České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídící techniky

Bakalářská práce

Automatizovaná aktualizace struktury a dat relační databáze dle doménového modelu podnikové aplikace

Ondřej Fremunt

Vedoucí práce: Ing. Michal Valenta, Ph.D.

Studijní program: Elektrotechnika a informatika, Bakalářský

Obor: Kybernetika a měření

27.5.2011

Poděkování

Chtěl bych poděkovat svému vedoucímu práce panu Ing. Michalu Valentovi, Ph.D. za ochotu a trpělivost, kterou se mnou měl během tvorby bakalářské práce. Dále bych chtěl poděkovat máce, tátovi, Makovi, babičkám a dědovi za obrovskou podporu, kterou mi projevili. Velký dík patří nejbližším kamarádům za odreagování na vandrech, bez kterého bych tuto práci asi nikdy nedokončil, Goofymu za zasvěcení do problematiky generativního programování a obětavou pomoc zejména při závěrečných úpravách práce a jejím sázení v DocBooku a dále Bedřichovi a Pluťákovi, kteří se kvůli mně před odevzdáním práce moc nevyspali.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu. Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27.5.2011

Abstrakt

Údržba a rozvoj informačních systémů zpravidla vyžadují provádění komplexních úprav relačních databází. Zejména v případě již existujících záznamů v databázi je začlenění potřebných strukturálních a datových změn obtížné a často znemožňuje použití obecných databázových nástrojů, které tyto úkony v databázích usnadňují.

Aby bylo možné potřebné aktualizace databází provádět efektivně a bez nutnosti uživatelské intervence, je třeba navrhnout a vytvořit nástroj, který porovná datovou strukturu a obsah stanovené databáze s požadovanou předlohou a na základě zjištěných rozdílů uvede databázi do stavu odpovídajícímu předloze.

Abstract

Maintenance and evolution of information systems generally require complex arrangements of relational databases. In case of existing records in a database, integration of structural and data changes is difficult and often disables utilization of common database tools, which are usually able to simplify these modifications.

To faciliate execution of necessary database changes efficiently and without user intervention, it is needed to design and develop a utility for comparison of database structure and data against a specified template and for application of detected changes onto a target database.

Obsah

1 Úvod **1**

1.1 Základní motivace 1

1.2 Cíl této práce 2

2 Analýza požadavků **3**

2.1 Základní přehled 3

2.2 Úvod do terminologie relačních databází 3

2.3 Cílová uživatelská skupina 4

2.4 Specifikace požadavků 4

2.5 Analýza uživatelské interakce 4

2.5.1 Výběr předlohy a cílové databáze 5

2.5.2 Porovnání struktury 6

2.5.3 Výběr tabulek pro porovnání dat 7

2.5.4 Porovnání dat 8

2.5.5 Aktualizace struktury a dat cílové databáze 9

2.6 Analýza aplikační logiky 9

2.6.1 Porovnání struktury 10

2.6.2 Nastavení porovnání dat 12

2.6.3 Typy datových rozdílů 15

2.6.4 Aktualizace struktury a dat cílové databáze 17

2.7 Datová analýza 20

2.7.1 Uchování informací o struktuře databáze 20

2.7.2 Uchování datových záznamů databáze 20

2.7.3 Zaznamenání rozdílů 21
2.7.4 Zaznamenání změn 21
2.8 Akceptovaný rozsah funkčnosti 22

3 Návrh řešení 23
3.1 Základní přehled 23
3.2 Uživatelské rozhraní 24
3.3 Uchování informací o struktuře a datovém obsahu databáze 25
3.4 Zaznamenání rozdílů 25
3.5 Zaznamenání změn 27
3.6 Transformace rozdílů na změny 27

4 Implementace řešení 29
4.1 Základní přehled 29
4.2 Využití sady tříd DataSet pro porovnání metadat 29
4.3 Využití kolekce ExtendedProperties třídy DataColumn 30
4.4 Využití technologie LINQ 30

5 Závěr 33

Literatura 35

A Používání nástroje 37

B Obsah přiloženého CD 43

C Seznam použitých zkratek a pojmů 45

Seznam obrázků

1.1 Databázové prostředí projektu využívajícího generativní infrastruktury . . 1

2.1 Ilustrace sledu kroků průvodce uživatelského rozhraní 6

2.2 Ilustrace sledu kroků průvodce pro zadání předlohy a cílové databáze . . . 6

2.3 Příklad zobrazení detekovaných strukturálních rozdílů 7

2.4 Rozvržení prvků formuláře pro zadání výběru tabulek pro porovnání dat . 8

2.5 Příklad zobrazení detekovaných datových rozdílů vybraných tabulek 8

2.6 Příklad zobrazení všech detekovaných rozdílů 9

2.7 Příklad zobrazení navržených změn 10

2.8 Postup při porovnání struktury 10

2.9 Postup porovnání tabulek a jejich sloupců 11

2.10 Postup porovnání omezení 12

2.11 Postup porovnání ostatních objektů 13

2.12 Povolené operace režimů porovnání tabulek 14

2.13 Příklad spárovaných tabulek zdrojové a cílové databáze 14

2.14 Příklad konečných stavů tabulky cílové databábáze dle režimu režimu porovnání dat 15

2.15 Způsob řazení změn na úrovni databáze 18

2.16 Statická struktura tříd pro uložení informací o struktuře databáze 21

2.17 Statická struktura tříd pro uložení databázových záznamů 21

2.18 Statická struktura tříd pro uložení rozdílů 22

2.19 Statická struktura tříd pro uložení změn 22

3.1 Návrh hlavních tříd uživatelského rozhraní 24

3.2 Statická struktura tříd pro uchování informací o struktuře databáze 26
3.3 Statická struktura tříd pro zaznamenání detekovaných rozdílů 26
3.4 Statická struktura tříd pro zaznamenání změn 27

A.1 Úvodní stránka průvodce 37
A.2 Stránka pro zadání zdrojové databáze 38
A.3 Stránka pro zadání cílové databáze 38
A.4 Seznam detekovaných strukturálních rozdílů 39
A.5 Výběr tabulek pro porovnání dat 39
A.6 Zobrazený seznam detekovaných datových rozdílů při daném nastavení porovnání 40
A.7 Rekapitulace všech zjištěných rozdílů 40
A.8 Pořadí navržených změn 40
A.9 Poslední stránka průvodce 41

Seznam tabulek

2.1 Funkční požadavky 5
2.2 Nefunkční požadavky 5
2.3 Podporované datové typy 11

3.1 Řešení (solution) a jednotlivé projekty aplikace Visual Studio 2010 23
3.2 Přehled činnosti stránek průvodce a jejich navigačních vztahů 25

4.1 Příkaz jazyka SQL pro vytvoření omezení DEFAULT z uložených metadat 30
4.2 Použití vlastnosti ExtendedProperties pro uložení vlastností sloupců 31
4.3 Příklad využívající konvenční příkazy 31
4.4 Příklad využívající LINQ a Lambda výrazy 32

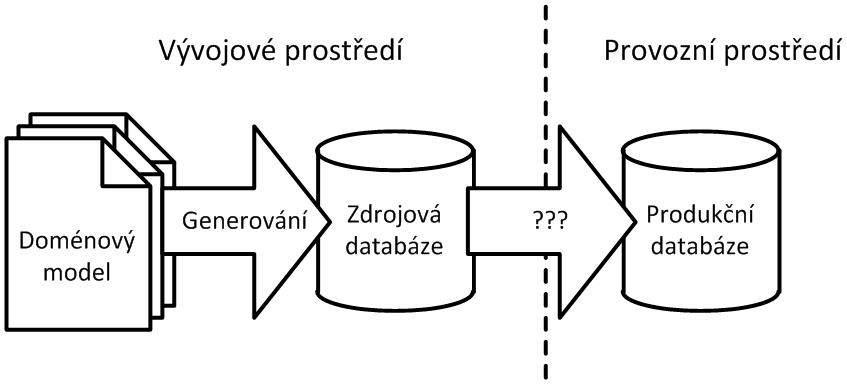
C.1 Seznam použitých zkratek a pojmů 45

Kapitola 1

Úvod

1.1 Základní motivace

Generativní přístup k vývoji podnikových aplikací nabízí efektivní řešení problémů souvisejících s nutností spravovat, udržovat a rozvíjet velké množství vývojových artefaktů, které jsou velmi často nositeli duplicitních informací (viz [3]). Tato skutečnost se týká i problematiky správy, údržby a rozvoje relačních databázových úložišť, které v rámci implementace podnikových aplikací v současné době dominují. Typické uspořádání vývojového a provozního prostředí projektu podnikové aplikace využívající generativní infrastrukturu a databázové úložiště v podobě relační databáze je znázorněno na obrázku 1.1.



Obrázek 1.1: Databázové prostředí projektu využívajícího generativní infrastruktury

Zdrojová databáze, která je zdrojem veškerých vývojově podmíněných strukturálních a datových změn, je vytvářena pomocí generátoru artefaktů a její aktuální podoba je vždy přesným obrazem zdrojového doménového modelu. I přes nesporné přínosy, které generativní infrastruktura pro správu podnikové databáze nabízí, stále zůstává velkým problémem přenos strukturálních a datových změn ze zdrojového do produkčního prostředí podnikové aplikace. Produkční databáze je totiž v průběhu činnosti aplikace

aktualizována po stránce datového obsahu, který ve většině případů činí velké potíže při začleňování datových záznamů ze zdrojové databáze softwarového projektu. Za těchto okolností často nezbývá jiné řešení než časově náročná a problematická manuální příprava transformačních mechanismů (např. příkazových dávek jazyka SQL), které umožní požadované změny datového úložiště zavést do produkčního prostředí.

Zdá se, že možným východiskem by mohlo být provedení detailní analýzy problematiky strukturálních a datových změn s následným návrhem a implementací vhodné nástrojové podpory, která by vývojářům a jiným technickým pracovníkům nabídla efektivní řešení této komplexní problematiky.

1.2 Cíl této práce

Cílem této práce je zanalyzovat problematiku změn struktury a dat relačních databází a následně navrhnout a vytvořit nástroj, který bude schopen vzájemně porovnat zdrojovou a cílovou databázi s přihlédnutím k uživatelem požadovanému režimu porovnání a aplikovat požadované změny na cílové databázi.

Vyvíjený nástroj by měl být pokud možno univerzálně použitelný a měl by umožnit případný další rozvoj dle aktuálních potřeb jeho uživatelů.

Tato práce je členěna dle klasických fází softwarových projektů a odráží tak skutečný postup řešení tohoto projektu.

Kapitola 2

Analýza požadavků

2.1 Základní přehled

V této kapitole jsou shrnuty a rozpracovány základní požadavky na funkčnost nástroje a identifikovány základní procesy umožňující řešení požadované problematiky.

Dále budou uvedeny následující podkapitoly:

- úvod do terminologie relačních databází,
- cílová uživatelská skupina,
- specifikace požadavků,
- analýza uživatelského rozhraní,
- analýza aplikační logiky,
- datová analýza,
- akceptovaný rozsah funkčnosti.

2.2 Úvod do terminologie relačních databází

Základní stavební jednotkou relačních databází je tzv. relace, tabulka, jež se dá členit na sloupce a řádky. Každá tabulka má v rámci databáze jedinečné jméno a obvykle představuje nějakou reálnou entitu. Její sloupce jsou pak atributy představující vlastnosti, které potřebujeme pro danou tabulku zaznamenávat. Například tabulka *Osoba* může obsahovat mimo jiné sloupce *Jméno*, *Příjmení*, *Datum narození* a další. Mezi nejdůležitější vlastnosti sloupců patří zejména jejich datový typ a jeho případné parametry nebo informace, zda takový sloupec akceptuje „neznámou“ hodnotu označovanou *NULL*.

Řádky tabulek můžeme nazývat záznamy. Ty tvoří datovou *n-tici* obsahující hodnoty, jež odpovídají jednotlivým sloupcům tabulky, přičemž každý sloupec obsahuje právě jednu skalární hodnotu. Záznamy obvykle odpovídají reálným prvkům tabulky, například řádek

tabulky *Osoba* by odpovídal jednomu konkrétnímu člověku. Každý řádek dané tabulky je jedinečný.

Rozsah možných hodnot záznamů tabulky je možné zúžit pomocí tzv. *omezení*, což jsou pravidla definovaná na jednom či více sloupcích tabulky. Omezení *UNIQUE* nedovolí přidat záznam se stejnými hodnotami v jím označených sloupcích, jaké má jiný záznam dané tabulky. V žádném z těchto sloupců též nesmí být hodnota *NULL*. Omezení *primárního klíče* má stejný charakter jako *UNIQUE*, na rozdíl od něj však tabulka může obsahovat maximálně jeden *primární klíč*. Na sloupce primárního klíče i omezení *UNIQUE* se může odkazovat *cizí klíč*, definovaný přes odpovídající množinu sloupců. Ke každé hodnotě *cizího klíče* pak musí existovat odpovídající hodnota *primárního klíče* v odkazované či rodičovské tabulce. Omezení *DEFAULT* udává hodnotu, kterou má mít buňka v případě, že žádná hodnota není uvedena. Omezení typu *CHECK* kontroluje platnost hodnoty daného sloupce podle nastaveného *ověřovacího pravidla*.

Základní seznámení s relačními databázemi nabízí například [6], pro více informací pak viz [2].

2.3 Cílová uživatelská skupina

Vzhledem k charakteru problematiky, jíž se vyvíjený nástroj zabývá, se dá očekávat, že jeho typickými uživateli budou takřka výhradně vývojáři podnikových aplikací a správci relačních databází. Tomu by mělo odpovídat uzpůsobení vzhledu a funkčnosti vyvíjeného nástroje.

2.4 Specifikace požadavků

Na základě diskuse s budoucími uživateli nástroje byly specifikovány klíčové požadavky, jež jsou uvedeny v následujících tabulkách. Funkční požadavky uvedené v tabulce 2.1 zachycují nároky kladené na samotnou funkčnost aplikace, v nefunkčních požadavcích v tabulce 2.2 jsou shrnuty požadavky, které vymezují způsob jejího provedení a technologie, které mají být použity (v souladu s rozdělením [7]).

2.5 Analýza uživatelské interakce

Interakce uživatele s nástrojem při porovnání a případné aktualizaci struktury a dat cílové databáze podle předlohy bude probíhat prostřednictvím průvodce v grafickém uživatelském rozhraní. V každém okamžiku se uživatel bude moci vrátit o krok zpět. Přehledový diagram aktivit ilustrující sled jednotlivých kroků průvodce je zobrazen na obrázku 2.1.

V následujících podkapitolách jsou tyto aktivity podrobně rozebrány.

Číslo	Oblast	Popis požadavku
RF1	struktura	Nástroj umožní načíst doménový model obsahující doplňkové informace o struktuře databáze.
RF2	struktura	Nástroj umožní načíst strukturu databáze.
RF3	struktura	Nástroj umožní porovnat strukturu cílové databáze proti zdrojové databázi (předloze) a rozdíl zaznamenat pro další zpracování.
RF4	struktura	Nástroj umožní aktualizovat strukturu cílové databáze.
RF5	struktura	Nástroj umožní uložit požadované strukturální změny do souboru pro pozdější zpracování.
RF6	data	Nástroj umožní načíst obsah databáze.
RF7	data	Nástroj umožní porovnat data uložená v databázi proti předloze a rozdíl zaznamenat pro další zpracování.
RF8	data	Nástroj umožní aktualizovat data v cílové databázi.
RF9	data	Nástroj umožní uložit požadované datové změny do souboru pro pozdější zpracování.
RF10	data	Nástroj umožní stanovit rozsah změn aplikovaných na cílovou databázi.

Tabulka 2.1: Funkční požadavky

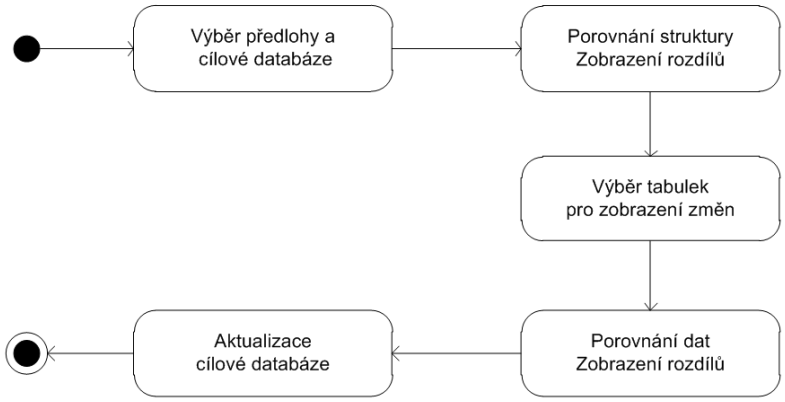
2.5.1 Výběr předlohy a cílové databáze

Pro získání informací potřebných datových struktur pro porovnání bude uživatel vyzván k zadání zdrojové a cílové databáze a nepovinně doménového modelu předlohy. Relační databáze budou specifikovány pomocí řetězce připojení k databázi, který uživatel zadá buď přímo, nebo mu jej nástroj pomůže sestavit. Doménový model bude načten ze specifikovaného souboru ve formátu dokumentu jazyka XML.

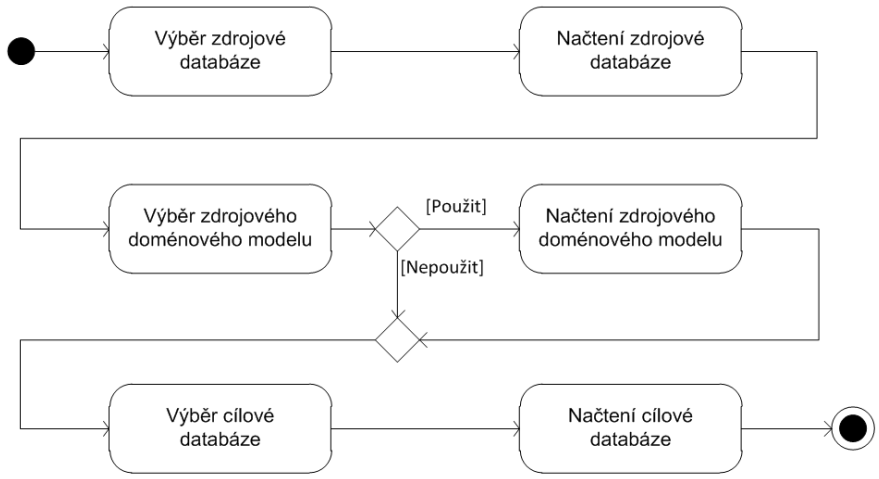
Pokud nebude možné přistupovat k zadané databázi či souboru nebo dojde k jiné chybě při čtení, nástroj vyzve uživatele ke změně zadání. Pořadí výběru ilustruje obrázek 2.2.

Číslo	Popis požadavku
RN1	Nástroj bude možné spustit jako aplikaci s grafickým uživatelským rozhraním.
RN2	Nástroj bude podporovat práci s databázemi Microsoft SQL Server 2008.
RN3	Nástroj bude navržen modulárně pro usnadnění pozdější integrace s jinými aplikacemi.
RN4	Nástroj bude provozován na platformě .NET Framework 4.0.
RN5	Nástroj bude vyvinut v integrovaném prostředí Microsoft Visual Studio 2010.

Tabulka 2.2: Nefunkční požadavky



Obrázek 2.1: Ilustrace sledu kroků průvodce uživatelského rozhraní



Obrázek 2.2: Ilustrace sledu kroků průvodce pro zadání předlohy a cílové databáze

Jediným důsledkem případného vynechání zadání doménového modelu bude v pozdějších krocích nutnost ručního doplnění informací potřebných pro porovnání dat.

2.5.2 Porovnání struktury

Po úspěšném načtení zdrojové a cílové databáze nástroj porovná jejich strukturu a zobrazí uživateli seznam strukturálních rozdílů. Pro lepší přehlednost budou rozdíly v tabulkách, sloupcích a omezeních řazeny po tabulkách, kterých se týkají. V samostatné položce pak budou zobrazeny rozdíly ostatních databázových objektů (např. pohledů či procedur).

Na obrázku 2.3 je znázorněna možná podoba takového seznamu rozdílů.

- Tabulka Activity
  - Chybějící sloupec „Note“
- Tabulka Client
  - Chybějící tabulka
  - Chybějící omezení primárního klíče „PK_Client_Id“
- ...
- Ostatní objekty
  - Chybějící pohled „Contact_View“

Obrázek 2.3: Příklad zobrazení detekovaných strukturálních rozdílů

2.5.3 Výběr tabulek pro porovnání dat

Nástroj zobrazí uživateli seznam všech tabulek zdrojové databáze s informací o tom, kolik záznamů obsahují. Uživatel z nich vybere ty, jejichž data požaduje porovnat, určí způsob jejich porovnání a vybere jeden nebo více tzv. *párovacích sloupců*, podle nichž má dojít ke vzájemnému přiřazení záznamů vybraných tabulek zdrojové a cílové databáze.

Uživatel požaduje tyto režimy porovnání a aktualizace:

- odstranění přebytečných záznamů,
- aktualizace identifikačních sloupců,
- umožnění změn v ostatních sloupcích.

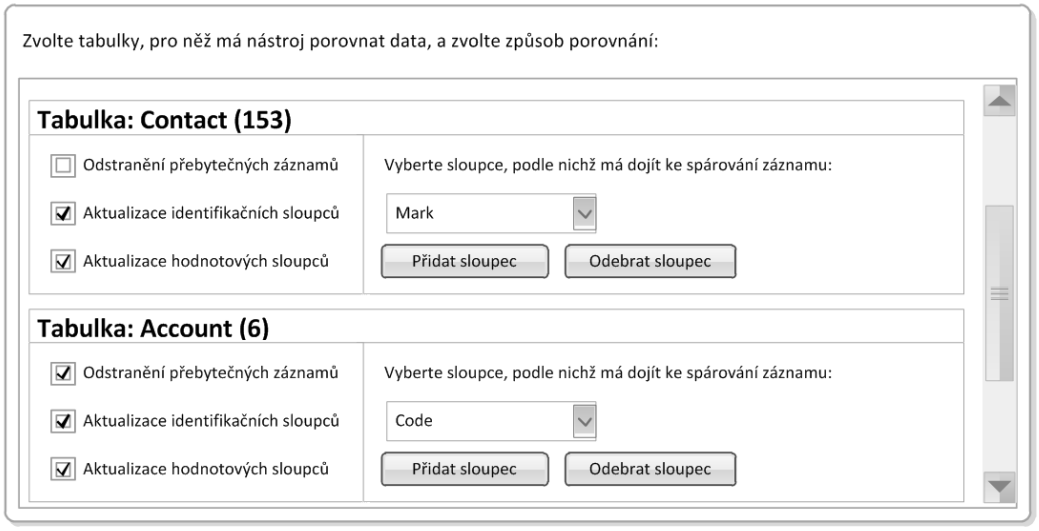
Zvolí-li uživatel režim umožňující smazání přebytečných záznamů, nástroj detekuje nejen chybějící záznamy v dané tabulce cílové databáze oproti zdrojové, ale i ty, které tabulka zdrojová databáze neobsahuje, a v dalších krocích tak umožní jejich smazání.

Umožněním aktualizace identifikačních sloupců se rozumí kontrola hodnoty primárního klíče spárovaných záznamů (za předpokladu, že párování probíhá podle jiného sloupce). Tento režim nemá smysl pro nově přidávané tabulky a tabulky bez definovaného primárního klíče.

Poslední režim porovnání udává, zda má nástroj kontrolovat hodnoty záznamů ve sloupcích neobsahujících primární klíč (a samozřejmě párovací sloupce).

Možný vzhled formuláře ukazuje obrázek 2.4.

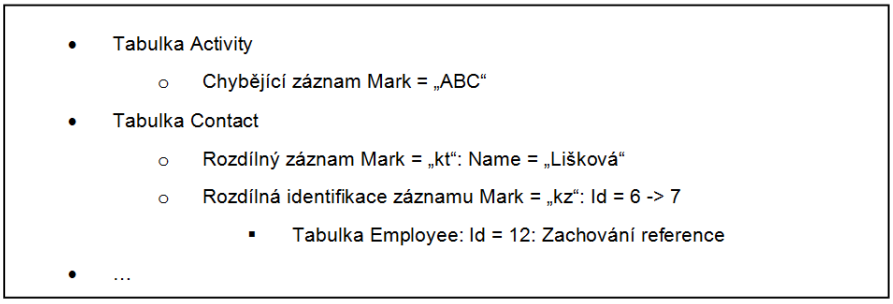
V případě, že uživatel v předchozích krocích specifikoval doménový model, nástroj z něj vytěží doplňkové informace o obsažených tabulkách a podle nich předvyplní pole pro výběr párovacích sloupců.



Obrázek 2.4: Rozvržení prvků formuláře pro zadání výběru tabulek pro porovnání dat

2.5.4 Porovnání dat

Výsledkem porovnání datového obsahu vybraných tabulek bude podobně jako při porovnání struktury seznam rozdílů přehledně řazený po tabulkách. Nástroj vždy zobrazí záznamy určené ke smazání, přidání i změně, přičemž u odebíraných a přesouvaných záznamů budou uvedeny i jimi ovlivněné záznamy tabulek, jež se neúčastnily porovnání. Přesunem záznamu se myslí změna hodnoty primárního klíče. Možná podoba výstupu porovnání dat je znázorněna na obrázku 2.5.



Obrázek 2.5: Příklad zobrazení detekovaných datových rozdílů vybraných tabulek

V případě, že nástroj detekuje přebývající záznam, na jehož primární klíč se odkazují jiné záznamy z tabulek neúčastnících se porovnání dat, bude uživatel vyzván, aby pro takové záznamy vybral vhodné řešení:

- nastavení hodnoty *NULL* sloupcům příslušného cizího klíče (pokud to připouští omezení daného sloupce předlohy),

- nastavení výchozí hodnoty sloupcům příslušného cizího klíče (v případě, že existuje v předloze),
- smazání záznamu (pokud se na něj neodkazuje žádný jiný záznam),
- „invazivní“ mazání – kaskádní smazání všech odkazujících se záznamů (pokud není možné smazat odkazující se záznam výše uvedeným „klasickým mazáním“).

Změny dat způsobené změnami struktury (např. smazáním sloupce) nebudou zobrazeny.

2.5.5 Aktualizace struktury a dat cílové databáze

Poslední fází synchronizace je provedení změn na cílové databázi. Nástroj nejprve uživateli zrekapituluje detekované rozdíly ve společném formuláři (obrázek 2.6), poté sestaví posloupnost změn řešící nalezené rozdíly, které opět zobrazí uživateli (obrázek 2.7).

- Tabulka Activity
  - Chybějící sloupec „Note“
  - Chybějící řádek Mark = „ABC“
- Tabulka Client
  - Chybějící tabulka
  - Chybějící omezení primárního klíče „PK_Client_Id“
- Tabulka Contact
  - Rozdílný záznamu Mark = „kt“: Name = „Lišková“
  - Rozdílná identifikace záznamu Mark = „kz“: Id = 6 -> 7
    - Tabulka Employee: Id = 12: Zachování reference
- ...
- Ostatní objekty
  - Chybějící pohled „Contact_View“

Obrázek 2.6: Příklad zobrazení všech detekovaných rozdílů

Po potvrzení změn dojde k jejich aplikaci na cílovou databázi. Uživatel bude informován, zda se podařilo úspěšně aktualizovat cílovou databázi, nebo bylo nutné vrátit ji do stavu, v jakém byla před zahájením provádění změn.

2.6 Analýza aplikační logiky

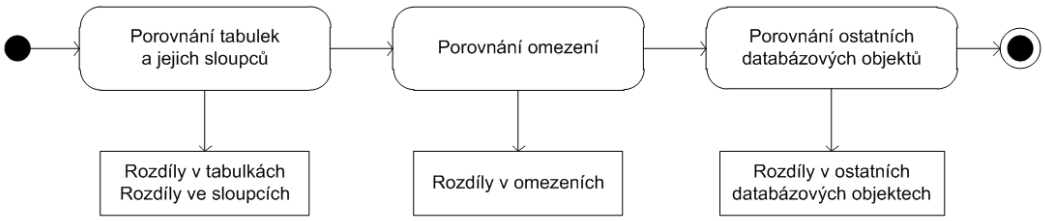
V následujících bodech je podrobně rozebrána podstata funkčnosti porovnání zadaných databází a aplikace změn na cílovou databázi z pohledu procesů aplikační logiky.

- Tabulka Activity: Přidání sloupce Note
- Tabulka Activity: Přidání řádku Mark = „ABC“
- Tabulka Client: Přidání tabulky
- Tabulka Client: přidání omezení primárního klíče „PK_Client_Id“
- Tabulka Contact: Změna záznamu Mark = „kt“: Name = „Lišková“
- Tabulka Contact: Přidání záznamu Mark = „kz“: Id = 7
- Tabulka Employee: Změna reference záznamu Id = 12: ContactId = 7
- Tabulka Contact: Smazání záznamu Mark = „kz“: Id = 6
- ...
- Smazání pohledu „Contact_View“

Obrázek 2.7: Příklad zobrazení navržených změn

2.6.1 Porovnání struktury

Úlohu porovnání struktury relačních databází tvoří tři samostatné činnosti, jež jsou zobrazeny na obrázku 2.8.



Obrázek 2.8: Postup při porovnání struktury

Tyto aktivity jsou dále podrobněji rozpracovány.

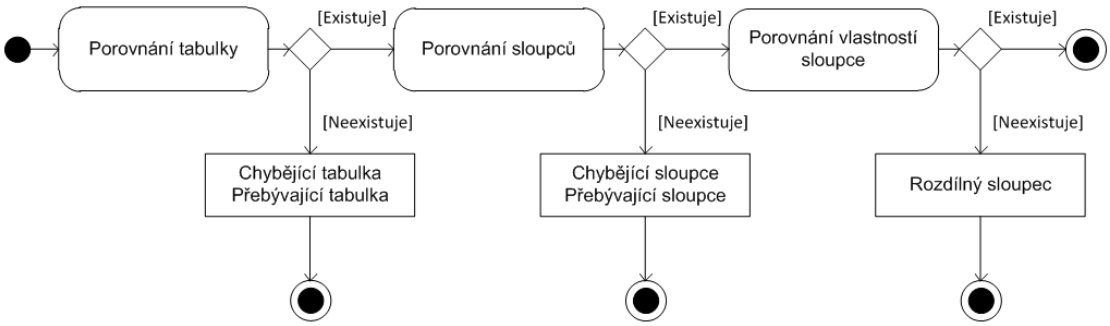
2.6.1.1 Porovnání tabulek a jejich sloupců

Pro každou tabulku cílové databáze se nástroj pokusí vyhledat tabulku zdrojové databáze stejného názvu. Přebývající tabulky budou přidány do seznamu rozdílů s návrhem na smazání, chybějícím tabulkám bude navrženo přidání do cílové databáze. Spárované tabulky budou dále podrobeny porovnání sloupců.

V dalších verzích nástroje by mohla být uživateli nabídnuta možnost ručního párování nespárovaných tabulek, jež mají v obou databázích různé jméno. Takovým tabulkám by byla místo návrhů na přidání respektive smazání z cílové databáze navržena změna názvu. Dále by pro ně proběhlo standardní porovnání sloupců.

Postup při porovnání tabulek a jejich sloupců je zobrazen na obrázku 2.9.

U tabulek, kterým byl ve zdrojové databázi nalezen protějšek, nástroj porovná jejich



Obrázek 2.9: Postup porovnání tabulek a jejich sloupců

sloupce. Párování bude probíhat podle názvu, přičemž kontrole budou podléhat všechny jejich parametry:

- datový typ,
- vlastnosti datového typu (např. délka u řetězce či přesnost u čísla),
- možnost nastavení hodnoty *NULL*,
- automaticky číslované pole (vlastnost *IDENTITY*).

Přebývající sloupce budou přidány do seznamu rozdílů s návrhem na smazání, chybějícím sloupcům bude navrženo přidání a odlišným sloupcům změna parametrů. Setká-li se nástroj s nepodporovaným datovým typem, odmítne s danou databází pracovat. Podporované datové typy jsou uvedeny v tabulce 2.3.

Název typu	**Popis**	**Příklad**
nvarchar	řetězcový (Unicode)	N'Testovací řetězec'
int	celočíselný znaménkový 32 bitů	12345
bit	pravdivostní	0 nebo 1
decimal	s pevnou řádovou čárkou	25487.2154
uniqueidentifier	jedinečný identifikátor GUID	b25c1b12-576c-4c19-9a6e-4ce359841de2
datetime	datum a čas	12.5.2010 18:45:23.667

Tabulka 2.3: Podporované datové typy

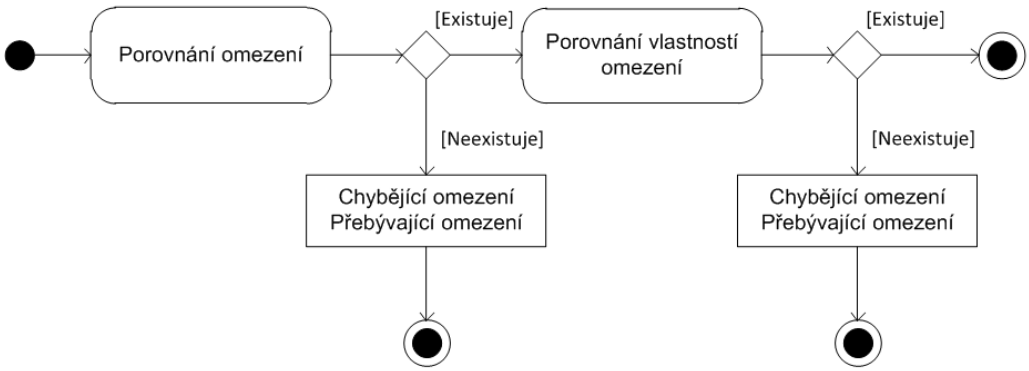
V dalších verzích nástroje by též uživatel měl mít možnost ručního párování nespárovaných sloupců odpovídajících si tabulek. Takovým sloupcům by byly zrušeny návrhy na přidání a smazání z cílové databáze a byla by jim navržena změna názvu.

2.6.1.2 Porovnání omezení

Nástroj dále porovná výskyt primárních a cizích klíčů, omezení *UNIQUE*, *DEFAULT* a *CHECK* a indexů. Přebývajícím omezením bude navrženo jejich smazání, chybějícím přidání do cílové databáze.

Nástroj spáruje jen omezení, která se shodují ve všech svých parametrech (např. množině sloupců, na něž se vztahují, nebo výchozí hodnotě u omezení *DEFAULT*).

Postup porovnání je zachycen na obrázku 2.10.



Obrázek 2.10: Postup porovnání omezení

2.6.1.3 Porovnání ostatních objektů

Nakonec nástroj porovná pohledy, spouště, procedury a funkce. Způsob porovnání a vypořádání se s nimi bude analogické porovnání omezení. Nástroj bude porovnávat:

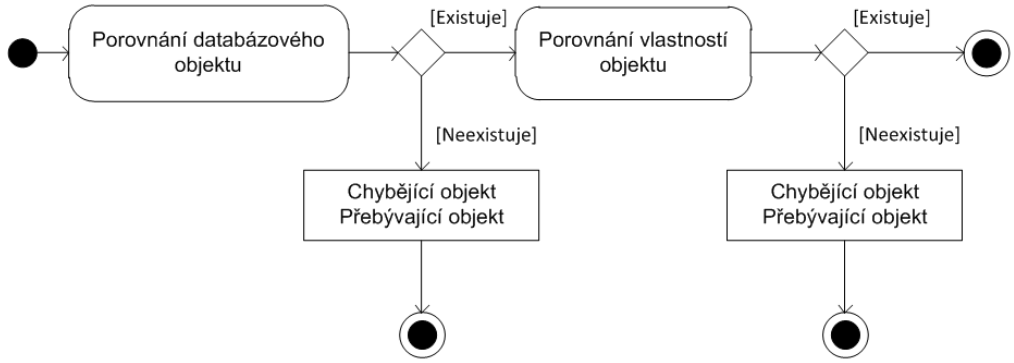
- typ objektu,
- jméno objektu,
- vlastnosti objektu.

Shodné objekty v předloze a cílové databázi zůstanou beze změny. Přebývajícím objektům v cílové databázi bude navrženo smazání a chybějícím přidání. Postup porovnání je zachycen na obrázku 2.11.

2.6.2 Nastavení porovnání dat

2.6.2.1 Význam párovacích sloupců

U každé porovnávané tabulky musí být specifikován jeden nebo více jejích sloupců, podle nichž se nástroj pokusí spárovat záznamy zdrojové a cílové databáze. Obvykle jím



Obrázek 2.11: Postup porovnání ostatních objektů

může být sloupec nesoucí primární klíč nebo sloupec obsahující kód či jiný vývojářem zvolený identifikátor záznamu.

Pokud hodnoty záznamů v párovacích sloupcích nebudou jednoznačnými identifikátory (tj. tabulka předlohy nebo cílové databáze bude obsahovat více stejných kombinací hodnot v těchto sloupcích), nástroj takovou skutečnost ohlásí uživateli a vyzve jej ke změně volby párovacích sloupců nebo zrušení požadavku na porovnání dotčené tabulky.

2.6.2.2 Režimy porovnání

Uživatel u každé porovnávané tabulky zvolí výběrem tří porovnávacích režimů jednu z osmi variant porovnání tabulek. Jedná se o kombinaci dvou variant pro přítomnost záznamů v tabulce a čtyř variant porovnání dat v záznamech. Význam jednotlivých režimů shrnuje obrázek 2.12.

Význam jednotlivých režimů a z nich vyplývajících operací uvedených ve výše uvedeném obrázku je následující:

Režim tabulky *Shoda* kontroluje přítomnost právě těch záznamů, které se vyskytují v předloze (přičemž tyto záznamy jsou identifikovány pomocí párovacích sloupců). Naproti tomu v režimu *Minimální obsah* předloha definuje záznamy, které cílová databáze musí obsahovat; kromě nich však může obsahovat i libovolný počet dalších záznamů. Jinými slovy, zatímco oba režimy tabulky umožňují přidání záznamů, režim *Shoda* na rozdíl od režimu *Minimální obsah* dovoluje i smazání přebývajících záznamů.

Režimy sloupce se zabývají změnami spárovaných záznamů a jejich případnými změnami. V režimu *Mark – Id* nástroj zkontroluje, zda hodnotám párovacích sloupců odpovídají hodnoty sloupců primárního klíče; ostatní sloupce tabulky se porovnání neúčastní. Zcela opačným případem je režim *Neměnné Id*, kdy nástroj proti hodnotám párovacích sloupců zkontroluje hodnoty záznamů s výjimkou sloupců primárního klíče. V režimu *Celý řádek* pak nástroj kontroluje všechny hodnoty záznamů. Režim *Existence*

Režim tabulky	Režim záznamu	Přidání záznamu	Smazání záznamu	Změna identifikace	Jiná změna
Shoda	Celý řádek	Ano	Ano	Ano	Ano
	Mark – Id	Ano	Ano	Ano	-
	Neměnné Id	Ano	Ano	-	Ano
	Existence	Ano	Ano	-	-
Minimální obsah	Celý řádek	Ano	-	Ano	Ano
	Mark – Id	Ano	-	Ano	-
	Neměnné Id	Ano	-	-	Ano
	Existence	Ano	-	-	-

Obrázek 2.12: Povolené operace režimů porovnání tabulek

je velmi omezený režim, ve kterém se kontroluje pouze přítomnost záznamu (hodnot párovacích sloupců) v tabulce.

2.6.2.3 Příklad porovnání dat

Jednotlivé režimy se dají dobře ilustrovat na následujícím příkladu. Na obrázku 2.13 je zobrazena zdrojové databáze a jí odpovídající tabulka cílové databáze před porovnáním dat. Primárním klíčem obou tabulek je sloupec *Id* a porovnání proběhlo při volbě párovacího sloupce *Mark*.

Zdrojová databáze (předloha)

Id	Mark	Text
1	A	áááá
2	N	en
3	C	cé

Cílová databáze

Id	Mark	Text
1	A	á
2	B	bé
3	C	cé

Obrázek 2.13: Příklad spárovaných tabulek zdrojové a cílové databáze

Důsledky zvolení jednotlivých kombinací režimů ilustruje obrázek 2.14.

Porovnáním databázových tabulek na obrázku 2.14 je vidět, že ve všech případech byl přidán do cílové databáze řádek obsahující značku *Mark* = *"N"*. V těch případech, kdy byla povolena změna identifikace, byl přidán na pozici s hodnotou primárního klíče *Id* = *4*. Naproti tomu v režimech *Celý řádek* a *Mark - Id* byl vložen místo řádku *Mark* = *"B"*,

Shoda
Celý řádek

Id	Mark	Text
1	A	áááá
2	N	en
3	C	cé

Shoda
Mark - Id

Id	Mark	Text
1	A	á
2	N	en
3	C	cé

Shoda
Neměnné Id

Id	Mark	Text
1	A	áááá
3	C	cé
4	N	en

Shoda
Existence

Id	Mark	Text
1	A	á
3	C	cé
4	N	en

Minimální obsah
Celý řádek

Id	Mark	Text
1	A	áááá
2	N	en
3	C	cé
4	B	bé

Minimální obsah
Mark - Id

Id	Mark	Text
1	A	á
2	N	en
3	C	cé
4	B	bé

Minimální obsah
Neměnné Id

Id	Mark	Text
1	A	áááá
2	B	bé
3	C	cé
4	N	en

Minimální obsah
Existence

Id	Mark	Text
1	A	á
2	B	bé
3	C	cé
4	N	en

Obrázek 2.14: Příklad konečných stavů tabulky cílové datababáze dle režimu režimu porovnání dat

který byl v případě režimu *Minimální obsah* odsunut na pozici *Id = 4* respektice smazán při porovnání v režimu *Shoda*. Přebývající řádek byl smazán také ve dvou zbývajících případech režimu *Shoda* zakazujících změnu identifikace, kdy po něm zbyla volná pozice *Id = 2*. Tento jev podrobněji vysvětluje kapitola 2.5.5. Hodnota záznamů ve sloupci *Text* pak byla aktualizována v případech, kdy byla povolen režim *Jiná změna*.

2.6.3 Typy datových rozdílů

V závislosti na zvoleném režimu porovnání může nástroj detekovat až pět různých druhů datových rozdílů:

- chybějící záznam,
- přebývající záznam,
- rozdílná hodnota identifikace,
- rozdílná reference,
- rozdílná hodnota obyčejného sloupce.

Tyto rozdíly jsou podrobněji rozebrány v následujícím výkladu.

2.6.3.1 Chybějící záznam

Chybějící záznam je možné detekovat ve všech režimech porovnání. Obsahuje-li zdrojová databáze záznam, jenž se v cílové databázi nevyskytuje, nástroj jej navrhne přidat. Neumožňuje-li režim porovnání změnu identifikátoru a jiný záznam již má stejnou hodnotu primárního klíče, nástroj jej přidá na jinou volnou pozici.

2.6.3.2 Přebývající záznam

Přebývající záznam je detekován tehdy, když cílová databáze obsahuje záznam, jenž není součástí předlohy, a tabulka je porovnávána v režimu *Shoda*, umožňující smazání záznamu.

Pokud se na přebývající záznam odkazují jiné záznamy z tabulek neúčastnících se porovnání dat, musí pro ně uživatel vybrat vhodné řešení:

- nastavení hodnoty *NULL* dané reference,
- nastavení výchozí hodnoty dané reference,
- smazání záznamu,
- „invazivní“ mazání – kaskádní smazání všech odkazujících se záznamů.

Nastavení hodnoty *NULL* je možné jen tehdy, když to umožňuje nastavení sloupce zdrojové databáze. Podobně nastavení výchozí hodnoty je umožněno pouze v případě, že v předloze existuje odpovídající omezení typu *DEFAULT*.

Odkazovaný záznam může být smazán standardním způsobem, jestliže se na něj již neodkazuje žádný další záznam. V opačném případě může být smazán jen tzv. „invazivním“ mazáním, které daný záznam smaže kaskádně.

2.6.3.3 Rozdílná hodnota identifikace

Jako rozdílná hodnota identifikace budou klasifikovány záznamy, jež se shodují v hodnotách párovacích sloupců, ale mají odlišnou hodnotu primárního klíče. Pro detekování je zároveň potřeba zvolit režim umožňující změnu identifikace a je nutná existence stejného primárního klíče ve zdrojové a cílové databázi.

Má-li dojít ke změně hodnoty primárního klíče, je třeba zachovat reference odkazujících se záznamů, pokud se jejich tabulky neúčastní porovnání dat nebo režim porovnání takových tabulek neporovnává hodnoty obyčejných sloupců. Nástroj proto dohledá všechny takové záznamy a informuje uživatele, že v případě provedení změny identifikace by muselo současně dojít k "vynuceným změnám"v tabulkách, u kterých se na základě nastavení explicitně nepředpokládá taková změna záznamů.

Odkazují-li se na přesouvaný záznam záznamy, pro něž dochází k porovnání hodnotových sloupců, je pro ně rozhodující stav, který popisuje zdrojová databáze.

2.6.3.4 Rozdílná reference

K detekování rozdílné reference dojde tehdy, když je umožněno porovnávat ostatní sloupce a zároveň dojde ke změně hodnoty ve sloupci s cizím klíčem (tzv. *absolutní reference*) nebo takové změně v odkazované tabulce, že se daný záznam syntakticky odkazuje na jiný záznam z pohledu párovacích sloupců (tzv. *relativní reference*).

2.6.3.5 Rozdílná hodnota obyčejného sloupce

Rozdílnou hodnotu obyčejného sloupce je možné zjistit, když je umožněno porovnání ostatních sloupců. Nemá žádné důsledky na vztahy mezi existujícími záznamy.

2.6.4 Aktualizace struktury a dat cílové databáze

V následujících bodech je podrobně popsán proces plánování změn a jejich aplikace na cílovou databázi.

2.6.4.1 Požadavky na plánování změn

Postup provedení požadovaných změn by měl v ideálním případě splňovat množství kritérií, z nichž mezi nejdůležitější patří například:

- provedení strukturálních i datových změn v jediné transakci,
- vykonávání co nejmenšího počtu operací nad cílovou databází,
- jednoduchost a univerzálnost algoritmu plánování změn,
- zachování správnosti dat po co nejdelší dobu („původních“ nebo „nových“),
- aktualizace jen jediné tabulky v každém okamžiku.

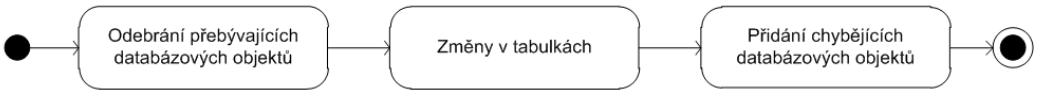
Z výše uvedeného seznamu je patrné, že všechny podmínky nelze splnit zcela, neboť některé se vzájemně vylučují. Popsaný algoritmus je proto navržen tak, aby co nejvíce splňoval nejdůležitější kritéria při použití v běžném provozu.

2.6.4.2 Princip řazení změn

Pro provádění datových změn v tabulkách je potřeba, aby aktuální omezení a vlastnosti sloupců umožnily takové změny provést. To znamená, že je potřeba před prováděním datových změn v dané tabulce provést „polehčující“ změny její struktury a naopak „přitěžující“ až provedení všech datových změn. Obdobná situace pak nastává ve vztahu tabulek k ostatním databázovým objektům.

Nástroj tak nejprve odstraní přebývající databázové objekty, poté podle určitého klíče bude postupně vybírat tabulky, z nichž nejprve odstraní přebývající omezení, poté provede datové změny jen s nutnými zásahy do jiných tabulek v případě odkazujících se záznamů na mazané a přesouvané záznamy, a přidá omezení tabulek. Na závěr přidá nově přidávané databázové objekty.

Výše popsaný postup je znázorněn na obrázku 2.15.



Obrázek 2.15: Způsob řazení změn na úrovni databáze

2.6.4.3 Výběr tabulky pro provedení změn

Nejproblematičtějšími oblastmi plánování změn jsou změny referencí a mazání záznamů. Má-li dojít ke změně reference, je třeba, aby odkazovaná tabulka již byla aktualizována, zejména musí obsahovat (nově přidané) záznamy, na které se měněná tabulka má odkazovat. Z tohoto pohledu se vyplatí postupovat „shora“ (v opačném směru než je orientace příslušného omezení typu cizí klíč). Naopak mazání záznamů by bylo výhodnější provádět „zdola“, neboť by odpadla nadbytečná režie při přesouvání záznamů odkazovaných odebíranými záznamy a zejména by bylo usnadněno kaskádní mazání záznamů. Jelikož se dá předpokládat, že mazání záznamů nebude moc časté, algoritmus je navržen zejména s ohledem na změny referencí a pro mazání záznamů volí zbývající složitější řešení.

Aby mohl nástroj provést všechny změny vždy jen v jediné tabulce (a v ostatních jen zachovávat reference pomocí vynucených změn), musí postupovat od těch, které se buď nikam neodkazují, nebo jsou vázány cizími klíči jen do již aktualizovaných tabulek. Pokud však cizí klíče několika tabulek tvoří orientovanou kružnici, může nastat situace, že ve všech těchto tabulkách má dojít k takovým změnám, že podle výše popsaného postupu nebude možné vybrat žádnou z nich. V tom případě nástroj provede všechny uskutečnitelné změny, čímž nastalou patovou situaci odblokuje, a zbylé změny dokončí později.

Z tohoto pohledu budou za strukturálně uspořádané tabulky považovány ty, u kterých nedochází k přidání primárního klíče, nebo již byl přidán. Odkaz na strukturálně uspořádané tabulky je podmínkou přidávání cizích klíčů. Podobně budou za datově uspořádané tabulky považovány ty, které v danou chvíli „mají záznamy na svých místech“, tedy ve kterých již nedojde k přidání ani přesunu žádného záznamu. Vazba cizími klíči pouze na takové tabulky je podmínkou pro provádění změn referencí. Z hlediska výběru tabulek jsou jednoznačně preferovány ty, které splňují obě podmínky.

2.6.4.4 Problematické změny v tabulce

Aby mohl být záznam smazán, je potřeba, aby se na něj v rozhodujícím okamžiku žádné záznamy neodkazovaly. To může být problém, pokud záznam, na který má být jejich reference změněna, nemůže být přidán dříve, než dojde k uvolnění požadované hodnoty primárního klíče. Při zachování základního požadavku na funkčnost nástroje, aby v každém okamžiku aktualizace zachovával význam uložených dat vůči zdrojové nebo cílové databázi, se zdá být nejjednodušším řešením této problematiky „odsunutí“ záznamů,

jež mají být smazány, z pozic, na které je potřeba přidat či přesunout jiné záznamy, na jinou, volnou pozici (pozicí rozumějme hodnotu primárního klíče záznamu). Takto „odsunuté“ záznamy můžou v tabulce zůstat až do doby, kdy je bude možné bezpečně smazat.

Obdobný problém nastává v situaci, kdy má dojít ke vzájemné či cyklické změně identifikátoru dvou nebo více záznamů. I v této situaci je potřeba „rozbít začarovaný kruh“ tím, že nástroj přesune jeden z těchto záznamů na volnou pozici, čímž umožní postupné přesunutí ostatních záznamů na požadované místo, načež bude možné přiřadit i tomuto záznamu požadovanou hodnotu primárního klíče.

2.6.4.5 Pořadí aplikace změn v tabulce

S ohledem na výše uvedené závislostí bude nástroj provádět změny ve vybrané tabulce v následujícím pořadí:

1. Přidání celé tabulky
2. Vypnutí kaskádních změn
3. Odebrání cizích klíčů vázaných na danou tabulku
4. Odebrání omezení a indexů tabulky
5. Smazání tabulky
6. Smazání sloupců
7. Přidání sloupců (akceptujících hodnotu *NULL*, a to nezávisle na předloze)
8. Změna vlastností sloupců (akceptují hodnotu *NULL* nezávisle na předloze)
9. Změny hodnot primárního klíče:
   (a) Překopírování záznamu na určené místo (nebo na volné, pokud určeno není)
   (b) Vynucené změny reference (změna reference všech záznamů odkazujících se na přesouvaný záznam)
   (c) Smazání původního záznamu
10. Přidání záznamů na určené místo (nebo na volné, pokud není určeno)
11. Změny hodnot obyčejných sloupců a referencí do datově uspořádaných tabulek
12. Změny reference záznamů v jiných tabulkách odkazujících se na záznam, který má být smazán
13. Aktualizace tabulek, jejichž záznamy, které mají být smazány, se odkazují na záznamy této tabulky, jež mají být smazány
14. Smazání záznamů, na které se žádný záznam neodkazuje
15. Změna vlastnosti sloupců na *NOT NULL*
16. Přidání omezení (cizí klíče jen v případě odkazu na strukturálně uspořádané tabulky)

17. Aktivování kaskádních změn

Poznámka – vysvětlení k bodu 13: Dochází-li ke smazání záznamu, na který se odkazují záznamy z jiných tabulek, které mají být také smazány, dojde v bodě 13 k přerušení provádění algoritmu pro aktuální tabulku. Nástroj vybere další tabulku a uspořádá ji podle stejného postupu. Po úspěšném dokončení uspořádání odkazujících se tabulek dojde k dokončení zbývajících kroků v rozpracované tabulce.

Po úspěšném provedení výše popsaných změn ve všech tabulkách dojde k invazivnímu smazání přebývajících záznamů.

2.6.4.6 Aplikace změn na cílovou databázi

Nástroj musí ve všech případech zaručit zachování platných záznamů v databázi. Proto všechny změny proběhnou v rámci jedné transakce, aby v případě selhání aktualizace cílové databáze bylo možné odvolat všechny již provedené změny.

2.7 Datová analýza

V následujících bodech jsou uvedeny a rozebrány požadavky na zaznamenání specifických datových struktur, s nimiž bude nástroj pracovat.

2.7.1 Uchování informací o struktuře databáze

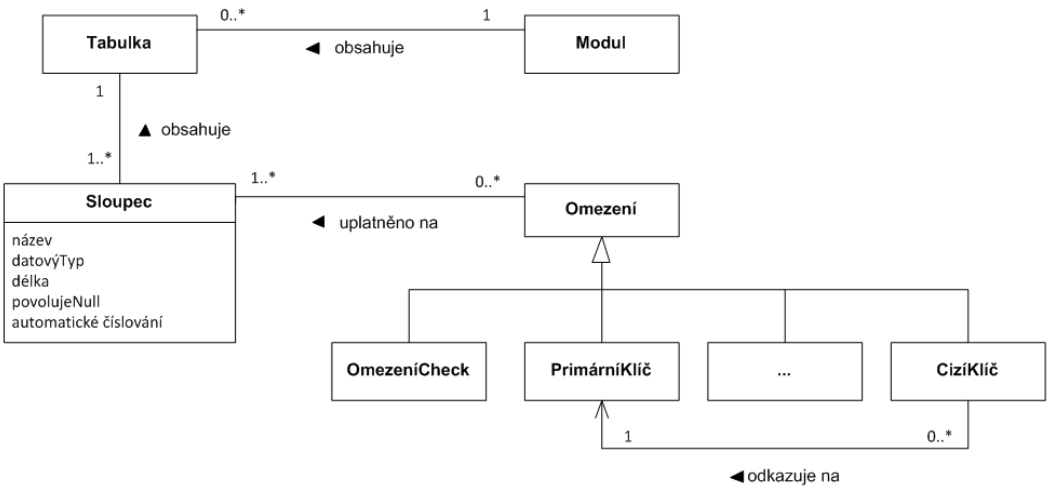
Pro potřeby porovnání a zpracování struktury databáze bude třeba načíst informace o tabulkách, jejich sloupcích a omezeních a ostatních databázových objektech do vhodného datového úložiště. Vhodná podoba takového úložiště je naznačena na obrázku 2.16.

Základním prvkem této struktury je tabulka patřící do modulu, která obsahuje množinu sloupců. Sloupce kromě názvu evidují datový typ a další vlastnosti. Na sloupce pak mohou být uplatněna omezení, z nichž speciální roli má cizí klíč, který se navíc odkazuje na primární klíč.

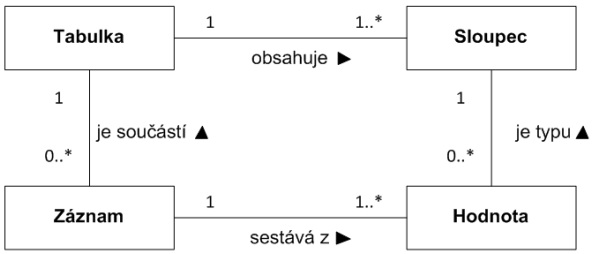
2.7.2 Uchování datových záznamů databáze

Stejně jako struktura databáze, i její záznamy musí být uchovány ve vhodné formě. Její možná podoba je na obrázku 2.17. Ideální by bylo tyto úkoly spojit a najít či vytvořit takové řešení, které umožní zkopírovat strukturu i data do instancí tříd jediné knihovny.

Každý záznam je součástí právě jedné tabulky a sestává z množiny hodnot datového typu daného sloupcem, do nějž tato hodnota patří.



Obrázek 2.16: Statická struktura tříd pro uložení informací o struktuře databáze



Obrázek 2.17: Statická struktura tříd pro uložení databázových záznamů

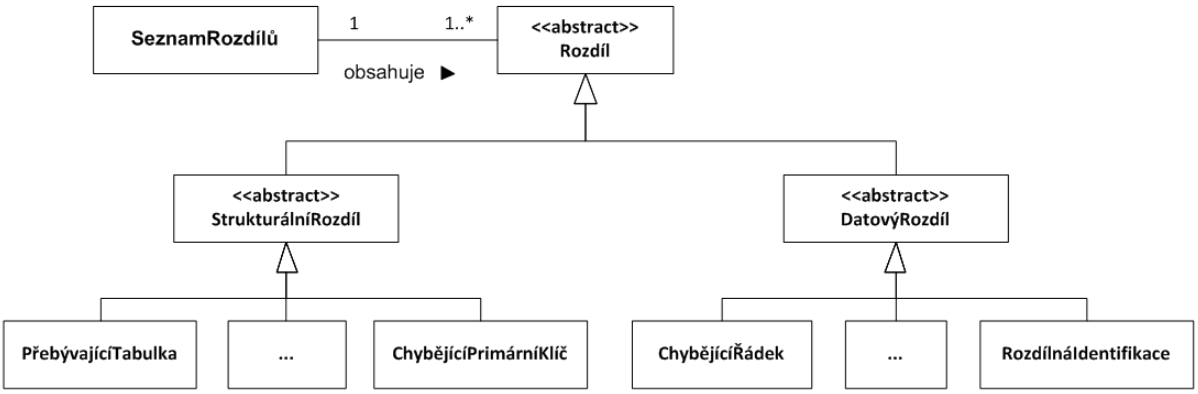
2.7.3 Zaznamenání rozdílů

Záznamy o rozdílech struktury a dat musí být uchovávány v příbuzných objektech, aby bylo možné s nimi pracovat jednotně, ačkoliv svou podstatou jsou zcela odlišné. Je potřeba umožnit jejich snadné řazení podle tabulek, kterých se týkají, evidovat způsob porovnání dat a umožnit jejich snadný převod na změny.

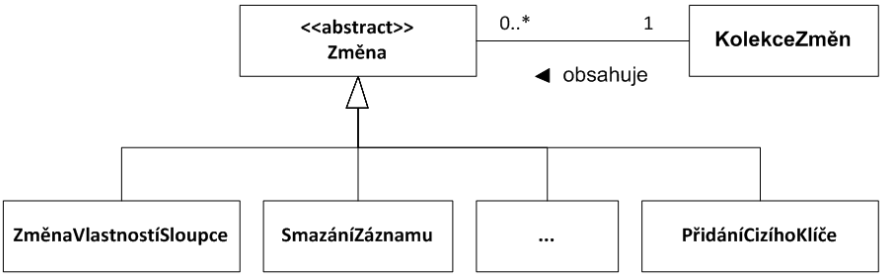
Jednotlivé specializované strukturální rozdíly mají společného abstraktního předka. V podobné situaci jsou datové rozdíly. Seznam rozdílů pak eviduje všechny rozdíly, jež jsou zastřešeny společnou bázovou třídou.

2.7.4 Zaznamenání změn

Třídy reprezentující jednotlivé změny musí být co nejvíce granulární a odrážet pokud možno elementární databázové příkazy. Při svém provádění by neměly obsahovat žádnou složitou logiku, aby bylo možné je seřadit a jednoduchým způsobem převést na databázové příkazy.



Obrázek 2.18: Statická struktura tříd pro uložení rozdílů



Obrázek 2.19: Statická struktura tříd pro uložení změn

Abstraktní třída změny zastřešuje své vysoce specializované potomky, které jsou pak uspořádány do kolekce změn.

2.8 Akceptovaný rozsah funkčnosti

Z výše uvedených poznatků vyplývá, že problematika automatizovaných změn v databázi je velmi obsáhlá, přičemž aktuální potřeby stávajících softwarových projektů představují pouze jistou podmnožinu, na kterou bylo rozhodnuto se v rámci tohoto projektu soustředit.

S ohledem na dostupný časový rámec a omezené vývojové kapacity bylo do dalších verzí aplikace přesunuto načítání doplňkových informací z doménového modelu a provádění složitějších databázových změn. Naopak byl upřednostněn vývoj takového nástroje, který bude snadno rozšiřitelný o další nalezenou požadovanou funkčnost. V první verzi by měl nástroj dokázat porovnat objekty a datové typy v aktuálně používané verzi schématu doménového modelu a ilustrovat, že je možné provádět základní jednoduché změny nad databází s výhledem postupného rozšiřování funkčnosti podle potřeby. S ohledem na snadné kontroly činnosti nástroje bylo rozhodnuto nepoužívat kaskádní změny na databázi, neboť nejsou u standardně navrhovaných databázových struktur používány.

Kapitola 3

Návrh řešení

3.1 Základní přehled

Nástroj bude implementován ve vývojovém prostředí Microsoft Visual Studio 2010 na platformě Microsoft .NET Framework 4.0 pomocí programovacího jazyka C# a bude pracovat s relačními databázemi Microsoft SQL Server 2008.

Řešení (solution) sestává z projektů uvedených v tabulce 3.1.

Název	**Popis**
AsBest.MetaToolkit.Synchronizer	Řešení (solution) obsahující všechny níže uvedené projekty.
AsBest.MetaToolkit.Reflector	Obsahuje pomocnou aplikační logiku pro načítání databázové struktury a dat.
AsBest.MetaToolkit.Synchronizer.Client	Definuje a obsluhuje uživatelské rozhraní.
AsBest.MetaToolkit.Synchronizer.Common	Obsahuje pomocné univerzální třídy.
AsBest.MetaToolkit.Synchronizer.Logic	Hlavní projekt zodpovědný za řízení chodu aplikace, zjištění rozdílů, jejich převod na změny a jejich aplikaci na cílovou databázi.
AsBest.MetaToolkit.Synchronizer.Setup	Vytváří instalační soubory aplikace.

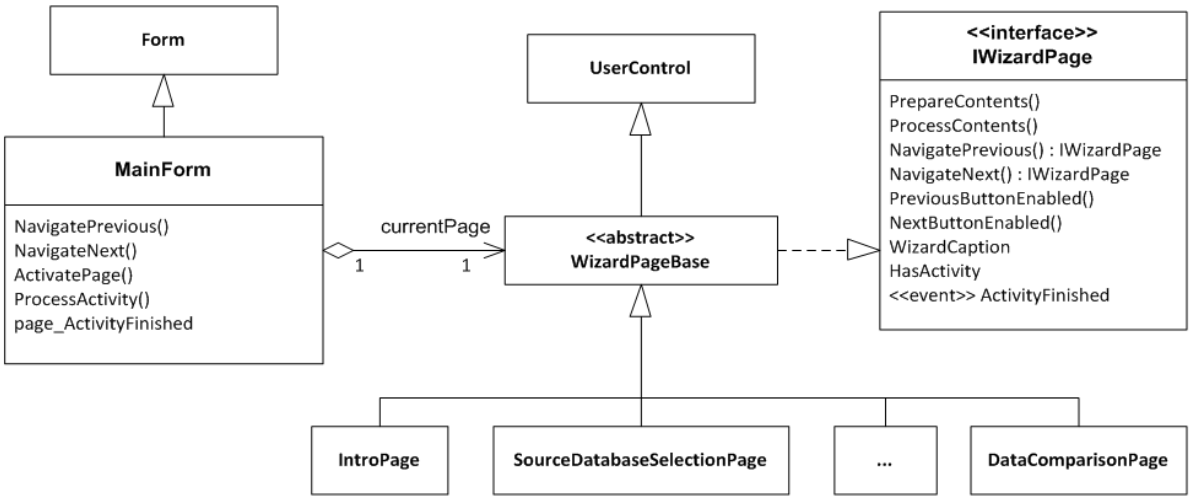
Tabulka 3.1: Řešení (solution) a jednotlivé projekty aplikace Visual Studio 2010

Projekt *AsBest.MetaToolkit.Reflector* obsahuje pomocnou aplikační logiku původně vyvinutou v rámci [3] pro načítání základní struktury a dat relační databáze a jejich uložení do souborů formátu XML. Tento projekt bude možné rozšířit pro požadovanou funkčnost vyvíjeného nástroje.

3.2 Uživatelské rozhraní

Nástroj bude komunikovat s uživatelem prostřednictvím formulářového okna grafického uživatelského rozhraní *MainForm*. Do tohoto okna budou podle potřeby vkládány stránky dědící z abstraktní třídy *WizardPageBase*, jež sama dědí prvky z třídy *UserControl* a implementuje rozhraní *IWizardPage*.

Diagram tříd popisující funkčnost uživatelského rozhraní je uveden na obrázku 3.1.



Obrázek 3.1: Návrh hlavních tříd uživatelského rozhraní

Instance formulářového okna *MainForm* obsahuje odkaz na aktuální stránku *currentPage*, pomocí níž tuto stránku ovládá. Metody *NavigateNext* a *NavigatePrevious* třídy *MainForm* řídí činnost stránek a posouvají je tím, že volají metody stránky *GetNextPage* či *GetPreviousPage*, které vrací hlavnímu oknu instanci odpovídající stránky. Metoda formuláře *ActivatePage* pak zobrazí vybranou stránku a předá jí řízení.

Metoda *PrepareContents* aktuální stránky slouží k její přípravě při posunu vpřed a je volána těsně před jejím zobrazením. Metoda *ProcessContents* je pak volána při opouštění stránky, díky čemuž má možnost zpracovat činnosti provedené uživatelem na stránce. Vlastnost *HasActivity* udává, zda na stránce probíhá automatický proces bez zásahu uživatele, například načítání databáze. Pokud ano, voláním metody *ExecuteActivity* z hlavního formuláře dojde ke spuštění této činnosti, po jejímž dokončení se vrátí řízení zpět pomocí události *ActivityFinished*. Podle úspěšnosti akce je pak volána metoda hlavního formuláře *NavigateNext*, nebo *NavigatePrevious*, pokud vykonání akce z nějakého důvodu nebylo úspěšné.

Tabulka 3.2 dává do souvislosti činnost jednotlivých stránek a navigaci mezi nimi.

Stránky, které načítají zdrojovou a cílovou databázi, porovnávají její strukturu a datový obsah a plánují a aplikují změny, jsou tzv. automatické a jejich činnost probíhá bez zásahu uživatele.

Číslo	Název	Činnost	Zpět	Další
1	IntroPage	Úvod	—	2
2	SourceDatabaseSelectionPage	Výběr zdrojové databáze	1	3
3	SourceDatabaseLoadingPage	Načtení zdrojové databáze	2	4
4	TargetDatabaseSelectionPage	Výběr cílové databáze	2	5
5	TargetDatabaseLoadingPage	Načtení cílové databáze	4	6
6	StructureComparisonPage	Porovnání struktury	4	7
7	StructuralDifferencesPage	Zobrazení strukturálních rozdílů	4	8
8	TableSelectionPage	Výběr tabulek pro porovnání dat	7	9
9	DataComparisonPage	Porovnání dat	8	10
10	DataDifferencesPage	Zobrazení datových rozdílů	8	11
11	ComparisonOverviewPage	Rekapitulace rozdílů	10	12
12	ChangesGainingPage	Plánování změn	8	13
13	ChangesSchedulePage	Zobrazení změn	8	14
14	ChangesApplicationPage	Aplikace změn	13	15
15	OutroPage	Konec	—	—

Tabulka 3.2: Přehled činnosti stránek průvodce a jejich navigačních vztahů

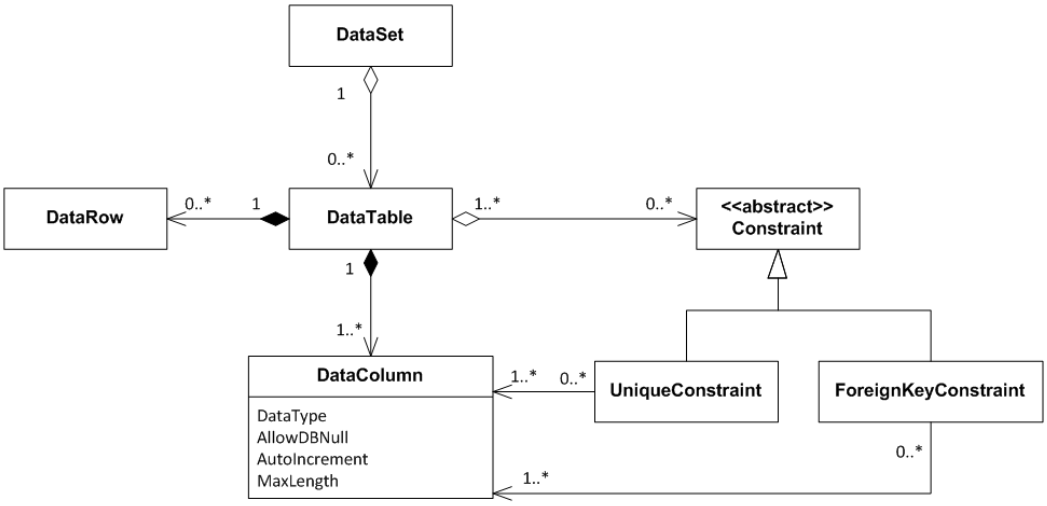
3.3 Uchování informací o struktuře a datovém obsahu databáze

Pro uchování informací o struktuře databází a jejich datového obsahu bude použita *sada tříd DataSet* knihovny pro přístup k datovým zdrojům ADO.NET. Její struktura, zobrazená na obrázku 3.2, velmi dobře koresponduje s požadavky, které jsou na ni kladeny. Další výhodou jejího využití je jednodušší integrace nástroje *Reflector*, který tuto knihovnu též využívá.

S ohledem na lepší srozumitelnost dalšího výkladu budou jako *zdrojový dataset* a *cílový dataset* označovány instance třídy *DataSet* obsahující načtenou strukturu a datový obsah zdrojové resp. cílové databáze.

3.4 Zaznamenání rozdílů

K zaznamenání rozdílů budou sloužit potomci abstraktních tříd *StructureDifference* pro strukturální rozdíly a *DataDifference* pro datové rozdíly. Obě tyto třídy implementují rozhraní *IDifference*, poskytující zejména popisek rozdílu v podobě vlastnosti *Caption* a informaci o tom, zda daný rozdíl již byl transformován na změnu. Třída

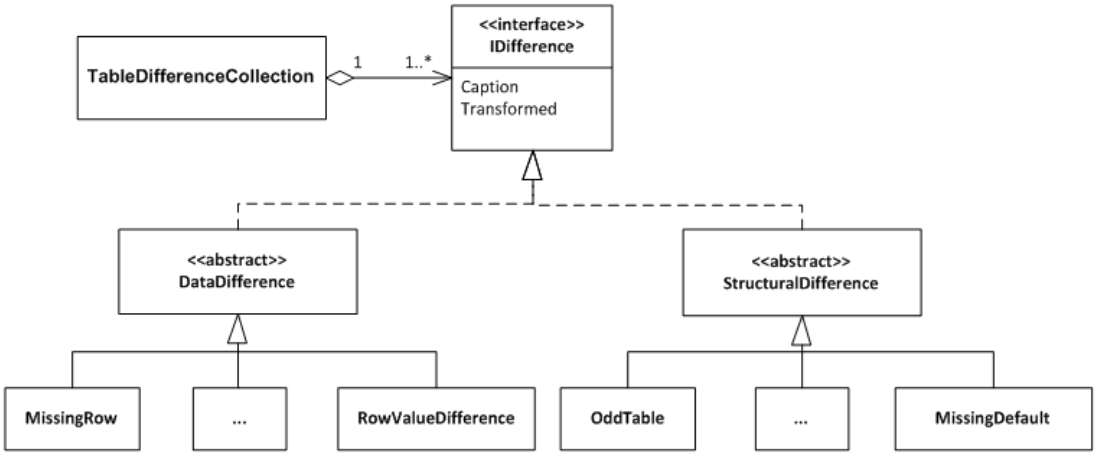


Obrázek 3.2: Statická struktura tříd pro uchování informací o struktuře databáze

TableDifferenceCollection reprezentující tabulku databáze pak uchovává tyto rozdíly v kolekci.

Pro zaznamenání rozdílů budou v maximální možné míře použity prvky ze zdrojového a cílového datasetu.

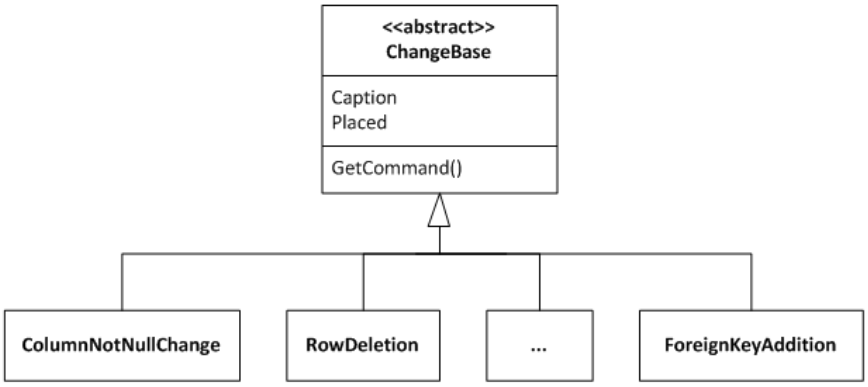
Odpovídající diagram tříd je pak znázorněn na obrázku 3.3.



Obrázek 3.3: Statická struktura tříd pro zaznamenání detekovaných rozdílů

3.5 Zaznamenání změn

Datová struktura uchovávající informace o změnách, jež mají být aplikovány na cílovou databázi, musí umožňovat jednoduchou transformaci změn na příkazy jazyka *SQL*, pokud možno v poměru 1:1. Tomuto požadavku odpovídá struktura naznačené na obrázku 3.4, kdy pro všechny vysoce specializované třídy existuje společná abstraktní třída *ChangeBase*, jež definuje metodu *GetCommand*. Děděné třídy pomocí této metody předávají příkaz jazyka *SQL*, pomocí nějž se na databázi provede požadovaná změna. Každá třída obsahuje údaje potřebné pro identifikaci prvku, na kterém má být změna provedena, a informace udávající jeho konečný stav.



Obrázek 3.4: Statická struktura tříd pro zaznamenání změn

3.6 Transformace rozdílů na změny

Proces transformace rozdílů na změny musí probíhat v takovém pořadí, v jakém budou později změny aplikovány na cílovou databázi, aby změna mohla nést informaci nejen o požadovaném stavu měněného prvku, ale i jeho aktuálního stavu v době aplikace změny, nutného pro správné určení aktualizovaného prvku. Tento požadavek v sobě nese nutnost při transformaci průběžně aktualizovat cílový dataset, což by mělo za následek ztrátu informace o jeho skutečné podobě v případě, že by převod na změny selhal a bylo by nutné vrátit se zpět. Mnohem nepříjemnějším problémem by však byla nutnost aktualizovat údaje uložené v ještě netransformovaných rozdílech.

Řešením tohoto problému se zdá být před započetím transformace vytvoření kopie cílového datasetu, který bude aktualizován tak, jak budou rozdíly transformovány na změny, zatímco původní cílový dataset zůstane nezměněn. Aby bylo možné přiřazovat prvkům cílového datasetu jejich aktualizovanou podobu v kopii, bude překladač obsahovat slovník v podobě instance třídy *Dictionary<TKey, TValue>*, jenž bude při každé transformované změně aktualizován.

Posledním nepříjemným problémem transformace je skutečnost, že některé rozdíly se

převádí na více změn, které mají proběhnout v jinou dobu. Například přidání sloupce zakazujícího hodnotu *NULL* do existující tabulky se záznamy je nutné provést ve dvou krocích; nejprve je potřeba přidat sloupec, jenž hodnotu *NULL* povoluje, aktualizovat záznamy v tabulce tak, aby bylo možné toto omezení aplikovat, a až poté dokončit transformaci. Jakékoli vkládání změn před jiné, které mají proběhnout dříve, by bylo velmi složité a zřejmě rizikové.

Vhodným řešením se zdá být využití více front pro vkládání vygenerovaných změn, které budou po skončení transformace složeny ve správném pořadí. Na úrovni databáze to jsou kategorie změn:

- počáteční změny – odebrání databázových objektů,
- tabulkové změny – změny v databázových tabulkách,
- konečné změny – smazání záznamů, jež dříve nemohly být smazány vč. přidání chybějících databázových objektů.

Po provedení počátečních změn bude překladač postupně vybírat takové tabulky, které bude možné transformovat, a provede překlad jejich rozdílů na změny. Pro tento účel bude tabulkový překladač obsahovat pět seznamů, do kterých bude přidávat odpovídající změny:

- počáteční strukturální změny – odebrání omezení, „polehčující“ změny sloupců,
- počáteční datové změny – nemazací změny,
- změny spojené s mazáním záznamů – aktualizace tabulek, jejichž záznamy se odkazují na mazané záznamy transformované tabulky,
- konečné datové změny – smazání záznamů,
- konečné strukturální změny – přidání omezení, „přitěžující“ změny sloupců,

Jelikož vlastnosti sloupců ani přítomnost omezení nemají vliv na charakterizování měněných prvků databáze, je možné přeložit v první fázi transformace i takové rozdíly, jejichž výstupem jsou změny, z nichž některé mají proběhnout před a jiné po provedení datových změn.

Nacházejí-li se v tabulce záznamy určené ke smazání, na které se odkazují záznamy z jiných tabulek, překladač po dokončení tvorby počátečních datových změn nechá transformovat potřebné tabulky a rozdíly, jež vzniknou překladem jejích rozdílů, přidá do seznamu změn spojených s mazáním.

Po úspěšném překladu všech přeložitelných rozdílů spojí seznamy do výsledné sekvence změn a předá ji překladači, který výběr tabulky inicioval – buď překladači databázových rozdílů, který ji přidá k tabulkovým změnám, nebo tabulkovému, který ji přiřadí do svého seznamu změn spojených s mazáním záznamů.

Kapitola 4

Implementace řešení

4.1 Základní přehled

V následujících podkapitolách jsou popsány některé poznatky z vývoje řešení aplikace. **??** uvádí příklad použití nástroje na testovacích databázích.

4.2 Využití sady tříd DataSet pro porovnání metadat

Základní datovou strukturou použitou pro práci s databázemi je dříve popsaný *DataSet*, jenž vyhovuje hlavním požadavkům kladeným na datovou strukturu potřebnou pro evidování struktury a dat relační databáze a umožňuje jednoduché načtení, správu a analýzu dat. Pro účely vyvíjené aplikace však bylo potřeba evidovat i některé údaje, které *DataSet* přímo nepodporuje.

V první řadě *DataSet* podporuje jen tato omezení:

- omezení primárního klíče,
- omezení cizího klíče,
- omezení *UNIQUE*.

Omezení *DEFAULT* v pojetí relačních databází sice má svůj ekvivalent ve vlastnosti sloupce *DataColumn*, tam však není možné zaznamenat jeho název a zejména může mít jen konstantní hodnotu. To znamená, že se nedá použít v případě, kdy požadovanou výchozí hodnotou je například volání funkce *GETDATE()* jazyka SQL, jež vrací aktuální datum. Omezení typu *CHECK DataSet* nepodporuje vůbec.

Z nabízejících se možností bylo zvoleno vytvoření virtuálních tabulek ve zdrojovém a cílovém datasetu, do kterých byly uloženy metainformace o těchto omezeních. Například pro omezení *CHECK* se jedná o tabulku *TEMP_Synchronizer_CheckConstraint* zaznamenávající jméno omezení, kontrolovanou podmínku a název tabulky a schématu,

do nějž je začleněna. Podobně je řešeno uložení informací u omezení *DEFAULT* a také pro ostatní databázové objekty. Výhodou tohoto řešení je jejich snadné porovnání a aktualizace, kdy dochází k obyčejnému porovnávání záznamů tabulek se stejnou strukturou. Možnou nevýhodou je pak požadavek, aby zdrojová ani cílová databáze neobsahovaly tabulku stejného názvu, jaký je použit pro tyto virtuální tabulky. Pro zmírnění rizika kolize byly proto vybrány názvy, které se u běžných tabulek nepředpokládají.

Vyhrazenými názvy tabulek jsou:

- *TEMP_Synchronizer_CheckConstraint*,
- *TEMP_Synchronizer_DefaultConstraint*,
- *TEMP_Synchronizer_DatabaseObject*.

Následující příklad zobrazený v tabulce 4.1 ukazuje blok kódu pro získání databázového příkazu pro vytvoření požadovaného omezení *DEFAULT*.

```
string command = string.Format(CultureInfo.InvariantCulture,
"ALTER TABLE {0}.{1} ADD CONSTRAINT {2} DEFAULT {3} FOR {4};",
constraint[DefaultListParserHelper.ModuleNameColumnName],
constraint[DefaultListParserHelper.TableNameColumnName],
constraint[DefaultListParserHelper.ConstraintNameColumnName],
constraint[DefaultListParserHelper.DefinitionColumnName],
constraint[DefaultListParserHelper.ColumnNameColumnName]);
```

Tabulka 4.1: Příkaz jazyka SQL pro vytvoření omezení DEFAULT z uložených metadat

4.3 Využití kolekce ExtendedProperties třídy DataColumn

Pro potřeby vyvíjeného nástroje bylo dále zapotřebí zjišťovat parametry sloupců udávajících přesnost a měřítko číselných datových typů, které třída *DataColumn* zastupující sloupec nepodporuje. Pro tyto účely byla využita její vlastnost *ExtendedProperties*, pomocí níž je možné rozšiřovat sadu definovaných vlastností.

V tabulce 4.2 je uvedena ukázka zápisu a porovnání údaje o měřítku sloupce.

4.4 Využití technologie LINQ

Při práci se seznamy a poli se osvědčila technologie LINQ, jež je dostupná na platformě Microsoft .NET Framework počínaje verzí 3.5. V kombinaci s tzv. *Lambda výrazy* v mnoha případech výrazně zjednodušila vyhodnocování podmínek nad množinou dat.

```
// Nastavení měřítka sloupce
{
  // ...
  DataColumn column = new DataColumn(columnName);
  column.ExtendedProperties.Add(NumericPrecisionKey, numericPrecision);
  // ...
}

// Porovnání měřítka sloupců zdrojové a cílové databáze
private static bool NumericPrecisionMatch(DataColumn source, DataColumn target)
{
  int sourceNumericPrecision = (int)source.ExtendedProperties[NumericPrecisionKey];
  int targetNumericPrecision = (int)target.ExtendedProperties[NumericPrecisionKey];
  return sourceNumericPrecision == targetNumericPrecision;
}
```

Tabulka 4.2: Použití vlastnosti ExtendedProperties pro uložení vlastností sloupců

Výhoda jejich využití se dá ilustrovat na následujícím příkladu, kdy je potřeba vybrat z množiny *optimalTables* takovou podmnožinu objektů typu *TableDifferenceCollection*, která neobsahuje žádnou vynucenou změnu typu *CleanTableRequest*. Možné konvenční řešení využívající podmínky a cykly *foreach* je v tabulce 4.3.

```
IList<TableDifferenceCollection> easyTables = new List<TableDifferenceCollection>();

foreach (TableDifferenceCollection table in optimalTables)
{
  bool canAdd = true;

  foreach (IDifference difference in table.Differences)
  {
    if (difference.ForcedChanges != null)
    {
      foreach (ChangeBase change in difference.ForcedChanges)
      {
        if (change is CleanTableRequest)
          canAdd = false;
      }
    }
  }

  if (canAdd)
    tables.Add(table);
}
```

Tabulka 4.3: Příklad využívající konvenční příkazy

S využitím LINQ dostáváme příkaz znázorněný v tabulce 4.4, který se dá relativně snadno rozluštit.

Porovnáním obou variant je patrné, že využití technologie LINQ s *Lambda výrazy* může zpočátku působit nezvykle, výsledný příkaz je však mnohem jednodušší, přehlednější a v

```
IList<TableDifferenceCollection> easyTables = optimalTables
  .Where(t => t.Differences
    .All(d => (d.ForcedChanges == null)
      || (d.ForcedChanges.All(c => !(c is CleanTableRequest)))))
  .ToList();
```

Tabulka 4.4: Příklad využívající LINQ a Lambda výrazy

konečném důsledku méně náchylný k chybě.

Kapitola 5

Závěr

Seznámil jsem se s problematikou modelem řízeného vývoje aplikací a pro jejich potřebu analyzoval, navrhl a implementoval nástroj, který umožňuje načíst schéma a vybraná data relačních databází, porovnat je a provést některé změny ve struktuře a datovém obsahu zadané databáze.

Při tomto úkolu jsem se soustředil zejména na rozbor problematiky porovnání s výhledem na budoucí rozšíření funkčnosti nástroje dle poznatků z jeho praktického využití. Aktuální verze nástroje splňuje požadavky kladené softwarovými projekty využívajícími generativní přístup pro správu, údržbu a rozvoj podnikových aplikací, pro něž byl tento nástroj primárně vyvíjen.

Literatura

[1] T. Stahl, M. Völter, *Model-Driven Software Development*. John Wiley & Sons, 2006.

[2] T. Conolly, C. Begg, R. Holowczak, *Mistrovství – Databáze, Profesionální průvodce tvorbou efektivních databází*. Computer Press, 2009.

[3] M. Jakubec, *Praktické využití konceptů generativního programování pro efektivní vývoj podnikových aplikací, Bakalářská práce*. FEL ČVUT, 2010.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Návrh programů pomocí vzorů - Stavební kameny objektově orientovaných programů*. Grada Publishing, 2003.

[5] J. Albahari, B. Albahari, *C# 4.0 in a Nutshell*. O'Reilly Media, 2010.

[6] A. Oppel, , *Databáze bez předchozích znalostí*. Computer Press, 2006.

[7] H. Kanisová, M. Müller, *UML srozumitelně*. Computer Press, 2006.

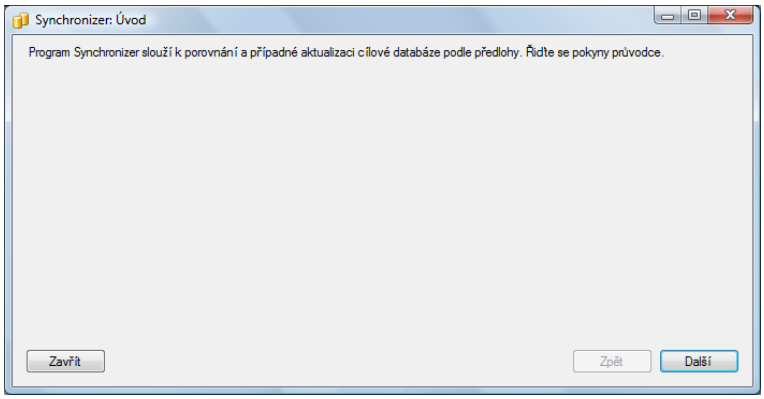
Příloha A

Používání nástroje

Ovládání nástroje probíhá pomocí průvodce. Uživatel se posouvá po jednotlivých stránkách pomocí navigačních tlačítek „Další“ a „Zpět“. V libovolném okamžiku má možnost činnost aplikace ukončit stisknutím tlačítka „Zavřít“.

Pro ukázku ovládání a činnosti nástroje byly zvoleny testovací databáze *Test_ Source* a *Test_ Target*, jež jsou k dispozici jako součást této práce na přiloženém CD.

Po spuštění programu je uživateli zobrazena úvodní stránka (viz obrázek A.1).

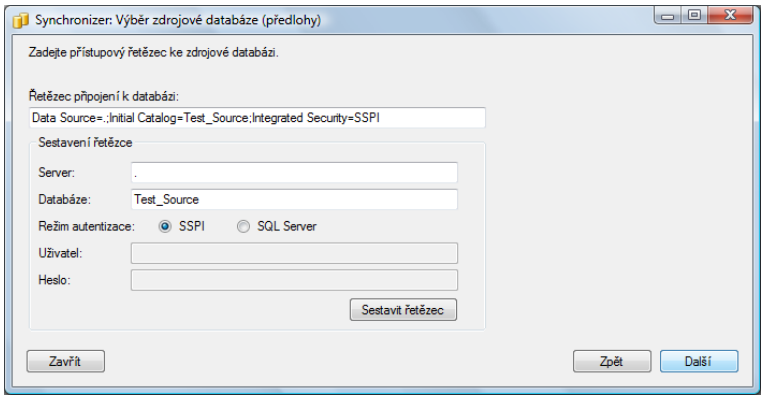


Obrázek A.1: Úvodní stránka průvodce

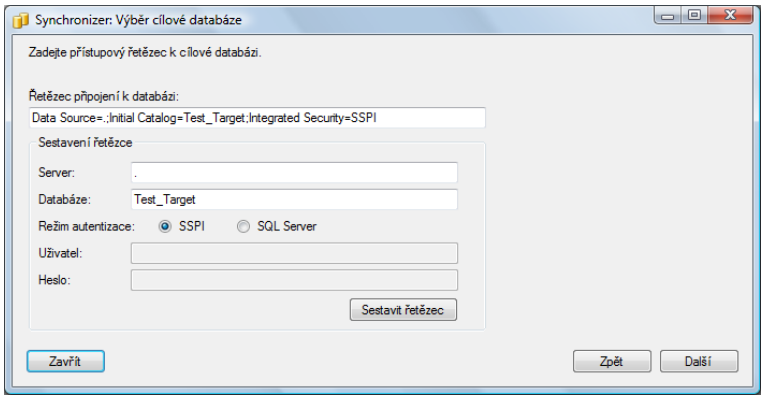
Pomocí tlačítka Další se přesuneme na stránku pro výběr zdrojové databáze. V řádku pro zadání řetězce připojení je již tato databáze předvyplněna. Uživatel má možnost změnit výběr buď přínou editací tohoto řetězce, nebo pomocí pomocníka pro jeho sestavení. Stiskem tlačítka Sestavit řetězec nástroj aktualizuje řetězec připojení podle uvedených údajů, jak je zobrazeno na obrázku A.2.

Po úspěšném připojení a načtení informací o zdrojové databázi nám nástroj zobrazí analogickou stránku pro výběr cílové databáze.

Na obrázku A.4 je ukázka stránky zobrazující seznam strukturálních rozdílů.



Obrázek A.2: Stránka pro zadání zdrojové databáze

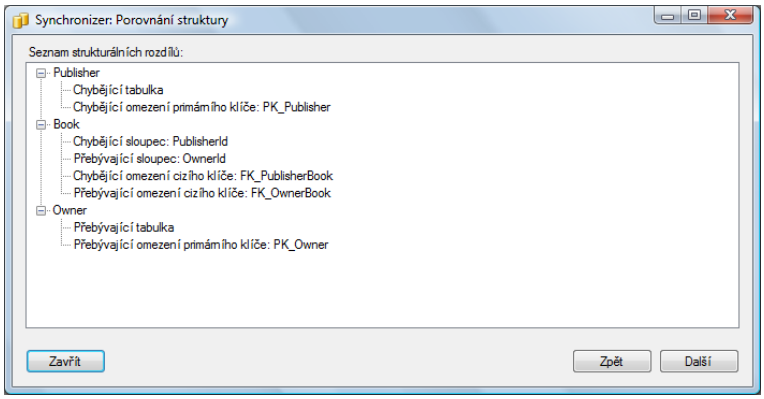


Obrázek A.3: Stránka pro zadání cílové databáze

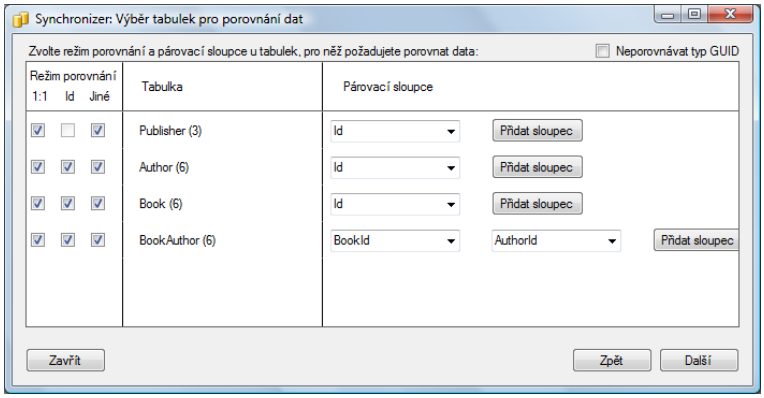
V dalším kroku jsme vyzváni k výběru tabulek pro porovnání dat, jak je znázorněno na obrázku A.5. Řekněme, že požadujeme uvést cílovou databázi do stavu přesně odpovídajícímu předloze, proto uvedeme párovací sloupce u všech tabulek a zvolíme úplné režimy porovnání.

Režim „1:1“ značí požadavek na shodu obsahu zdrojové a cílové tabulky, jeho označením umožníme smazat záznamy, jež nejsou obsaženy v předloze. Zvolením „Id“ požadujeme, aby hodnota primárního klíče pro spárované záznamy zdrojové a cílové tabulky byly stejné; tím, že jsme jako párovací sloupce ve všech případech zvolili sloupce nesoucí primární klíč, nemá tato volba vliv na výsledek porovnání. Povšimněme si, že nemáme možnost zvolit tento režim porovnání u tabulky Publisher, neboť se jedná o nově přidávanou tabulku. Zaškrtnutím volby Jiné pak požadujeme porovnat hodnoty záznamů i v ostatních sloupcích tabulky.

Na obrázku A.6 vidíme výsledek porovnání pro námi zvolené nastavení. Většinou se jedná o přidání nebo smazání záznamu, pouze v Tabulce *Book* dochází ke změnám hodnot ve sloupci PublisherId, neboť se jedná o nově přidávaný sloupec a my jsme povolili porovnání v režimu „Jiné“.



Obrázek A.4: Seznam detekovaných strukturálních rozdílů

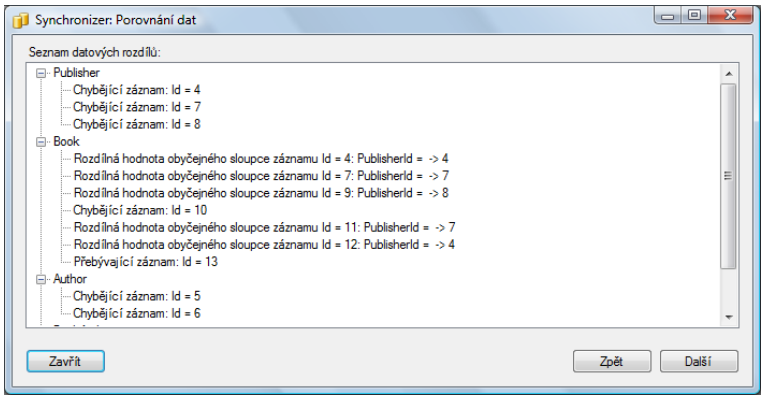


Obrázek A.5: Výběr tabulek pro porovnání dat

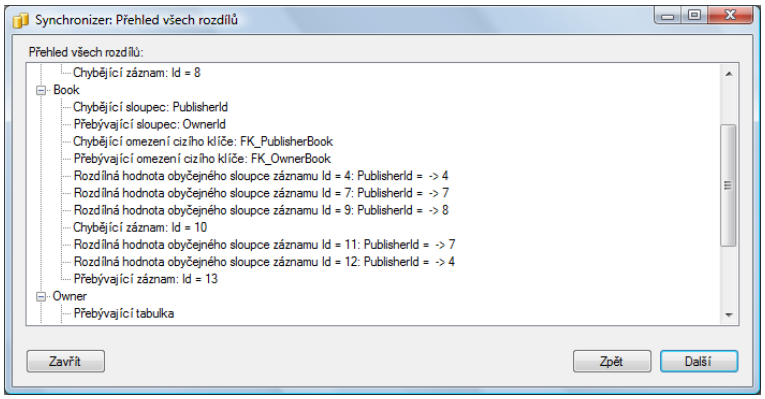
Poznámka: Pokud bychom v tabulce Book neporovnávali hodnoty ostatních sloupců, nástroj by se při provádění změn pokusil do nově přidaného sloupce všem záznamům nastavit hodnotu *NULL*. Jelikož však nastavení tohoto sloupce prázdnou hodnotu nedovoluje, nebylo by možné dokončit transakci a uživatel by byl informován o příčině problému. Hravý čtenář si takovou možnost jistě rád vyzkouší.

Jestliže jsou zobrazené rozdíly podle našich předpokladů, můžeme se posunout na další stránku rekapitulující všechny rozdíly (obrázek A.7). Po jejím potvrzení nástroj naplánuje provedení změn, jejichž seznam nám zobrazí na následující stránce, jak je vidět na obrázku A.8.

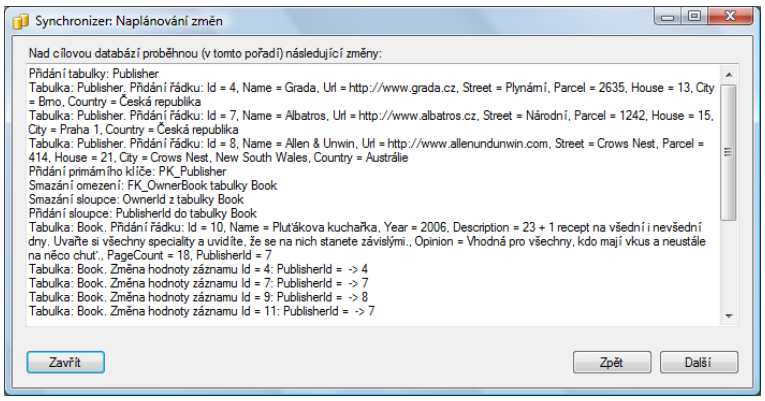
Zde naposledy můžeme zkontrolovat, že nástrojem navrhnuté změny na základě našeho výběru jsou žádoucí, a stisknutím tlačítka Další necháme nástroj aktualizovat cílovou databázi. V případě úspěchu se nám zobrazí poslední stránka průvodce, jež je znázorněna na obrázku A.9.



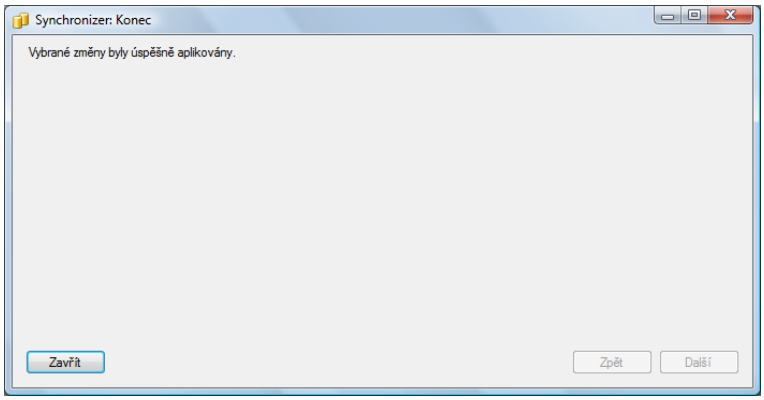
Obrázek A.6: Zobrazený seznam detekovaných datových rozdílů při daném nastavení porovnání



Obrázek A.7: Rekapitulace všech zjištěných rozdílů



Obrázek A.8: Pořadí navržených změn



Obrázek A.9: Poslední stránka průvodce

Příloha B

Obsah přiloženého CD

Informace jsou na datovém nosiči uspořádány do následujících složek:

- `thesis` – obsahuje dokument bakalářské práce ve formátu PDF,
- `source` – obsahuje zdrojový kód vyvíjené aplikace,
- `install` – obsahuje instalační program vyvíjené aplikace,
- `binary` – obsahuje spustitelné soubory vyvíjené aplikace,
- `example` – obsahuje skripty jazyka SQL pro založení testovacích databází.

Příloha C

Seznam použitých zkratek a pojmů

Cílová databáze	Zpravidla produkční databáze, jež má být porovnána a případně aktualizována podle zdrojové databáze.
Cílový dataset	Instance třídy DataSet reprezentující cílovou databázi.
Datově aktualizovaná tabulka	Tabulka, ve které již nemá dojít k přidání ani přesunu záznamu.
Hodnotový sloupec	Takový sloupec tabulky, který není identifikačním ani párovacím sloupcem.
Identifikační sloupec	Jeden nebo více sloupců tabulky s primárním klíčem.
Invazivní mazání záznamů	Nástrojem řízené smazání daného záznamu a všech záznamů jej odkazujících.
Kaskádní změna	Libovolná akce událostí ON DELETE a ON UPDATE cizího klíče.
LINQ	Langere Integrated Query, Integrovaný dotazovací jazyk platformy .NET Framework.
Nástroj	Vyvíjená aplikace.
Obyčejný sloupec	Viz Hodnotový sloupec.
Párovací sloupec	Jeden nebo více sloupců, podle nichž dochází k přiřazení záznamů mezi tabulkou zdrojové a cílové databáze.
Přesun záznamu	Změna hodnoty primárního klíče záznamu.
SQL	Structured Query Langure. Strukturovaný dotazovací jazyk pro komunikaci s relační databází.
Strukturálně aktualizovaná tabulka	Tabulka, ve které již nemá dojít k přidání primárního klíče.
Vynucená změna	Změna, jejíž provedení je bezprostředně vázáno na provedení jiné změny.
XML	Extensible Markup Langure, Standardizovaný rozšiřitelný značkovací jazyk.
Zdrojová databáze	Databáze sloužící jako předloha, podle níž dochází k porovnání a případně aktualizaci cílové databáze.
Zdrojový dataset	Instance třídy DataSet reprezentující zdrojovou databázi.

Tabulka C.1: Seznam použitých zkratek a pojmů