT
Znaková data maximální délky $(2^{16} - 1)$ bytů.

MEDIUMBLOB
Binární data maximální délky $(2^{24} - 1)$ bytů.

MEDIUMTEXT
Znaková data maximální délky $(2^{24} - 1)$ bytů.

LONGBLOB
Binární data maximální délky $(2^{32} - 1)$ bytů.

LONGTEXT
Znaková data maximální délky $(2^{32} - 1)$ bytů.

ENUM
Položka může nabývat pouze určitých definovaných hodnot (až 65535 různých hodnot). Interně je reprezentován jako celé číslo.

SET
Položka může nabývat žádné nebo více definovaných hodnot. Interně je reprezentován jako celé číslo. 64 členů, 255 definic.

1.3.2.2 Číselné datové typy

Rozlišují se na signed a unsigned. Většina číselných datových typů může tvořit pole.

BIT
: Bitové pole o M prvcích, 1 až 64 bitů.

TINYINT
: 8-bitové celé číslo. Znaménkový rozsah -128 až 127, neznaménkový rozsah 0 až 255.

BOOL, BOOLEAN
: Synonyma pro TINYINT(1). Číslo 0 se bere jako nepravda a kladná čísla jako pravda. FALSE je mapováno na 0 a TRUE je mapováno na 1.

SMALLINT
: 16-bitové celé číslo. Znaménkový rozsah -32768 až 32767, neznaménkový rozsah 0 až 65535.

MEDIUMINT
: 24-bitové celé číslo. Znaménkový rozsah -8388608 až 8388607, neznaménkový rozsah 0 až 16777215.

INT, INTEGER
: 32-bitové celé číslo. Znaménkový rozsah - 2147483648 až 2147483647, neznaménkový rozsah 0 až 4294967295.

BIGINT
: 64-bitové celé číslo. Znaménkový rozsah - 9223372036854775808 až 9223372036854775807, neznaménkový rozsah 0 až 18446744073709551615.

DECIMAL, NUMERIC
: Desetinné číslo s pevnou desetinnou čárkou. Udává se s počtem číslic (*precision*) a počtem číslic za desetinnou čárkou (*scale*). DECIMAL(M) je ekvivalentní s DECIMAL(M, 0). Implicitní hodnota M je 10. Maximum je 65 číslic.

FLOAT
: Je 4-bytové desetitinné číslo s plovoucí čárkou. Rozsah hodnot je od -3.402823466E+38 do -1.175494351E-38, 0 a od 1.175494351E-38 do 3.402823466E+38. Lze definovat *precision* a *scale*.

DOUBLE, DOUBLE PRECISION
: Je 8-bytové desetitinné číslo s plovoucí čárkou. Rozsah hodnot je od -1.7976931348623157E+308 do -2.2250738585072014E-308, 0 a od 2.2250738585072014E-308 do 1.7976931348623157E+308. Lze definovat *precision* a *scale*.

1.3.2.3 Datum a čas

DATE
Uchovává datum. Rozsah od 1. 1. 1000 do 31. 12. 9999.

DATETIME
Uchovává datum a čas. Rozsah od 1. 1. 1000 00:00:00.000000 do 31. 12. 9999 23:59:59.999999.

TIMESTAMP
Uchovává datum a čas jako počet sekund od 1. 1. 1970 UTC. Podporuje časová pásma. Rozsah od 1. 1. 1970 00:00:01.000000 do 19. 1. 2038 03:14:07.999999.

TIME
Uchovává čas. Rozsah od -838:59:59.000000 do 838:59:59.000000.

YEAR
Rok jako 4 číslice. Rozsah od 1901 do 2155 a 0000.

1.3.3 PostgreSQL

Informace jsem čerpal z oficiální dokumentace PostgreSQL pro verzi 9.4 [7, Data Types], která byla aktuální v době vzniku této práce.

PostgreSQL nabízí širokou škálu datových typů od těch základních až po velice specifické datové typy, které se těžko mapují na datové typy z jiných databází. Uvádím zde pouze základní datové typy, se kterými bych mohl později pracovat. Mnoho datových typů zde má z historického hlediska aliasy (synonyma).

1.3.3.1 Znakové datové typy

CHARACTER, CHAR
Znakový řetězec fixní délky.

CHARACTER VARYING, VARCHAR
Znakový řetězec proměnné délky, kde uživatel definuje maximum.

TEXT
Neomezený znakový řetězec proměnné délky.

1.3.3.2 Číselné datové typy

SMALLINT, INT2
2-bytové celé číslo. Rozsah od -32768 do 32767.

INTEGER, INT, INT4
4-bytové celé číslo. Rozsah od -2147483648 do 2147483647.

BIGINT, INT8
8-bytové celé číslo. Rozsah od -9223372036854775808 do 9223372036854775807.

NUMERIC, DECIMAL
Desetinné číslo s pevnou desetinnou čárkou. Velmi přesné a pomalé výpočty. Nastavuje se velikost a počet desetinných míst.

REAL, FLOAT4
4-bytové číslo s plovoucí desetinou čárkou.

DOUBLE PRECISION, FLOAT8
8-bytové číslo s plovoucí desetinou čárkou.

SMALLLSERIAL, SERIAL2
2-bytový autoinkrement. Rozsah od 1 do 32767.

SERIAL, SERIAL4
4-bytový autoinkrement. Rozsah od 1 do 2147483647.

BIGSERIAL, SERIAL8
8-bytový autoinkrement. Rozsah od 1 do 9223372036854775807.

1.3.3.3 Datum a čas

U časů lze nastavit požadovaná přesnost. Níže uváděná přesnost je maximální možná přesnost.

TIMESTAMP
Uchovává datum a čas v 8 bytech. Možno zvolit rozlišování časových pásem. Rozsah od 4713 př. n. l. do 294276 n. l. s přesností na mikrosekundy.

DATE
Uchovává datum ve 4 bytech. Rozsah od 4713 př. n. l. do 294276 n. l. s přesností na dny.

TIME
Uchovává čas v 8 nebo 12 bytech (s časovým pásmem). Rozsah od 00:00:00 do 24:00:00 s přesností na mikrosekundy.

INTERVAL
Uchovává časový interval v 16 bytech od -178000000 let do 178000000 let s přesností na mikrosekundy.

1.3.3.4 Binární datové typy

BITEA
Bytový řetězec proměnné délky.

1.3.3.5 Bitové řetězce

BIT
Bitový řetězec pevné délky. Nedoplňuje se, hodnoty musí mít přesně stejnou délku.

BIT VARYING, VARBIT
Bitový řetězec proměnné délky se zadaným maximem. Když maximum není zadáno, znamená to neomezenou délku.

1.3.3.6 Boolean

BOOLEAN, BOOL
Obsahuje buď hodnotu TRUE, nebo FALSE, nebo UNKOWN (NULL).

1.3.4 Shrnutí

Jak lze vidět, datových typů je mnoho a k některým se těžko hledá protějšek v jiné databázi. Není zde uveden ani celý výčet datových typů analyzovaných databází. Snažil jsem se uvést pouze ty, které nejsou příliš specifické a existuje u nich šance najít podobný datový typ v jiném databázovém systému a připadají tak v úvahu pro budoucí mapování.

1.3.4.1 Nepřevoditelné datové typy

Některé datové typy jsou nepřevoditelné způsobem, který podporuje tento nástroj. Jedná se především o datové typy se složitější strukturou, které nemají žádný ekvivalent v druhé databázi a tak by bylo nutné místo jednoho sloupce vytvořit sloupců více, abychom mohli zachovat strukturu původního datového typu. Toto by ovšem přineslo velké problémy s porovnáváním, protože by se sloupce již nemohly párovat 1:1. To je důvod, proč se těmito datovými typy nebudu dále zabývat. Navíc většina uživatelů databáze si vystačí s primitivními datovými typy.

1.4 Mapování datových typů

Výsledkem průzkumu na internetu bylo nalezení velkého množství různých doporučených převodů datových typů mezi dvěma databázemi. Bohužel nebyl objeven žádný standard, kterým by se tyto převody řídily. Některé datové typy jsou definovány v SQL jako standardní, ale mnohé databáze se ani tímto neřídí. Nenašel jsem tedy žádný univerzální převod mezi všemi databázemi. Proto jsem se rozhodl, že mapování vyřeším sám na základně předchozí analýzy datových typů 1.3 a inspiruji se právě standardními datovými typy v SQL [8].

Návrh mapování datových typů probíhal zároveň s návrhem jeho implementace 2.1. Nejdříve byla snaha vytvořit univerzální mapování pro všechny

databáze. Z této myšlenky nakonec sešlo a přešlo se na systém převodů mezi dvojicemi databází. Byly vytvořeny převodní tabulky, které jsou konfigurovatelné. To znamená, že mapování datových typů bylo vyřešeno pomocí převodních tabulek, které můžete nalézt na přiloženém CD. Toto řešení však není definitivní, mnoho uživatelů nemusí s touto počáteční konfigurací souhlasit a přetvoří si tabulky tak, aby vyhovovaly jejich potřebám. Problém mapování datových typů se tedy částečně přenáší na uživatele, kteří si ho mohou sami vyřešit.

Více detailů a důvody těchto rozhodnutí se můžete dočíst v následující kapitole 2.1.

Kapitola 2

Návrh

Tato kapitola se zabývá návrhem implementace mapování datových typů.

2.1 Mapování datových typů

V této části jsou rozebrána a zhodnocena řešení mapování datových typů. Je zde také zdůvodněno, proč bylo vybráno následně implementované řešení.

2.1.1 Původní návrh - mapování pomocí tříd

Moje původní myšlenka byla taková, že budu porovnávat datové typy na základě tříd, nikoli na základě shody jména, jak tomu bylo doposud. Chtěl jsem vytvořit speciální třídu pro každý datový typ, který by byl specificky reprezentován v každé databázi. Pro lepší pochopení uvedu příklad 2.1.

Tabulka 2.1 znázorňuje, jak budou jednotlivé datové třídy reprezentovány ve specifických databázích. Jak můžeme vidět, některé datové typy jsou ve sloupci obsažené vícekrát například CLOB, ale mapování musí být jedno-

Tabulka 2.1: Třídní mapování

Třída	Oracle	MySQL	PostgreSQL
Char	CHAR	CHAR	CHAR
Varchar	VARCHAR2	VARCHAR	VARCHAR
NChar	NCHAR	CHAR	CHAR
NVarchar	NVARCHAR2	VARCHAR	VARCHAR
TinyText	CLOB	TINYTEXT	TEXT
Text	CLOB	TEXT	TEXT
MediumText	CLOB	MEDIUMTEXT	TEXT
LongText	CLOB	LONGTEXT	TEXT

značné, proto musíme zvolit, jako která třída se má datový typ načíst. Zvolíme třeba, že CLOB se bude načítat jako třída Text.

2.1.1.1 Příklad porovnání

Jako ukázku předvedu porovnání datových typů u Oracle a MySQL. Z MySQL databáze načtu datový typ MEDIUMTEXT na třídu MediumText. V Oracle databázi příslušný sloupeček neexistuje, tak ho vytvořím s datovým typem CLOB podle řádku tabulky 2.1, odpovídající třídě MediumText.

Provedu opět porovnání těchto dvou databází. Z MySQL databáze opět načtu datový typ MEDIUMTEXT na třídu MediumText, ale v Oracle databázi již příslušný sloupeček existuje, tak načtu CLOB na třídu Text, jak jsme si zvolili dříve. A vidíme, že třídy si neodpovídají i přes to, že jsme právě aplikovali změny na Oracle databázi. Databáze si nebudou odpovídat do té doby, než aplikujeme změny na MySQL databázi a porovnání se provede přes třídu Text.

2.1.1.2 Shrnutí

Tento způsob porovnávání je možný, ale nese s sebou problémy uvedené výše. Nástroj by pak mohl být použit pro převod databáze na jiný typ, ale jeho porovnávací schopnosti by byly v tomto směru omezené. Především nezkušeného uživatele by mohlo mást, proč si dvě databáze neodpovídají, přestože jsme právě aplikovali změny na jednu z databází. To je důvod, proč jsem toto řešení nakonec nepoužil.

2.1.2 Mapování pomocí převodních tabulek

Původní nápad mapovat datové typy na třídy jsem zavrhl, také jsem upustil z představy vytvořit univerzální mapování napříč všemi databázemi.

Můj předchozí průzkum mi napověděl, že snažit se vytvořit univerzální mapování napříč všemi databázemi je velice obtížný úkol, ne-li nemožný, protože jinak bych nějaký hotový způsob určitě nalezl. Proto jsem od této představy upustil a rozhodl se, že mapování budu řešit zvlášť pro všechny dvojice databází. Toto už je vyřešeno mnoha způsoby, ale já už základ svého řešení měl z předchozího návrhu.

2.1.2.1 Převodní tabulka

Převodní tabulka, nám vlastně definuje, jak bude jeden datový typ z dané databáze převeden na datový typ jiné databáze. Vzhledem k tomu, že existuje mnoho způsobů, jak si tyto převody určit, rozhodl jsem se, že tyto tabulky budou nahrávány z konfiguračních souborů, kde si je uživatel nastaví podle sebe bez nutnosti zásahu do kódu. Konkrétní strukturu převodní tabulky naleznete v sekci 3.2.4.

2.1.2.2 Princip převodů a rovností

Je zřejmé, že převodní tabulky nám určují, jak se má daný datový typ převést do jiného typu databáze. To však neřeší problém s porovnáváním datových typů. Můžeme si to představit tak, že každý řádek převodní tabulky představuje převod, ale zároveň rovnost, která právě tento problém řeší.

Konkrétní implementace převodní tabulky a příklady použití převodů a rovností se nacházení v těchto částech 3.2.4, 3.2.5.1, 3.2.6.1.

Převod
Převod zajišťuje nápravu datového typu, pokud je jiný nebo jej databáze vůbec neobsahuje. Převod musí být jednoznačný. Jednoznačnost se zaručí tím, že se stanoví jasná priorita pro výběr převodů 3.2.5.

Rovnost
Rovnost zaručuje, že nástroj identifikuje dva datové typy jako stejné, pokud převodní tabulka obsahuje řádek s odpovídající rovností/převodem. Rovnost nemusí být jednoznačná a jeden datový typ může mít více ekvivalentů v jiné databázi. Rovnosti garantují, že datový typ a převedený datový typ budou v opakovaném porovnání identifikovány jako shodné. Konkrétní práce s rovnostmi je popsána v sekci 3.2.6.

2.1.2.3 Výhody

Tento způsob dává uživateli obrovský prostor k vlastní konfiguraci převodů datových typů na míru jeho potřebám. Konfigurace je poměrně jasná a snadná. Konfigurace si poradí i s pro aplikaci neznámými datovými typy. Konfigurace umožnuje aplikaci poradit si i s datovými typy, které aplikace nezná.

Pokud aplikuji změny na jednu databázi, tak v dalším srovnání jsou datové typy díky principu rovností shodné, takže nedochází k problémům jako u původního nápadu s třídním mapováním.

2.1.2.4 Nevýhody

Aby byl uživatel naprosto spokojen, zřejmě mu nebude stačit základní konfigurace převodních tabulek a bude si ji muset upravit podle sebe. Převodních tabulek je velké množství, protože musí existovat tabulka pro každou dvojici typů databází v obou směrech. Hledáme tedy počet všech dvojic, kde záleží na pořadí. Jejich počet (p) lze v závislosti na počtu typů databází (n) vyjádřit jako:

$$p = \frac{n!}{(n-2)!} = n \cdot (n-1) = n^2 - n$$

Počet převodních tabulek s přibývajícími typy databází tedy roste kvadraticky. Aktuálně jich je 6.

Kapitola 3

Řešení

Po dohodě s vedoucím práce jsem se rozhodl nejdříve rozšířit stávající aplikaci o nové zdroje MySQL a PostgreSQL bez řešení otázky datových typů. K tomuto rozhodnutí mě vedlo několik důvodů. Především jsem se potřeboval hlouběji seznámit s aplikací mých předchůdců a řádně se v ní zorientovat, abych mohl navrhnout ideální řešení otázky datových typů. Dále jsem chtěl vidět, jak budou vypadat konflikty mezi datovými typy různých databází, ke kterým bude nejspíše v omezené míře docházet i po úspěšné implementaci vzájemných převodů.

3.1 Implemetace MySQL a PostgreSQL

Ač byla původní aplikace prezentována jako snadno rozšiřitelná, bylo docela obtížné úspěšně přidat nové datové zdroje. Paradoxně se musela měnit velká část prezentační vrstvy, která by měla být na zdrojích nezávislá. V business vrstvě se musela také udělat menší úprava. Datová vrstva byla hodně zaměřená na Oracle databázi, takže jsem se ji snažil zobecnit a zapojit více dědičnost a polymorfismus, bez kterého by byl téměř stejný kód několikrát kopírován.

3.1.1 Úprava prezentační vrstvy

Původě jsem se domníval, že v rámci tohoto úkolu se nebude do této vrstvy vůbec zasahovat, jen se přidá pár obrázků jako loga. Nakonec byl ale přeci zásah nutný. Bylo potřeba přidat pouze pár drobností, ale najít místa, kde všude bylo nutno nové zdroje přidávat, byl velmi obtížný a zdlouhavý úkol. Proto jsem pamatoval na mé případné další nástupce a v krizových místech (jedná se především o rozhodování na základě datového zdroje), kde je potřeba dopsat další funkčnost v případě přidání dalšího zdroje, byl kód doplněn o vyhazování patřičných vyjímek (`UnsupportedOperationException`).

3.1.1.1 Přizpůsobení GUI novým zdrojům

Jelikož se nové zdroje liší od původního Oracle, bylo potřeba přetvořit okno připojení vyhovující novým zdrojům. Téměř každá databáze má jinou hierarchii a terminologii uspořádání databází a schémat.

Všechny databáze vyžadují pro připojení k databázi tyto údaje: server a port, na kterém databáze běží, uživatelské jméno a heslo. Rovněž bylo žádoucí, aby GUI umožnovalo zadat prázdné heslo. Protože implicitní administrátor MySQL databáze je uživatel „root“ bez hesla. Specifické požadavky databází jsou uvedeny níže.

Oracle
Oracle vyžaduje specifikovat název databáze (SID) a název schématu, neboť jedno SID v sobě může obsahovat více schémat. Název schématu se v původní aplikaci nepoužíval, předpokládalo se, že bude shodný s uživatelským jménem. Z tohoto důvodu je název schématu odvozen z uživatelského jména, pokud uživatel tuto kolonku nevyplní.

PostgreSQL
PostgreSQL, podobně jako Oracle, používá název databáze a název schématu. Pokud uživatel název schématu nevyplní, vyplní se hodnota „public“, to je výchozí název schématu.

MySQL
MySQL na rozdíl od dvou předchozích případů používá pouze název databáze. Databáze tedy obsahuje pouze jedno schéma.

3.1.2 Implementace datové vrstvy

Při implementaci datové vrstvy jsem vycházel z původních zdrojových kódů pro Oracle databázi. Bylo potřeba si dávat pozor na drobné odlišnosti různých databází.

3.1.2.1 JDBC vs. SQL dotazy

Načtení modelu z databáze bylo možné udělat dvěma různými způsoby.

Použití JDBC je univerzální a lze předpokládat, že řešení bude totožné pro všechny databáze. V práci [1] se dočteme, že použití JDBC značně prodloužilo načítání integritních omezení z Oracle databáze plné dat, protože spouštělo statistiky. To ale neznamená, že tento problém nastane i u MySQL a PostgreSQL. Proto jsem se snažil maximálně využít možnosti JDBC, abych nemusel každou databázi řešit zvlášť.

Použití přímých SQL dotazů je z mého pohledu elegantnější, ale SQL dotazy jsou silně vázány na použitý typ databáze. Proto jsem tuto metodu použil

pouze tam, kde mi JDBC nebylo schopné dát požadované výsledky, jako třeba načtení pohledů s tabulkami, kterých se pohled týká.

3.1.2.2 Načítání pohledů

Jak bylo uvedeno výše, načítaní pohledů se provádí pomocí SQL dotazů nad informačním schématem u obou nových databází. Pro účely nástroje se potřebuje získat název pohledu a názvy všech tabulek, kterých se daný pohled týká.

PostgreSQL
Zde bylo požadovaného výsledku dosaženo docela snadno. Informační schéma v PostgreSQL totiž obsahuje tabulku `view_table_usage` speciálně pro tyto účely. Tento způsob je v podstatě totožný s použitým řešením u Oracle databáze.

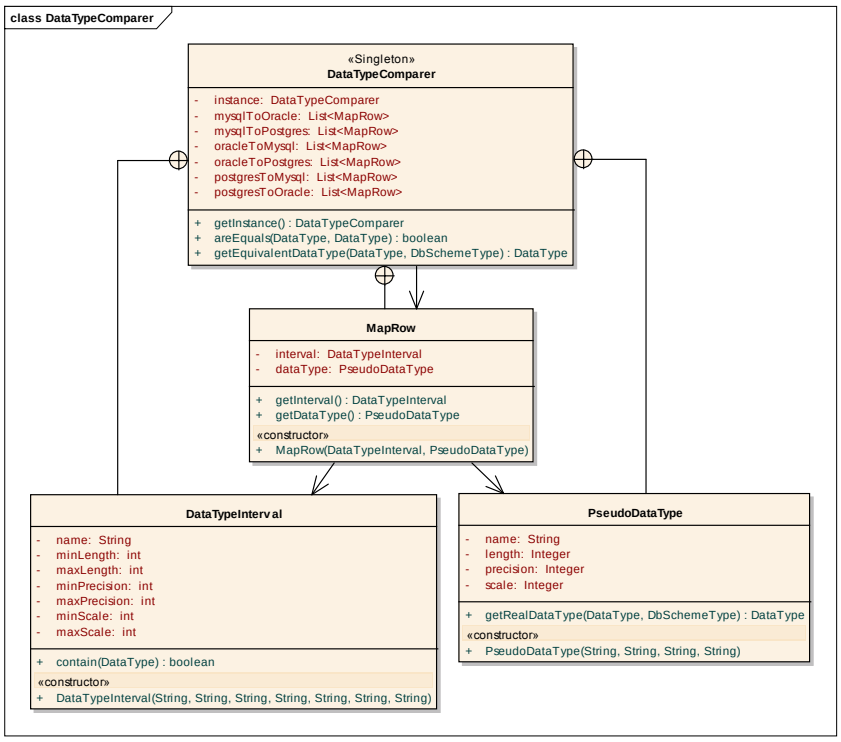
MySQL
Bohužel u MySQL žádná podobná tabulka neexistuje. Je zde ale jiná tabulka `views`, která obsahuje celou definici pohledu. Pomocí SQL dotazu se tedy získá název a definice pohledu. Z definice pak musí získat názvy tabulek, kterých se pohled týká. To zajišťuje použití regulárního výrazu. Dlouho jsem hledal vhodnější způsob, jak toto provádět, bohužel jsem pro MySQL nic lepšího nenašel, ani nevymyslel.

3.1.2.3 Změna datového typu u PostgreSQL

U databáze PostgreSQL byl identifikován problém při pokusu o změnu datového typu pomocí SQL scriptu. PostgreSQL totiž dovoluje změnit datový typ sloupce pouze v případě, že dokáže konvertovat data v tomto sloupci na nový datový typ. Tento problém bohužel nastává i v situaci, kdy databáze neobsahuje žádná data.

Změna datového typu pro databázi PostgreSQL byla vyřešena podobným SQL scriptem, který používá Oracle databáze. Nejdříve se vytvoří sloupec s novým datovým typem a dočasným názvem. Do tohoto sloupce se zkopírují data z původního sloupce. Původní sloupec se zruší a nový sloupec se přejmenuje na název původního sloupce.

Ani toto však u PostgreSQL databáze neprojde, pokud databáze nedokáže data konvertovat na nový datový typ. Aby script fungoval i za těchto okolností, musí z něj uživatel vyřadit kopírování dat (ztratí data) nebo musí určit, jak se mají data konvertovat. Právě z těchto důvodů autor nedoporučuje aplikovat vygenerovaný SQL script přímo z nástroje dbCompare. Lepší je aplikovat tento script nástrojem, který je k tomuto účelu určen přímo výrobcem databáze a který vás většinou upozorní na případné nedostatky v SQL scriptu.



Obrázek 3.1: DataTypeComparer

3.2 Implemetace mezidatabázového provnávání datových typů

3.2.1 Datový typ

Třída `DataType` sdružuje vlastnosti datového typu (název, *length*, *precision*, *scale*) a především obsahuje typ databáze, ze které pochází.

Tato třída se stará o náhradu aliasů a přenastavování nesmyslných hodnot u datových typů, které tyto hodnoty nepoužívají. Také umí vracet vlastní SQL definici pro generování změnových scriptů.

3.2.2 DataTypeComparer

Třída `DataTypeComparer` se stará o porovnávání a převody datových typů. Načítá tedy převodní tabulky. Třída je realizována jako návrhový vzor *Singleton* s lazy inicializací [9]. To znamená, že třída má v programu maximálně jednu instanci, která se vytvoří až v momentě, kdy tuto třídu potřebujeme. Toto řešení jsem zvolil proto, aby program nemusel načítat převodní tabulky, když k žádnému porovnání ani převodům datových typů nedojde.

Diagram třídy `DataTypeComparer` včetně vnitřních tříd je na obrázku 3.1.

3.2.2.1 DataTypeInterval

`DataTypeInterval` je vnitřní třídou třídy `DataTypeComparer`. Jak už název napovídá, umožňuje ukládat datový typ jako název a intervaly jeho atributů. Využívá se pro vnitřní reprezentaci převodní tabulky jako klíč, podle kterého se hledá datový typ, na který má být převeden vstupní datový typ.

3.2.2.2 PseudoDataType

`PseudoDataType` je vnitřní třídou třídy `DataTypeComparer`. Jeho struktura je velmi podobná struktuře třídy `DataType`, ale navíc mohou být hodnoty atributů nespecifikované (`null`) a neobsahuje typ databáze. Využívá se pro vnitřní reprezentaci převodní tabulky jako datový typ, na který má být převeden vstupní datový typ.

3.2.3 Definice vs intervaly

Původně bylo v plánu převodní tabulky nahrát do hashovací tabulky, kde by se převedený typ získával v konstantním čase. To by ovšem vyžadovalo jasnou definici datového typu v převodní tabulce. Kvůli tomu, že datový typ může nabývat mnoha různých hodnot v *length*, *precision* a *scale* a je nesmyslné vyplňovat tabulku pro všechny kombinace hodnot, by nástroj musel postupně předefinovávat daný datový typ na datový typ s nespecifikovanými hodnotami. Tento způsob bohužel neumožní pracovat s intervaly těchto hodnot, což je škoda, protože v některých případech je práce s intervaly výhodná. Například, když má první databáze jeden datový typ s definovatelným rozsahem a druhá databáze spoustu datových typů s různým pevným rozsahem.

Proto byla na závěr zvolena varianta využívající intervaly. Je to sice přidání práce uživateli, který si chce převody sám nakonfigurovat, ale za to mu struktura převodní tabulky nabízí mnoho možností, o které by byl jinak ochuzen. Toto řešení bohužel znemožňuje využití hashovací tabulky, protože ze zadaného datového typu nezjistíme přesný interval. Mohli bychom, tento interval sice postupně rozšiřovat, ale toto řešení by bylo velmi neefektivní a použití hashovací tabulky by ztrácelo smysl.

Proto je převodní tabulka, v programu reprezentována jako seznam dvojic, který se postupně prohledává. Hledá se, kterému datovému typu odpovídá ten zadaný a do kterých intervalů patří. Z hlediska efektivity není absence hashovací tabulky nijak zásadní, protože předpokládaná délka převodní tabulky se pohybuje mezi 20 až 100 řádky.

Díky kontrole intervalů je možné ošetřit případy, kdy mají různé databáze jiná omezení na rozsahy ekvivalentních datových typů.

Tabulka 3.1: Ukázka převodní tabulky MySQL→Oracle

MySQL data types	min length	max length	min precision	max precision	min scale	max scale	Oracle data types	length	precision	scale
TINYINT	min	max	min	max	min	max	NUMBER	0	3	0
SMALLINT	min	max	min	max	min	max	NUMBER	0	5	0
MEDIUMINT	min	max	min	max	min	max	NUMBER	0	8	0
INT	min	max	min	max	min	max	NUMBER	0	10	0
BIGINT	min	max	min	max	min	max	NUMBER	0	19	0
DECIMAL	min	max	min	max	min	max	NUMBER	0	x	x

3.2.4 Struktura převodní tabulky

Převodní tabulka 3.1 se skládá z jedenácti sloupců a různého počtu řádků. První řádek je záhlaví, které uživateli říká, co který sloupec má obsahovat. V programu je záhlaví ignorováno a načítá se až druhý řádek. Tabulka je uložena ve formátu CSV s hodnotami oddělenými středníkem pro snadné načítání tabulek. Kvůli metodě zpracování řádku nenechávejte v tabulce nevyplněné hodnoty.

3.2.4.1 Vstupní datový typ

Prvních sedm hodnot na řádku definuje vstupní datový typ pomocí intervalů. Je z nich vytvořena třída `DataTypeInterval`. První hodnota obsahuje název datového typu napsaný verzálkami (velkými písmeny). Zbylé hodnoty definují rozsah intervalů pro parametry datových typů v pořadí *lenght*, *precision* a *scale*. Hodnoty pro intervaly se zadávají číselně. Sloupec s dolní mezí intervalu může navíc obsahovat speciální hodnotu `min` a sloupec s horní mezí intervalu speciální hodnotu `max`, což je považováno za minimum respektive maximum. Interval se považuje za uzavřený, tudíž do intervalu spadají i jeho meze.

3.2.4.2 Výstupní datový typ

Poslední čtyři hodnoty na řádku definují převod datového typu, je z nich vytvořena třída `PseudoDataType`. První z těchto hodnot obsahuje buď název datového typu, na který se má provádět mapování, nebo speciální hodnotu „x“, která značí, že k datovému typu jsme nevybrali ekvivalent. Řádek s hodnotou „x“ je tedy v tabulce zbytečný, ale usnadní nám orientaci v datových typech. Hodnoty zbylých tří sloupců obsahují buď přesné číselné hodnoty, nebo speciální hodnotu „x“, která říká, že hodnota se zkopíruje ze vstupního datového typu.

3.2.5 Výběr převodu

Nejvyšší prioritu mají převody na začátku tabulky. Priorita postupně klesá. To pro výběr převodu znamená, že se použije první převod z převodní tabulky, kde jsou splněny podmínky pro vstupní datový typ. Tento způsob je srozumi-

Tabulka 3.2: Ukázka převodní tabulky Oracle→MySQL

Oracle data types	min length	max length	min precision	max precision	min scale	max scale	MySQL data types	length	precision	scale
NUMBER	min	max	0	2	0	0	TINYINT	0	0	0
NUMBER	min	max	3	4	0	0	SMALLINT	0	0	0
NUMBER	min	max	5	6	0	0	MEDIUMINT	0	0	0
NUMBER	min	max	7	9	0	0	INT	0	0	0
NUMBER	min	max	10	18	0	0	BIGINT	0	0	0
NUMBER	min	max	min	max	min	max	DECIMAL	0	x	x

telný a jednoduchý na implementaci a efektivní, protože se většinou nemusí prohledávat celá tabulka.

Pro uživatele to znamená, aby upřednostňované převody s vyšší prioritou umisťoval do čela tabulky, především pokud se jedná o užší intervaly. To můžeme vidět v tabulce 3.2.

Pokud není nalezen vhodný převod, nástroj se pokusí použít původní datový typ ze zdrojové databáze.

3.2.5.1 Příklad převodu

MySQL→Oracle
Chceme převést datový typ `INT` z MySQL databáze do Oracle databáze. Použije se tedy tabulka 3.1, která se začne prohledávat popořadě od začátku. Nejprve se hledá shoda názvu datového typu, na shodu se narazí až na čtvrtém řádku. Dále se zkoumá, zda jednotlivé atributy datového typu vyhovují definovaným intervalům. Protože jsou zde definovány maximální intervaly, nemusí nás zajímat konkrétní hodnoty atributů a převod použijeme a ukončíme hledání. Datový typ `INT` se tedy převede na Oracle datový typ `NUMBER` s hodnotami atributů ($length = 0$, $precision = 10$, $scale = 0$).

Oracle→MySQL
Chceme převést datový typ `NUMBER` s hodnotami atributů ($length = 0$, $precision = 10$, $scale = 0$) z Oracle databáze zpátky do MySQL databáze (pozor, nejedná se o porovnávání, ale převod). Použije se tedy tabulka 3.2, která se začne prohledávat. Už na prvním řádku se narazí na shodu jmen datových typů, ale převod se nepoužije, protože atribut *precision* nespadá do definovaného intervalu ($10 \notin \langle 0; 2 \rangle$). Ze stejného důvodu se nepoužijí další tři převody. Nakonec se tedy použije převod z pátého řádku, kde už interval vyhovuje ($10 \in \langle 10; 18 \rangle$). Daný datový typ se tedy převede na MySQL datový typ BIGINT s hodnotami atributů ($length = 0$, $precision = 0$, $scale = 0$).

Čtenář by nejspíše očekával, že dostane zpět původní datový typ z předchozího příkladu, ale převody z jedné strany nemusí vůbec odpovídat převodům z druhé strany, protože o porovnávání se starají rovnosti.

3.2.6 Práce s rovnostmi

Rovnosti se na rozdíl od převodů hledají v obou tabulkách (MySQL->Oracle, Oracle->MySQL) a stačí, když je požadovaná rovnost alespoň v jedné z nich. Proto není vůbec nutné mít odpovídající převody v obou převodních tabulkách.

Zjišťování, zda jsou dva datové typy shodné, se provádí tak, že se nejprve vezme první datový typ a v příslušné tabulce se provedou všechny odpovídající převody. Tím se získá seznam ekvivalentů prvního datového typu. Pokud je nějaký ekvivalent shodný s druhým datovým typem, označí se datové typy jako shodné, pokud ne, opakuje se stejný postup získávání ekvivalentů pro druhý datový typ a porovnávání s prvním datovým typem. Pokud ani zde nebyla shoda, označí se datové typy jako rozdílné a později provedeme převod.

Pokud chce uživatel vytvořit rovnost, ale nechce takto provádět převod, stačí, když umístí příslušný převod/rovnost za převody, které již pokrývají celý jeho definovaný interval. Zpravidla stačí umístit převod na konec převodní tabulky, zvláště pokud je definovaný převod pro maximální interval.

3.2.6.1 Příklad rovnosti

MySQL=Oracle
Chceme zjistit, zda se datový typ `MEDIUMINT` (*length* = 0, *precision* = 0, *scale* = 0) z MySQL databáze rovná datovému typu `NUMBER` (*length* = 0, *precision* = 6, *scale* = 0) z Oracle databáze. Nejdříve se použije tabulka 3.1 a uskuteční se všechny vyhovující převody. Tím se získá seznam „ekvivalentů“. V tomto případě jím je pouze `NUMBER` (*length* = 0, *precision* = 8, *scale* = 0). Provede se porovnání s Oracle datovým typem a zjistí se, že se datové typy nerovnají kvůli rozdílné *precision*. Tím to ale nekončí, na řadu jde druhá tabulka 3.2, ze které získáme `MEDIUMINT` (*length* = 0, *precision* = 0, *scale* = 0) a `DECIMAL` (*length* = 0, *precision* = 6, *scale* = 0). Při porovnání s MySQL datovým typem se narazí na shodu. Rozhodne se tedy, že uvedené dva datové typy se „rovnají“ a není je potřeba nijak nahrazovat.

MySQL≠Oracle
Chceme zjistit, zda se datový typ `BIGINT` (*length* = 0, *precision* = 0, *scale* = 0) z MySQL databáze rovná datovému typu `NUMBER` (*length* = 0, *precision* = 20, *scale* = 0) z Oracle databáze. Z první tabulky 3.1 se získá `NUMBER` (*length* = 0, *precision* = 19, *scale* = 0), to nevyhovuje kvůli rozdílné *precision*. Z druhé tabulky 3.2 se získá `DECIMAL` (*length* = 0, *precision* = 20, *scale* = 0), což nevyhovuje kvůli rozdílným názvům.

3.2.7 Ukládání nalezených výsledků

Jelikož vyhledávání v převodních tabulkách může být pomalé, zvláště pokud jsou tabulky dlouhé a dotazování probíhá často, rozhodl jsem se ukládat na-

lezené výsledky provedeného hledání. Vedl mě k tomu předpoklad, že uživatel databáze používá pouze jistou část datových typů povětšinou se stejnými parametry. Navíc vyhledávání probíhá většinou dvakrát, jednou pro získání ekvivalentů pro porovnávání a podruhé pro získání převodního ekvivalentu.

Paměť (hashovací tabulka) je na začátku prázdná. Pokud vznikne požadavek na vrácení ekvivalentů konkrétního datového typu, nejprve se provede pokus o načtení této informace z hashovací tabulky, pokud záznam ještě neexistuje, znamená to, že konkrétní vyhledávání ještě neproběhlo a ekvivalenty se načtou z převodní tabulky a následně se uloží do hashovací tabulky.

Na rozdíl od případu uvedeného výše 3.2.3 zde je možné hashovací tabulku použít, neboť je znám konkrétní datový typ, pro který se provádí vyhledávání. Struktura hashovací tabulky je jednoduchá. Jako klíč je použita dvojice konkrétní datový typ a typ cílové databáze a jako hodnota se přiřadí nalezený seznam ekvivalentů tohoto datového typu.

Převodní ekvivalent je první v seznamu ekvivalentů, protože odpovídá prvnímu nalezenému převodu viz 3.2.5. Z tohoto důvodu je možné sloučit vyhledávání převodního ekvivalentu s vyhledáváním všech ekvivalentů a provádět pouze vyhledávání všech, což umožní ukládat nalezené výsledky v požadované formě v obou případech.

3.3 Změny spojené s EA

Zde jsou shrnuty změny týkající se práce s EA, které byly provedeny. Informace pro provedení těchto změn byly čerpány z příručky [10].

Aby nástroj mohl správně převádět datové typy, musí znát typ databáze modelu. Bohužel model databáze v EA tuto informaci neobsahuje a musí se tedy načíst z první tabulky modelu. To s sebou nese riziko pro porovnávání, když model obsahuje tabulky více typů databází (ve správném modelu by se toto nemělo vyskytovat) nebo EA soubor obsahuje více modelů databází různých typů. Z tohoto důvodu je nutné pro správný běh programu, vyplňovat typy databází a v případě většího počtu modelů specifikovat cestu k jednomu konkrétnímu.

Dále bylo potřeba zajistit, aby nově vzniklým elementům byl správně nastaven typ databáze, protože původní aplikace všude vyplňovala typ databáze Oracle.

Další nepatrnou změnou bylo načítání primárních klíčů pro MySQL, kde se název primárního klíče z modelu ignoruje a použije se název `PRIMARY`, protože MySQL neumožnuje primární klíče pojmenovat jinak. Kdyby tato změna nebyla provedena, tak by nástroj neustále identifikoval rozdíl mezi modely bez ohledu na to, kolikrát by uživatel aplikoval změnový script na MySQL databázi.

Kapitola 4

Refaktoring a oprava chyb v původní aplikaci

4.1 Refaktoring

„Refaktorování je proces provádění změn v softwarovém systému takový způsobem, že nemají vliv na vnější chování kódu, ale vylepšují jeho vnitřní strukturu. Je to disciplinovaný způsob pročišťování kódu s minimálním rizikem vnášení chyb.“[11]

Vzhledem k tomu, že se mi do rukou dostala první verze této aplikace, která byla s velkou pravděpodobností dokončována ve spěchu a kterou tvořil tým tří lidí, v kódu se ve značné míře vyskytovaly programátorské prohřešky. Nedostatky, které jsem nalezl, jsem se snažil napravit. Jednalo se hlavně o často se opakující kód, nevyužívané metody a proměnné. Systém vyhazování výjimek byl zřejmě připraven na produkční nasazení a nepočítalo se s dalším vývojem, proto se neočekávané výjimky často ztrácely a bylo těžké zjistit, co se vlastně stalo. Zda program přestal pracovat z důvodu chyby či ne. Jak jsem již uvedl výše, také jsem upravil kód tak, abych usnadnil jeho další vývoj. Toho jsem docílil například nastavením defaultního chování pro neznámé typy databází na vyhození výjimky, která upozorní vývojáře na zatím nepodporované chování, které musí doplnit.

4.1.1 Generování SQL scriptů

Velký refakroring by si také zasloužilo generování SQL scriptů, kde se nejspíše kvůli špatnému návrhu aplikace velmi často používá dotazování na typ třídy a následné přetypování, což odporuje principům objektově orientovaného programování [9]. Tento problém se týká především tříd, které reprezentují změnu schématu. Zásah by vyžadoval velké změny v návrhu a nejspíše by se musela měnit podstatná část aplikace, proto jsem si na tento úkol netroufl vzhledem k mým časovým možnostem.

Jedno z možných řešení, které nevyžaduje změnu celého návrhu, spočívá v tom, že by třída reprezentující změnu sama generovala SQL script pro provedení této změny nebo by to prováděla třída reprezentující element, kterého se změna týká, a změna by byla aplikovatelná na tento element. Musel by se tedy změnit přístup k reprezentaci databázového schématu, kde jsou všechny třídy pouhými držiteli informací, ale neumí s nimi pracovat.

4.2 Oprava chyb

Aplikace byla poměrně dobře otestována, takže jsem v ní moc závažných chyb nenalezl. Jednalo se spíše o opravu komentářů. Mezi závažnější chyby patří následující.

Špatně fungující SQL scripty pro Oracle, kde se při změně datového typu vytváří nový sloupec, do kterého se mají překopírovat data ze starého. Chyba spočívala v tom, že se data kopírovala obráceně, tedy z prázdného sloupce do sloupce plného, který se následně zrušil. Tato chyba je v produkční databázi velmi závažná, protože ztratíme důležitá data.

Původní aplikace se připojovala k databázi stejného jména, jako byl uživatel, což znemožňovalo připojit se k databázi pod jiným uživatelem, například administrátorem. Nyní se název schématu může zadat v GUI, pokud se nechá prázdné, aplikace se připojí k uživatelské databázi.

Kapitola 5

Testování

5.1 Automatizované testování

Automatizované testování této aplikace je poněkud obtížné, protože ve většině případů vyžaduje připojení k databázi. Je možné sice vytvořit zástupné objekty, které budou simulovat chování databáze, ale dalo by to mnoho práce a výsledný užitek by byl velmi malý.

5.1.1 Unit testy

Unit testy se zaměřují především na testování jednotlivých tříd a metod. K testování byl použit framework JUnit, který je určený k psaní unit testů pro Javu.

Toto rozšíření se týkalo především datové vrstvy, kde se jednalo hlavně o načítání modelu a generování SQL scriptů, popřípadě o velice primitivní metody, které není potřeba testovat. Jak už jsem uvedl výše, k načítání modelu bych potřeboval připojení k databázi. Testovat, zda je vygenerovaný SQL script shodný s předpokládaným výstupem na základě shodnosti textových řetězců, je velice omezený a špatně udržovatelný test, proto jsem unit testy vytvořil pouze pro třídy, kde se nepracuje s databází a negenerují se žádné scripty a zároveň je test přínosný (není primitivní). Otestovány byly hlavně nové třídy `DataType` a `DataTypeComparer`.

5.1.1.1 Údržba testů

Tato aplikace již obsahovala několik sad JUnit testů, které bylo po mém zásahu potřeba opravit a udržet je tak nadále funkční. Bohužel hlavně testy na generování SQL dotazů jsou špatně udržitelné a neotestují toho mnoho, proto by bylo lepší provádět tyto testy SQL scriptů manuálně.

5.2 Manuální testování

Aplikace byla nejvíce testována manuálně, protože je to nejjednodušší způsob testování a často bylo potřeba otestovat jednu specifickou věc v průběhu vývoje. Dále byly provedeny následující testovací scénáře pro finální verzi aplikace.

5.2.1 Testovací scénáře

Aby mohl být nástroj otestován, bylo potřeba nejprve připravit testovací modely jak pro EA, tak pro skutečné databáze. Nástroj byl mimo jiné podroben následujícím testům.

5.2.1.1 Shodnost modelů

Nejjednodušší test. Pokud se načte totožný model, nástroj nahlásí, že modely jsou stejné. Podobný test je ten, kdy je vytvořen model v EA, vygeneruje se create script, který je následně spuštěn na reálné databázi. Opět by oba modely měly být stejné.

5.2.1.2 Různost modelů

Do porovnávání vstupují dva rozdílné modely, získané tak, že jeden ze dvou shodných modelů se upraví. Nástroj objeví změny, které byly provedeny a nenahlásí shodnost modelů.

5.2.1.3 Aplikace změn

Do porovnávání vstupují dva rozdílné modely. Vybere se možnost aplikace změn na jeden model. Při opakovaném porovnání nástroj zahlásí, že modely jsou shodné. Tento testovací scénář nemusí procházet vždy, protože existují změny, které nástroj neumí aplikovat, například přidání pohledů a převod složitých datových typů 1.3.4.1.

5.2.2 Uživatelské testování

Uživatelské testování znamená, že aplikaci testují samy uživatelé. Výhoda tohoto testování spočívá v tom, že tester je nezaujatý člověk a hlásí tak chyby i připomínky, které nejsou závažné, ale zpříjemňují uživatelům práci s aplikací.

Toto testování prováděl vedoucí práce Ing. Jiří Mlejnek, který poctivě reportoval chyby a navrhoval vylepšení nástroje. Tyto vylepšení se týkaly především GUI.

5.3 Zhodnocení testování

Testování odhalilo mnoho chyb v aplikaci. Většina z nich souvisela s generováním SQL scriptů. Díky testování se také narazilo na různé problémy, které by bez otestování zůstaly skryty (např. 3.1.2.3). Všechny identifikované chyby byly opraveny. Nástroj byl testován poměrně komplexně (zkoušely se téměř všechny varianty změn a kombinace databází), tudíž by finální verze již neměla obsahovat žádné závažné chyby.

Unit testy by měly v průběhu dalšího vývoje poskytovat zpětnou vazbu, zda se do programu nezanesla chyba. Aby tyto testy byly užitečné, musí se průběžně spouštět a udržovat.

Závěr

Cílem této práce bylo zanalyzovat a rozšířit nástroj pro porovnávání databázových modelů dbCompare o nové datové zdroje, kterými byly databáze MySQL a PostgreSQL.

Po počáteční analýze nástroje a analýze datových typů jednotlivých databází jsem se pustil do implementace přímého rozšíření o nové datové zdroje na základě nabízených rozhraní. V této fázi jsem ještě nevyřešil problém s datovými typy, ale v průběhu implementace jsem získal dostatek informací na to, abych mohl navrhnout řešení tohoto problému. Z počátku jsem se snažil navrhnout univerzální mapování datových typů napříč všemi databázemi, tento způsob však nebyl příliš vhodný, protože by musel tolerovat mnoho kompromisů. Nakonec jsem tento problém vyřešil pomocí převodů datových typů mezi databázemi. Výsledkem této práce tedy není pouze tento text a výsledná aplikace, ale také šest převodních tabulek v základní konfiguraci, které určují mezidatabázové převody datových typů.

Cíl práce jsem tedy splnil. Nástroj nyní umí pracovat s Enterprise Architektem, Oracle, MySQL a PostgreSQL databází a umí převádět většinu datových typů, což umožnuje použít nástroj na převedení databázového modelu na jiný typ databázového modelu. Na závěr jsem aplikaci otestoval a myslím si, že jsem odstranil většinu chyb, které se v aplikaci vyskytovaly, takže nástroj mohou začít používat uživatelé.

Hlavním důvodem tohoto rozšíření bylo, aby uživatelé nástroje mohli pracovat s jinými modely než Oracle, což se povedlo velmi brzy. Možnost porovnávat modely různých typů je tedy takovým bonusem a je až sekundárním účelem aplikace. Primárním účelem je samozřejmě synchronizace modelů v různých prostředích v průběhu vývoje.

Použité zdroje

[1] Effenberger, J.: *Datová vrstva nástroje pro synchronizaci struktur databázových modelů*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2013.

[2] Novák, M.: *Business vrstva nástroje pro synchronizaci struktur databázových modelů*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2013.

[3] Kála, O.: *Prezentační vrstva nástroje pro synchronizaci struktur databázových modelů*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2013.

[4] Schroeder, A.; Veys, N.; Laffra, C.: FAQ Is SWT better than Swing? [online]. 2010, [cit. 2015-5-7]. Dostupné z: `http://wiki.eclipse.org/FAQ_Is_SWT_better_than_Swing%3F`

[5] Oracle Corporation: Oracle Database Online Documentation 11g Release 1 (11.1) [online]. [cit. 2014-12-2]. Dostupné z: `http://docs.oracle.com/cd/B28359_01/index.htm`

[6] Oracle Corporation: MySQL 5.7 Reference Manual [online]. [cit. 2015-2-8]. Dostupné z: `http://dev.mysql.com/doc/refman/5.7/en/index.html`

[7] PostgreSQL Global Development Group: PostgreSQL 9.4.1 Documentation [online]. [cit. 2015-2-10]. Dostupné z: `http://www.postgresql.org/docs/9.4/static/index.html`

[8] W3SCHOOLS: SQL General Data Types [online]. [cit. 2015-4-17]. Dostupné z: `http://www.w3schools.com/sql/sql_datatypes_general.asp`

[9] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994,

ISBN 9780321700698. Dostupné z: http://books.google.cz/books?id=6oHuKQe3TjQC

[10] Sparx Systems Pty Ltd: Enterprise Architect Software Developers' Kit. 2010. Dostupné z: http://www.sparxsystems.com.au/downloads/resources/booklets/enterprise_architect_sdk.pdf

[11] Fowler, M.; Beck, K.: *Refaktoring: zlepšení existujícího kódu*. Moderní programování, Grada Publishing, 2003, ISBN 9788024702995. Dostupné z: https://books.google.cz/books?id=hsvzc-XE3nYC

[12] Oracle Corporation: Java Platform Standard Edition 7 Documentation [online]. Dostupné z: http://docs.oracle.com/javase/7/docs/

[13] Eclipse Foundation: Eclipse Platform 4.4.1 [software]. Dostupné z: http://www.eclipse.org/downloads

[14] Effenberger, J.; Kála, O.; Novák, M.: dbCompare 1.0 [software].

[15] Sparks System: Enterprise Architect 8.0.864 [software]. Dostupné z: http://www.sparxsystems.com.au/

[16] Hamano, J.; Torvalds, L.: Git 1.9.5 [software]. Dostupné z: http://git-scm.com/downloads

[17] Oracle Corporation: Oracle Database11.1 [software]. Dostupné z: http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html

[18] Oracle Corporation: MySQL 5.7 [software]. Dostupné z: http://dev.mysql.com/downloads/

[19] PostgreSQL Global Development Group: PostgreSQL 9.4.1 [software]. Dostupné z: http://www.postgresql.org/download/

[20] Beck, K.; Gamma, E.; Saff, D.; aj.: JUnit 4.11 [framework]. Dostupné z: https://github.com/junit-team/junit/wiki/Download-and-Install

Příloha **A**

Seznam použitých zkratek

CSV Comma-separated values

EA Enterprise Architect

GUI Graphical user interface

JDBC Java Database Connectivity

SID System Identifier

SQL Structured Query Language

SWT Standard Widget Toolkit

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc` ..... adresář obsahující JavaDoc dokumentaci
- `exe` ..... adresář se spustitelnou formou implementace
- `map-tables` ..... adresář s převodními tabulkami
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `dbcompare` ..... Eclipse projekt
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF