VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroj pro modelování relační databáze

Relational Database Modeling Tool

2017 Radek Svoboda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radek Svoboda**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro modelování relační databáze
Relational Database Modeling Tool

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit aplikaci s grafickým uživatelským rozhraním pro návrh schématu relační databáze formou E-R diagramu. Aplikace bude podporovat jak aktualizaci schématu databáze na základě změn v E-R diagramu, tak aktualizaci E-R diagramu podle existujícího schématu. Aplikace bude pracovat se SŘBD Microsoft SQL Server a Oracle Database.
Práce bude splňovat následující body:
1. Popis notace E-R diagramu.
2. Popis a srovnání stávajících CASE nástrojů pro modelování relačních databází.
3. Návrh a implementace grafického editoru E-R diagramů.
4. Návrh a implementace synchronizace E-R diagramů s relačním schématem.

Seznam doporučené odborné literatury:

[1] CHEN, Peter Pin-Shan. The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems (TODS), 1976, 1.1: 9-36.
[2] ULLMAN, Jeffrey D.; WIDOM, J. Database Systems: The Complete Book. 2000.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2016
Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce Ing. Petru Lukášovi za poskytnuté rady a čas strávený konzultacemi.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací softwarového nástroje pro návrh schématu relační databáze formou ER diagramu. Nástroj umožní uživateli nejen tvorbu nových schémat databáze, ale také vizualizaci již existujících schémat a jejich následné modifikace prostřednictvím změn ER diagramu v grafickém editoru. V první části se práce zabývá notací ER diagramu a srovnáním již existujících nástrojů určených k tvorbě ER diagramů. Druhá část se zaměřuje na návrh a implementaci jak grafického editoru, tak na problematiku synchronizace ER diagramu s existujícím relačním schématem.

Klíčová slova: relační databáze, ER diagram, relační schéma, CASE nástroj

Abstract

This thesis describes design and implementation of software tool for relational database schema design in form of an ER diagram. Tool is not limited only for creation of new database schemas, but it also provides visualization of already existing schemas and their subsequent modifications through changing ER diagram in graphic editor. The first part deals with ER diagram notation and the comparison of existing tools for the creation of ER diagrams. The second part focuses on the design and implementation of both the graphical editor, and on the synchronization of ER diagram with existing relational schema.

Key Words: relational database, ER diagram, relational schema, CASE tool

Obsah

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
DDL	–	Data Definition Language
DML	–	Data Manipulation Language
CASE	–	Computer-Aided Software Engineering
ERD	–	Entity-Relationship Diagram
SŘBD	–	Systém Řízení Báze Dat
DSD	–	Data Structure Diagram
EER	–	Enhanced Entity–Relationship Model
IDEF	–	Integration Definition for Information Modeling
ICAM	–	Integrated Computer-Aided Manufacturing
JSON	–	JavaScript Object Notation
XML	–	Extensible Markup Language
CSV	–	Comma-Separated Values
PDF	–	Portable Document Format
PNG	–	Portable Network Graphics
JDBC	–	Java Database Connectivity
JPEG	–	Joint Photographic Experts Group
GUI	–	Graphical User Interface
WPF	–	Windows Presentation Foundation
MVVM	–	Model-View-ViewModel
MVC	–	Model-View-Controller
XAML	–	Extensible Application Markup Language
BFS	–	Breadth-First Search
XPS	–	XML Paper Specification

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Strukturované uložení dat je jednou ze základních potřeb většiny vyvíjených aplikací. S rostoucím množstvím uchovávaných dat a potřebou jejich analýzy už dávno není dostačující prosté uložení do souboru. Na tyto potřeby reagují dnes ve velké míře používané relační databáze. Před samotným uložením dat je nutné definovat jejich strukturu, tedy datový model. Dnešní svět se velmi rychle mění a tyto změny často vyvolají potřebu upravit datový model tak, aby co nejlépe reflektoval modelovanou realitu. Právě pro tuto činnost jsou dnes stále častěji používané CASE nástroje, které umožňují snadnou změnu datového modelu prostřednictvím grafického uživatelského rozhraní bez nutnosti psaní vlastních DDL skriptů.

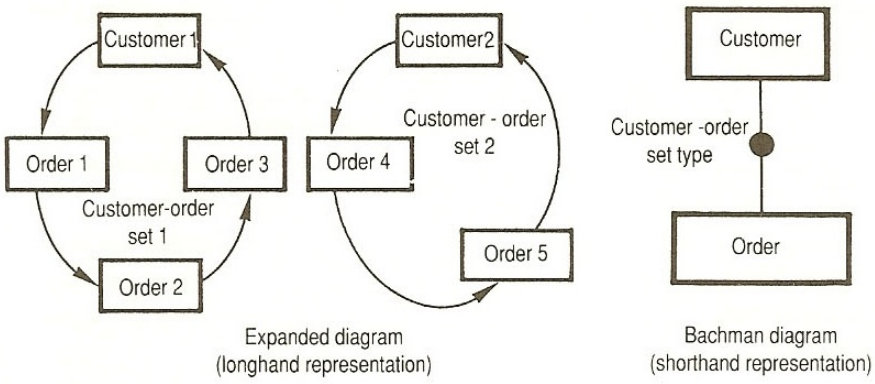
Tato bakalářská práce popisuje návrh a implementaci aplikace pro návrh a aktualizaci schématu relační databáze. Aplikace podporuje SŘBD Microsoft SQL Server a Oracle Database. První kapitola věnována popisu notace ER diagramu, který se v relačním schématu používá pro vizualizaci entit a vztahů mezi nimi nejčastěji. Kapitola 2 se zaměřuje na srovnání možností již existujících nástrojů pro tvorbu ER diagramu. Následuje kapitola 3, kde je popsán návrh a implementace grafického editoru ER diagramu, včetně algoritmů pro vyhledávání cest při vizualizaci vztahů. Poslední kapitola se zabývá synchronizací mezi ER diagramem a relačním schématem, zejména spoluprací použitých návrhových vzorů pro dosažení požadované funkcionality pro oba dříve zmíněné SŘBD.

2 Popis notace ER diagramu

V kontextu relačních databází jsou ER diagramy nejběžnějším způsobem vizualizace entit a jejich vzájemných vztahů. V průběhu let vznikla celá řada notací, které se liší nejen použitými grafickými symboly pro znázornění entit, kardinality a povinnosti vztahů, ale i dalšími vlastnostmi [3]. Dodnes nepředstavuje žádná z notací standard v oblasti modelování databází, což je nejvíce patrné na celé řadě CASE nástrojů, které často používají svou vlastní notaci či dokonce kombinaci symbolů z několika různých notací.

2.1 Historie

Za předchůdce ER diagramu jsou považovány diagramy datových struktur (DSD) [4], které rovněž slouží k zachycení entit a vztahů mezi nimi včetně vztažených omezení. Cílem DSD je graficky znázornit dekompozici složitých entit na jednotlivé elementy. K zachycení struktury dat se předpokládalo použití tzv. datových slovníků, které se dají zjednodušeně popsat jako množiny tabulek obsahující metadata. Z tohoto principu vychází i systémové katalogy moderních SŘBD.



Obrázek 1: Bachmanův diagram (Zdroj: [5])

Speciálním typem DSD je Bachmanův diagram, který slouží k vytvoření relačního datového modelu bez ohledu na způsob fyzického uložení dat v systému. Ukázku tohoto diagramu vidíme na obrázku 1, který zachycuje vztah mezi zákazníky (customer) a objednávkami (order). Tento diagram měl patrně největší vliv na pozdější vznik ER diagramu, který publikoval Peter Chen roku 1976 [1]. Největším rozdílem mezi DSD a ERD, opomineme-li grafickou podobu, je fakt, že DSD se zaměřuje na popis struktury složitých entitních typů a vztahy modeluje na úrovni jednotlivých složek entitních typů, zatímco ERD řeší vztahy mezi samotnými entitními typy.

2.2 Využití ER diagramu

Nejčastější použití ER diagramů představuje tvorba nových relačních schémat při návrhu databází. Rozhodně se ale nejedná o jedinou oblast využití. ER diagram velmi často stojí na počátku analýzy a návrhu mnoha informačních systémů využívajících relační databázi. Časté je i využití ER diagramu ve spojitosti s diagramy datových toků pro znázornění uložení dat business procesu. Diagramy můžeme dělit do více kategorií, viz kapitola 2.3.

Dalším velmi častým použitím ER diagramu je vizualizace již existujících schémat databáze, protože právě grafické znázornění schématu je ideální pro snadné odhalení logických chyb, ke kterým mohlo v rámci návrhu systému dojít. Vizualizace existujících schémat nemusí být použita pouze pro odhalení chyb, ale také při situaci, kdy je nutné datový model z různých důvodů upravit. ER diagram je tedy vhodný prostředek nejen k analýze stávajícího schématu pro návrh změn, ale také pro jejich přímé provádění prostřednictvím CASE nástrojů.

2.3 Kategorie modelů

Podle úrovně abstrakce rozlišujeme tři kategorie datových modelů [6], které využíváme při návrhu databázového schématu. Nejvyšší úroveň abstrakce nabízí *konceptuální model*, který zachycuje pouze entity a vztahy mezi nimi a je nejméně detailní. Konceptuální model není závislý na použitém SŘBD. Více detailů nabízí *logický model*, poskytuje nižší úroveň abstrakce, ale stále je nezávislý na použitém SŘBD. Posledním typem je *fyzický model*, ten vychází z logického modelu, nicméně se od něj může v mnoha ohledech lišit, protože jeho cílem je obsáhnout dostatečný počet technických detailů pro implementaci databáze. Rozdíly mezi logickým a fyzickým modelem způsobuje fakt, že fyzický model je vytvářen s ohledem na konkrétní použitou technologii. Proto i v případě, že zvolíme relační databázi, se mohou pro různé SŘBD fyzické modely více či méně lišit. Přehledné shrnutí rozdílů mezi modely nabízí tabulka 1 (znak ✓značí, že daná položka je součástí modelu).

Tabulka 1: Rozdíly mezi modely

Modelované vlastnosti	Konceptuální	Logický	Fyzický
Názvy entit	✓	✓	✗
Vztahy	✓	✓	✗
Atributy	✓	✓	✗
Primární klíče	✗	✓	✓
Cizí klíče	✗	✓	✓
Názvy tabulek	✗	✗	✓
Názvy sloupců	✗	✗	✓
Datové typy sloupců	✗	✗	✓

2.4 Vývoj notací

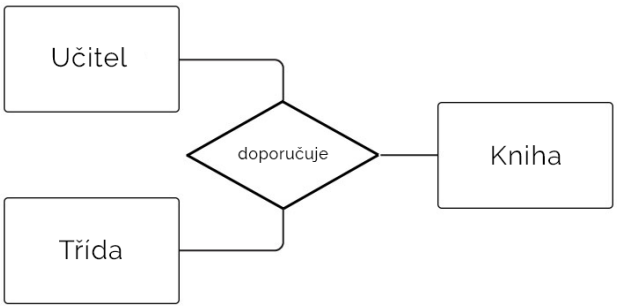
Stejně, jako se vyvíjely potřeby návrhu datových modelů, vznikaly i nové notace ER diagramu. Zásadním mezníkem pro vývoj notací byl objektově orientovaný přístup k vývoji softwaru, na který se mnohé notace snažily reagovat. Tento trend způsobil rozšíření notací ER diagramu o koncept generalizace/specializace a dědičnosti, tento typ modelu se označuje jako *Enhanced entity–relationship model* (EER). Vlivem popularity objektově-relačních databází jsou některé notace často modifikovány tak, aby umožňovaly zahrnout v modelu metody a operace entit. Tento přístup je dnes k vidění v celé řadě CASE nástrojů a je vhodný zejména pro modelování komplexních databází pro geografické informační systémy nebo telekomunikace.

2.5 Vlastnosti notací

Na první pohled mohou různé notace vyvolávat dojem, že nejvýraznějším rozdílem mezi nimi je grafické znázornění entit a vztahů, nicméně rozdílů je mnohem více a některé mohou mít vliv na výslednou podobu fyzického modelu databáze.

2.5.1 Binární a *n*-ární modely

Nejzásadnějším rozdílem mezi notacemi je podpora n-árních modelů. Z toho důvodů rozlišujeme dva typy modelů: binární a n-ární [3]. Pro kategorii binárních modelů je typické, že každý objekt, který má alespoň jeden atribut, je považován za entitu. Není tedy možné přiřadit atribut vztahu, jak to můžeme běžně vidět u Chenovy notace. Tento fakt je nejvíce patrný u vztahu M:N, kterému chceme přiřadit atribut pro zaznamenání dalších informací. Přidání neklíčového atributu způsobí nutnost přidat do modelu asociační entitu pro tento vztah.



Obrázek 2: Příklad ternárního vztahu

Modely umožňující n-ární vztahy mohou určité situace popisovat výstižněji než jejich rozklad na vztahy binární. V případě n-árních modelů není nutné vytvářet další entity k přidání atributů pro vztah. Při přechodu na modely nabízející nižší úroveň abstrakce, je nutné n-ární vztahy převést na sekvenci binárních vztahů, což může vést ke ztrátě cenných informací, protože už pro ternární vztah M:N:P může způsobů dekompozice existovat více a na první pohled nemusí

být ztráta části informace patrná. Na obrázku 2 vidíme diagram modelující situaci, kdy učitel doporučuje knihy třídám, nicméně převod na sekvenci binárních vztahů může znemožnit zjištění informací o tom, kdo z učitelů doporučil danou knihu.

2.5.2 Kardinalita a povinnost

Vyjádření kardinality a povinnosti vztahu je další z mnoha odlišností notací ER diagramu. Pro vyjádření kardinality se používá buď grafické znázornění, nebo označení číslem na obou stranách vztahu v případě starších notací, např. Chenova [2]. Povinnost bývá vyjádřena nejčastěji graficky, buď stylem čáry nebo určitým symbolem na stranách vztahu. Speciálním vyjádřením je dvojice (min, max), která vyjadřuje jak kardinalitu, tak povinnost. Jednotlivé notace se liší také tím, na které straně je informace vyjádřena. Rozlišujeme proto dva styly zápisu: *look here* a *look across* [3].

Pokud modelujeme situaci, ve které jeden zaměstnanec pracuje právě v jednom oddělení a oddělení může mít několik zaměstnanců, v look across notaci zapíšeme 1 na stranu oddělení a N na stranu zaměstnance. Pro notaci look here tomu bude přesně naopak.

2.6 Chenova notace

Jedná se o notaci představenou poprvé roku 1976 Peterem Chenem [1] a dodnes se jedná o jednu z nejvíce používaných. Tato notace podporuje n-ární vztahy. Původní specifikace obsahovala prvky jen pro entity, vztahy, včetně kardinality, a atributy. Později byla rozšířena o rozlišení povinnosti vztahu a koncept generalizace/specializace. Pro povinnost se používá styl look here, pro kardinalitu look across. Na obrázku 3 vidíme příklad použití této notace. V této části jsou rovněž popsány prvky používané při tvorbě ER diagramu. Na další notace a jejich rozdíly vhledem k této notaci se zaměřuje kapitola 2.7.

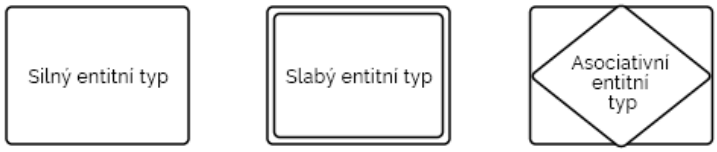
Obrázek 3: Příklad Chenovy notace (Zdroj: [6])

2.6.1 Entitní typ

Entitní typy (často označovány v souvislosti s ERD jen jako entity) jsou jednoznačně identifikovatelné typy objektů vyskytující se v problémové doméně. Obvykle se označují podstatným jménem v jednotném čísle. Rozlišujeme následující typy entit:

- **Silný entitní typ** může existovat nezávisle na ostatních entitních typech, protože obsahuje alespoň jeden atribut, který entitu jednoznačně odlišuje od ostatních, značí se obdélníkem (viz obrázek 4a).

- **Slabý entitní typ** jehož existence je závislá na jiném entitním typu, protože neobsahuje atribut umožňující jednoznačnou identifikaci, značí se obdélníkem s dvojitým okrajem (viz. obrázek 4b).
- **Asociativní entitní typ** slouží k vyznačení vztahu mezi entitami, obsahuje atributy identifikující tento vztah, značí se kosočtvercem uvnitř obdélníku (viz obrázek 4c).



(a) Silný entitní typ (b) Slabý entitní typ (c) Asoc. entitní typ

Obrázek 4: Znázornění entit podle Chenovy notace

2.6.2 Vztah

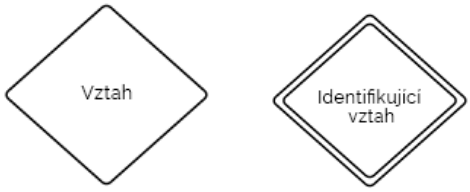
Vztah slouží k vyjádření asociace mezi entitními typy. Obecně vyjadřuje informace, které nelze vyjádřit pouhými entitními typy, např. student *studuje* předmět, novinář *napsal* článek. Označuje se slovesem. Znázorňuje se kosočtvercem (viz obrázek 5a).

Speciálním typem vztahu je identifikující vztah, který spojuje slabý entitní typ s entitním typem, na kterém je závislý, ten se označuje jako vlastník. Značí se kosočtvercem s dvojitým okrajem (viz. obrázek 5b). Podle počtu entit, které se v nich vyskytují rozlišujeme tyto typy:

- **Unární vztah**, ve kterém se vyskytuje pouze jedna entita, ta je ve vztahu sama se sebou (např. zaměstnanec *je nadřízeným* jiného zaměstnance).
- **Binární vztah** existuje mezi dvěma entitami (např. zaměstnanec *je vedoucím* oddělení).
- **Ternární vztah** existuje mezi třemi entitami najednou (např. učitel *doporučuje* knihu třídě).
- **N-ární vztah** je mezi n entitami zároveň. Jedná se o zobecnění předchozích uvedených vztahů.

Nejčastějšími typy vztahů jsou vztahy unární a binární. N-ární vztahy, kde $n > 2$, se vyskytují v ER diagramech velmi málo, protože vedou k problémům s dekompozicí na několik binárních vztahů při přechodu z konceptuálního modelu na logický.

Vztahy dále dělíme podle kardinality, která udává, kolikrát může instance entitního typu být ve vztahu s instancí jiného entitního typu, pro binární typ vztahu rozlišujeme kardinalitu 1:1, 1:N, M:N. Pro ternární typ vztahu rozlišujeme kardinalitu 1:1:1, 1:1:N, 1:N:M, a M:N:P, pro



(a) Vztah (b) Identifikující vztah

Obrázek 5: Znázornění vztahů podle Chenovy notace

n-ární typy vztahu se typy kardinality odvozují analogicky. Poslední důležitou vlastnosti vztahu je povinnost, určující zda může instance entitního typu existovat bez vztahu k jiné instanci entitního typu. Rozlišujeme povinné a nepovinné vztahy.

2.6.3 Atribut

Atributy jsou vlastnosti charakterizující entitní typ. Označují se podstatným jménem, znázorňují se elipsou. Rozlišujeme následující kategorie:

- **Jednoduchý atribut**, nelze dále dělit (např. *značka automobilu*, viz obrázek 6a).
- **Složený atribut** se dělí na více atomických atributů (např. *adresa* se dělí na *město, ulici a PSČ*, viz. obrázek 6b).
- **Odvozený atribut**, jehož hodnota je vypočtena na základě hodnot ostatních atributů. Fyzicky nemusí v relační databázi existovat, znázorněn je elipsou s čárkovanou hranou (např. *průměrná mzda*, viz obrázek 6c).

(a) Jednoduchý atribut (b) Složený atribut (c) Odvozený atribut

Obrázek 6: Znázornění atributů podle Chenovy notace

Další rozdělení je podle počtu hodnot atributu na:

- **Jednohodnotové** atributy obsahují pouze jednu hodnotu (např. *rodné číslo*).
- **Vícehodnotové** atributy mohou obsahovat více hodnot, značí se elipsou s dvojitým okrajem (např. *telefonní číslo na pevnou linku i mobilní telefon.*

).

Označení atributů lze kombinovat, např. *jednohodnotový odvozený atribut.*

2.7 Často používané notace

Jak už bylo zmíněno, notací existuje celá řada. Jednotlivé notace se v mnoha směrech liší. V této kapitole jsou vybrané notace popsány včetně příkladů. Všechny příklady modelují situaci, kdy jeden člověk má právě jedno místo narození a na jednom místě narození se mohlo narodit více lidí.

2.7.1 Bachmanova notace

Tento typ notace podporuje nejvýše binární vztahy, vyjádření kardinality používá styl look across, povinnost pak look here, stejně jako je tomu u Chenovy notace. Bachmanova notace [3] modeluje cizí klíče už na konceptuální úrovni, její předpokládané použití je tedy modelování relačních databází. Entity jsou znázorněny obdélníky, vztahy čárami. Vztahům není možné přiřadit atributy, pokud potřebujeme u vztahu M:N přidat neklíčový atribut, je nutná dekompozice pomocí asociační entity. Povinnost vztahu vyjadřuje kruh bez výplně, pro nepovinné vztahy, povinný vztah je naopak vyjádřen plným černým kruhem. Povinnost je vyjádřena vždy na konci vztahu. Pokud čára končí šipkou, jedná se o kardinalitu N, 1 se značí pouhou čárou bez symbolu. Příklad diagramu v Bachmanově notaci vidíme na obrázku 7.

Obrázek 7: Příklad Bachmanovy notace (Zdroj: [6])

2.7.2 Crow's foot notace

Tato notace se označuje také jako Information engineering nebo Martinova notace [3]. Opět podporuje nejvýše binární vztahy, které jsou znázorněny čárami. Není možné přiřazovat atributy vztahu bez asociační entity. Kardinalita i povinnost vztahu používá look across styl a obojí je znázorněno graficky. Entity jsou znázorněny obdélníky, do kterých jsou kromě samotného názvu vepsány také všechny atributy. Zajímavé je, že ačkoli každá notace používá své grafické symboly, tak znaky pro znázornění kardinality a povinnosti použité touto notací najdeme velmi často v kombinaci s jinými notacemi, jako je např. IDEF1X (viz kapitola 2.7.5). Tato notace je populární zejména v oblasti CASE nástrojů. Příklad použití je vyobrazen na obrázku 8.

Obrázek 8: Příklad Crow's foot notace (Zdroj: [6])

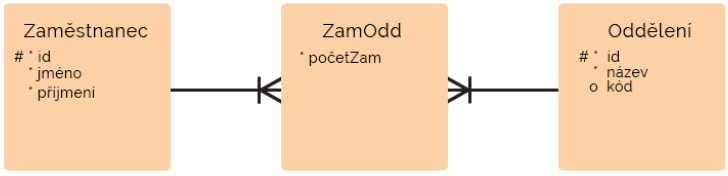
2.7.3 Barkerova notace

Barkerova notace [3] je velmi populární v oblasti tvorby datových modelů pro Oracle Database. Vznikla roku 1981 a po příchodu Richarda Barkera do společnosti Oracle se stala v této oblasti velmi populární. Jedná se o notaci podporující nejvýše binární vztahy, které jsou vyjádřeny čárami. Plná čára symbolizuje povinný vztah, čárkovaná pak vztah nepovinný. Pro vyjádření kardinality je zde použít také symbol vraní nohy jako u Crow's foot notace (viz kapitola 2.7.2). Povinnost používá styl look here, kardinalita pak styl look across. Ukázku této notace vidíme na obrázku 9. Atributy jsou zde trojího typu, každý typ označuje kategorii atributu. Atributy se uvádí volitelně. Notace používá tyto znaky pro kategorizaci atributů:

- # pro vyznačení identifikujícího atributu.
- ∗ pro vyznačení povinného atributu.
- ○ pro vyznačení nepovinného atributu.

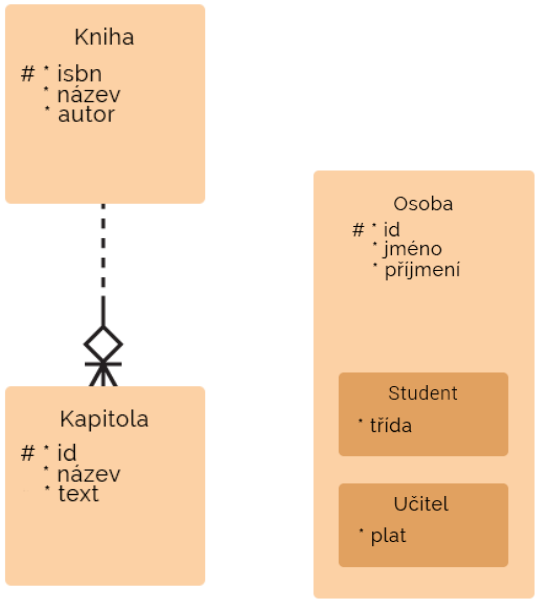
Obrázek 9: Příklad Barkerovy notace [6]

Oproti ostatním notacím zde nalezneme určitá specifika. Prvním z nich je tzv. UID čára (viz obrázek 10), která se kreslí pouze u slabých entitních typů a vyjadřuje, že je identifikován atributy silných entitních typů.



Obrázek 10: Specifika Barkerovy notace - UID

Dalším specifikem, je možnost vyznačit nepřenositelný vztah (viz obrázek 11a). Pokud jej použijeme, už jej nelze později změnit. Příkladem je vztah mezi kapitolou a knihou, kdy kapitolu z jedné knihy nemůžeme přiřadit knize druhé. Graficky se tento typ vztahu značí přidáním kosočtverce k vraní noze.



(a) Nepřenositelný vztah (b) Podtypy

Obrázek 11: Specifika Barkerovy notace

Notace podporuje také dědičnost, kdy podtypy dědí atributy nadřazeného typu (viz obrázek 11b). Graficky je tento jev vyjádřen přidáním obdélníků jednotlivých podtypů do společného obdélníku, který odpovídá nadřazenému typu.

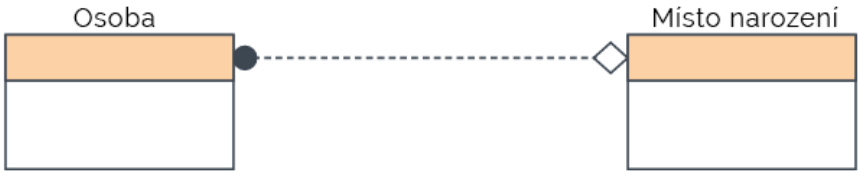
2.7.4 Min-Max notace

Jedná se o notaci, která podporuje nejvýše binární vztahy. Pro vyjádření kardinality a povinnosti vztahu, zde není použit žádný grafický symbol, ale uspořádaná dvojice (min, max), kde minimální počet instancí entitního typu (min), které musí ve vztahu participovat, určuje povinnost, maximální počet (max) určuje kardinalitu. Styl pro kardinalitu a povinnost se používá jednotný, zpravidla look here (viz obrázek 12).

Obrázek 12: Příklad Min-Max notace [6]

2.7.5 IDEF1X notace

IDEF1X [3] je notace, která vnikla v rámci programu ICAM financovaném US Air Force, jako součást IDEF technik pro modelování informačních systémů. Vztahy jsou podporované nejvýše binární a neklíčové atributy vztahu je nutné modelovat použitím asociační entity. Kardinalita a povinnost vztahu je vyjádřena pomocí (min, max) notace buď graficky nebo textově. Nejčastěji používá styl look across. Tato notace rozlišuje graficky silné a slabé entitní typy. Podobně jako u Barkerovy notace je zde možné vyznačit identifikující vztahy. Navzdory tomu, že notace nabízí vlastní grafické symboly pro znázornění kardinality a povinnosti vztahu, je často použita pro tento účel část Crow's foot notace (viz kapitola 2.7.2), zbytek prvků IDEF1X je ponechán. Ukázku použití zachycuje obrázek 13.



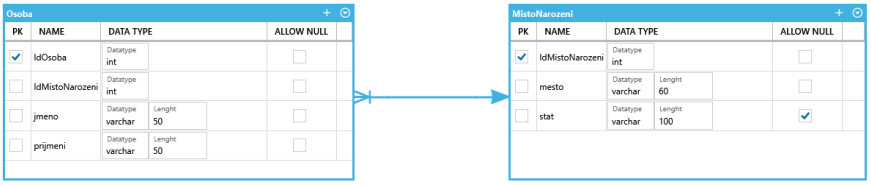
Obrázek 13: Příklad IDEF1X notace [6]

2.8 Nevýhody ER diagramů

ER diagramy jsou určeny primárně pro vytváření modelů relačních struktur. Pokud pracujeme s nestrukturovanými daty, které není jednoduše možné reprezentovat relacemi, není ER diagram vhodným vizualizačním nástrojem. Toto omezení platí i pro částečně strukturovaná data, jako např. XML nebo JSON. Navzdory tomu, že jsme schopni rozlišit jednotlivé entitní typy, může dojít k situaci, kdy se entity téhož typu liší svými atributy.

2.9 Notace použitá ve vyvíjeném nástroji

Nástroj je určen pro návrh a správu relačních schémat v rámci již fyzicky existujících databází. Pro tuto činnost je používán výhradně fyzický model, protože je nutné pracovat s konkrétními datovými typy daného SŘBD a integritními omezeními v podobě primárních a cizích klíčů. Z tohoto důvodu byla zvolena Oracle CASE notace, která vychází z Barkerovy notace. Z důvodu práce na nejnižší úrovni abstrakce je tato notace modifikována. Kardinalitu i povinnost vyjadřuje notace graficky. Styl look-here pro povinnost a look-across pro kardinalitu je z původní notace zachován. Vztahy jsou podporovány pouze unární a binární, modelování podtypů umožněno není. Ukázku vidíme na obrázku 14.



Obrázek 14: Ukázka notace použité ve vyvíjeném nástroji

2.10 Shrnutí

Každá notace má své klady i zápory. Volba notace by měla vycházet z potřeb pro tvorbu konkrétního modelu. Největší rozdíl mezi notacemi představuje možnost modelování ternárních vztahů, případně i vztahů s vyšší aritou. Proto je nutné zvážit zda sémantické možnosti binárních notací jsou pro vytvářený model dostatečné.

Znázornění kardinality a povinnosti se u jednotlivých notacích liší spíše graficky a všechny zmíněné notace poskytují v této oblasti takřka identické možnosti. Mezi poslední parametry, které mohou volbu notace ovlivnit, řadíme možnost modelování cizích klíčů na konceptuální úrovni. Tímto sice dojde k snížení úrovně abstrakce, nicméně konverze na fyzický model je poté jednodušší. Posledním kritériem je možnost modelování odvozených typů a nepřenositelných vztahů. Toto kritérium, ale nabídku notací velmi omezí. Přehledné srovnání vybraných notací nabízí tabulka 2.

Tabulka 2: Srovnání vybraných notací

Vlastnost	Chen	Bachman	Crow's foot	Berker	Min-Max	IDEF1X
Ternární vztahy	✓	✗	✗	✗	✗	✗
Look-across kardinalita	✓	✓	✓	✓	✗	✓
Look-across povinnost	✗	✗	✓	✗	✗	✓
(Min, Max) notace vztahů	✗	✗	✓	✗	✓	✓
Neklíčové atributy vztahů	✓	✗	✗	✗	✗	✗
Cizí klíče na koncept. úrovni	✗	✓	✗	✗	✗	✓
Podtypy	✓	✓	✓	✓	✓	✗

3 Popis a srovnání stávajících CASE nástrojů pro modelování relačních databází

V dnešní době je využití CASE nástrojů v mnoha fázích softwarového procesu de facto standardem. Jinak tomu samozřejmě není ani u návrhu datových modelů. Na trhu nalezneme celou řadu nástrojů od komerčního softwaru po open-source projekty podporující tvorbu ER diagramů. Tyto nástroje z pravidla nenabízí pouze grafický editor pro vytváření těchto diagramů pro programovou dokumentaci, ale obsahují řadu dalších užitečných funkcí, jako je např. generování patřičných DDL skriptů.

3.1 Výhody a nevýhody CASE nástrojů pro modelování relačních databází

Jednou z největších výhod je urychlení vývoje rozsáhlých projektů, protože spousta činností lze použitím CASE nástrojů automatizovat. Mnoho z nástrojů umožňuje také vygenerování ER diagramu z existující databáze nebo na základě DDL skriptu. Stejně tak lze některé nástroje využít k tvorbě objektově-relačního mapování pro přístup k datovému zdroji, což ušetří mnoho času a práce.

Použití CASE nástrojů s sebou nenese jen samá pozitiva. Největším problémem je obvykle nulová podpora pro provedení formální analýzy vytvořeného modelu, nástroje plně spoléhají na odborné znalosti uživatele. Dalším problémem je časová investice nutná k naučení se efektivně pracovat s daným nástrojem. Tento problém se netýká pouze modelování relačních databází, ale projevuje se u většiny specializovaných nástrojů. Za poslední nevýhodu považuji cenu komerčních nástrojů, protože zejména u menších projektů nemusí být využity veškeré možnosti, které daný produkt nabízí a týmové licence mohou být velmi drahé. Řešením je použití freeware nástrojů, případně tzv. community verze některého z nástrojů, pokud to licenční politika dovolí.

3.2 Srovnání stávajících nástrojů

Cílem práce je vytvoření CASE nástroje, který podporuje Oracle Database a Microsoft SQL Server. Právě podpora SŘBD se stala primárním požadavkem při volbě porovnávaných nástrojů. Každý z následujících nástrojů tedy podporuje alespoň jeden z dvojice uvedených SŘBD. Každý z následujících nástrojů nabízí alespoň zkušební verzi pro otestování jeho funkcionality. V ideálním případě je možné pořídit licenci pro studenty zdarma nebo využít bezplatné community edice. Odkazy na oficiální webové stránky testovaných CASE nástrojů a zmíněných SŘBD jsou k dispozici v příloze C.

3.2.1 Oracle SQL Developer Data Modeler

Autorem prvního nástroje je firma Oracle, která jej vyvíjí od roku 2008. Data Modeler je šířen bezplatně a nainstalovat jej můžeme buď samostatně nebo v rámci nástroje SQL Developer.

3.2.1.1 Popis funkcionality

Data Modeler nabízí tvorbu modelů na dvou úrovních abstrakce: logický model a relační (fyzický) model. Výhodou je možnost vygenerování několika oddělených relačních modelů z jediného logického pro vizualizaci možných schémat před samotným nasazením. V nabídce je trojice notací pro logický model. Implicitně je použita Barkerova notace (viz kapitola 2.7.3), která je v oblasti Oracle databází nejpoužívanější, nicméně ji lze přepnout na Bachmanovu (viz kapitola 2.7.1) nebo Information engineering (viz kapitola 2.7.2). Oproti jiným nástrojům tuto možnost volby oceňuji, protože si každý uživatel může notaci nastavit podle svých preferencí. Pro relační model už možnost volby neexistuje a je použita modifikovaná Barkerova notace, označována někdy také jako Oracle CASE notace.

Diagramy je možné exportovat ve formátu PDF, případně ve formě PNG nebo JPEG obrázku. Pro samotný relační model je možností exportu hned několik. Pokud nechceme použít nativní formát nástroje Data Modeler, můžeme exportovat do CSV. V tomto případě vznikne hned několik CSV souborů, rozdělených podle jednotlivých prvků (tabulek, vztahů apod.) diagramu. Dalšími možnostmi je XMLA nebo Cube View Metadata, používané v kombinaci s DB2 (viz výpis 1).

```
<xs:complexType>
   <xs:sequence>
      <xs:element name="idOdd" msdata:ReadOnly="true" msdata:AutoIncrement
         ="true" msprop:FriendlyName="idOdd" msprop:DbColumnName="idOdd"
         msprop:DataSize="-1" type="xs:integer" />
      <xs:element name="nazev" msdata:ReadOnly="true" msdata:AutoIncrement
         ="true" msprop:FriendlyName="nazev" msprop:DbColumnName="nazev"
         msprop:DataSize="100" type="xs:VARCHAR " />
      <xs:element name="kod" msdata:ReadOnly="true" msdata:AutoIncrement="
         true" msprop:FriendlyName="kod" msprop:DbColumnName="kod"
         msprop:DataSize="4" type="xs:CHAR " />
   </xs:sequence>
</xs:complexType>
```

Výpis 1: Příklad části exportu do XMLA

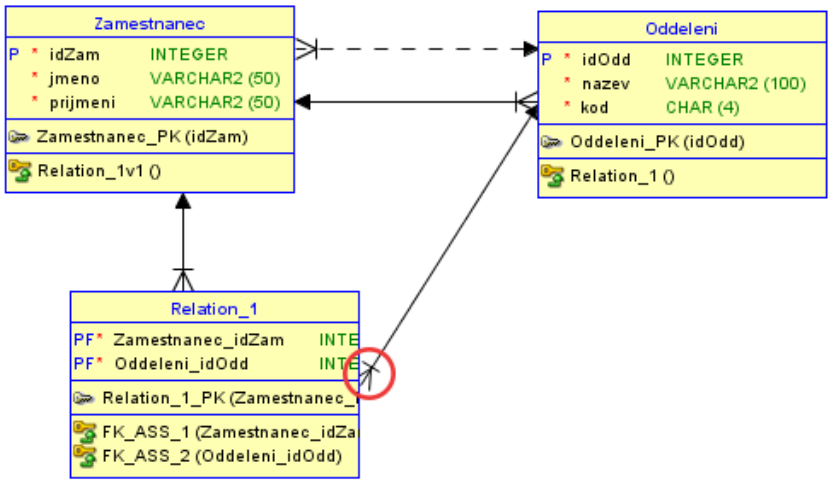
Nástroj podporuje vygenerování DDL skriptu na základě relačního modelu. Před exportem je nutné nastavit požadovaýé SŘBD. V nabídce je Oracle Database, Microsoft SQL Server a IBM DB2, všechny zmíněné jsou podporovány v několika verzích. Exportování DDL skriptu je doprovázeno kontrolou chyb, což napomáhá k odhalení problémů ještě před samotným spuštěním skriptu.

Možností importu je opět několik. Importovat lze nejen z CSV, Data Cube metadat, XMLA nebo nativní formát nástroje Data Modeler, ale také přímo DDL skript. Tato možnost je výhodná v případě vizualizace již existujících schémat. Ta nejvíce praktickou ovšem považuji možnost importu pomocí data dictionary, které umožňuje připojení k existující databázi přes JDBC. Tento způsob importu je možné využít nejen pro Oracle, ale také pro Microsoft SQL Server a IBM DB2 (viz příloha C).

Po importu a reverzním vytvoření diagramu nástroj umožňuje rovněž aktualizaci schématu databáze na základě změn v ER diagramu. Změny se neprovádějí ihned po provedení, ale synchronizaci je nutné spustit ručně. Tento způsob implementace umožňuje nepotvrzené změny jednoduše vrátit zpátky synchronizací diagramu s aktuálním schématem databáze.

3.2.1.2 Popis grafického editoru

Grafický editor reaguje při manipulaci s objekty poměrně rychle, nicméně při přesunu entity nepříjemně problikávají. Vlivem otočení jsou špatně rozlišitelné symboly pro znázornění kardinality. Tento problém zachycuje obrázek 15.



Obrázek 15: Chyby grafického editoru v Oracle SQL Developer Data Modeler

3.2.1.3 Nevýhody

V průběhu používání jsem se setkal s několika problémy, které nejčastěji souvisely s grafickým editorem diagramů. Nepříjemná je drag and drop funkcionalita pro přidání existujících tabulek do diagramu. Pozice kde tabulku umístíme přetažením je ignorováno a Data Modeler ji umístí obvykle jinde. Po přetažení se navíc otevře detail struktury tabulky, což je nepříjemné, zejména pokud máme v úmyslu přidat více tabulek.

Nejzávažnější problém se vyskytl při převádění logického modelu na relační, kdy vztahy kardinality M:N byly převedeny chybně. Došlo sice k vytvoření asociační tabulky, ale bohužel přibyly atributy cizích klíčů také v obou tabulkách z logického modelu (viz obrázek 15). V logickém modelu je tedy nutné nejprve vytvořit ručně asociační entitu, byť neobsahuje žádné neklíčové atributy, a vztah převést na dvojici identifikujících vztahů 1:N. Po tomto opatření již proběhne převod správně.

Nepříjemná je také nutnost vytvoření nového relačního schématu ještě před jeho vygenerováním, protože jinak je vygenerováno relační schéma opakovaně do stejného diagramu a veškeré prvky se vyskytuji duplicitně. V této situaci nezbývá nic jiného než smazat veškerý obsah diagramu a vygenerovat jej znova, protože příkaz *zpět* tento problém neřeší. Dalším častým problémem je zobrazování dialogových oken mimo viditelnou plochu obrazovky po odpojení sekundárního monitoru. Problém je možné vyřešit smazáním konfiguračního souboru s lokálním uživatelským nastavením.

3.2.2 Toad Data Modeler

Další nástroj je vyvíjen od roku 2006 firmou Quest Software. Jedná se o komerční produkt určený pro operační systém Microsoft Windows. Toad podporuje více než 10 různých SŘBD v několika verzích, mezi nimi např. Oracle Database, Microsoft SQL Server nebo PostgreSQL (viz příloha C).

3.2.2.1 Popis funkcionality

Toad umožňuje vytvořit 3 typy modelů: fyzický, logický a univerzální. Nejpoužívanější je fyzický model, který je cílen na konkrétní SŘBD, další možností je logický model, ten se ovšem doporučuje jen v případě využití dědičnosti. Posledním typem je univerzální model, který umožňuje tvorbu diagramu bez návaznosti na konkrétní SŘBD, případně pokud určený SŘBD není mezi podporovanými. Notace jsou tentokrát v nabídce pouze dvě, implicitně nastavený Information Engineering (viz kapitola 2.7.2) a volitelná IDEF1X (viz kapitola 2.7.5).

Export je možné provést do CSV. Dle mého názoru je export řešen lépe než v nástroji Oracle SQL Developer Data Modeler, protože CSV soubor je exportován pouze jeden. Další možnosti je export do XLS. Poslední možností je export ve formě obrázku. Tato volba nabízí mnoho nastavení od velikosti plátna nebo rozdělení na stránky pro tisk po kompresi a barevnou hloubku.

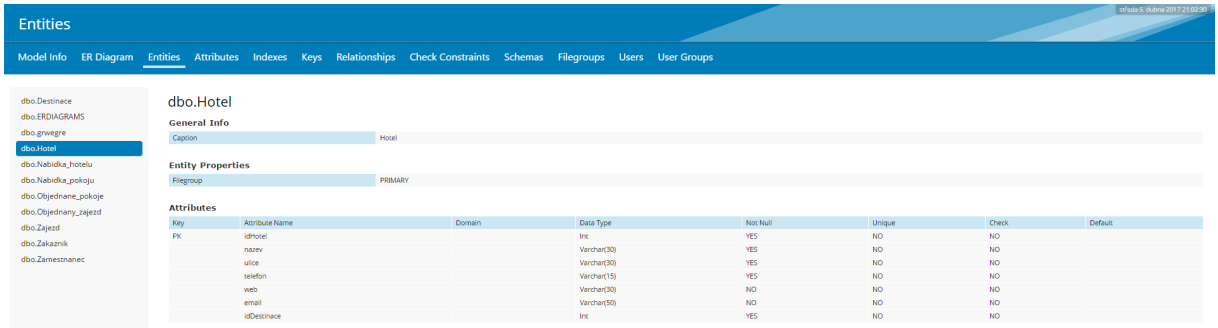
Z diagramu je možné vygenerovat DDL skript pro všechny podporované SŘBD bez ohledu na to, jaký byl zvolen při zakládaní projektu. DDL skripty je možné generovat buď úplné nebo obsahující jen změny v diagramu provedené v případě reverzního vygenerování. Takto vygenerovaný skript neobsahuje pouze tabulky a integritní omezení, ale může zahrnovat i uložené procedury, funkce, triggery nebo indexy.

Nástroj umožňuje rovněž kontrolu vytvořeného modelu než přejdeme ke generování skriptů. Fyzických modelů umožňuje vytvořit Toad Data Modeler několik v jednom projektu a následně je schopen provést jejich porovnání a přehledně vypsat v čem se liší. Modely je možné dokonce i sloučit. Další zajímavou volbou je možnost migrace na jiný SŘBD než z jakého byl model importován.

Import modelu je možný nejen z CSV, XLS a nativního formátu, ale také z jiných aplikací, konkrétně ER/Studio Data Architect a Toad for Oracle. Vytvoření modelu z DDL skriptu bohužel v případě tohoto nástroje chybí. Samozřejmostí je reverzní vygenerování modelu z připojené databáze. Do projektu zde lze zahrnout i indexy, procedury a další objekty z databáze, které poté mohou být rovněž součástí DDL skriptu.

Aktualizace databáze na základě změn v ER diagramu je možné provádět buď spuštěním vygenerovaného DDL skriptu obsahující změny samostatně, nebo lze využít definovaného připojení a změny provést přes GUI. Toad Data Modeler nabízí uložení modelů v různých fázích úprav, tento přístup umožňuje modely porovnat a případně vygenerovat skript na základě rozdílů mezi nimi.

Velmi užitečnou funkcí je možnost vytváření reportů. Reporty lze generovat do PDF nebo jako statické webové stránky, což je výhodné pro tvorbu dokumentace. Report obsahuje kompletní datový slovník, přehledně vypsané informace o integritních omezeních, zdrojové kódy uložených funkcí, procedur a mnoho dalšího. Před vygenerováním lze nastavit, co vše má být v reportu obsaženo. Příklad reportu vidíme na obrázku 16.



Obrázek 16: Report v podobě webové stránky z Toad Data Modeleru

3.2.2.2 Popis grafického editoru

Grafický editor pro tvorbu diagramů reaguje velmi rychle a za dobu testování jsem se nesetkal s žádnými výraznými problémy. Zobrazení entit lze měnit od pouhého názvu po detailní výpis atributů. Vztahům lze přidávat popisky, které lze v případě potřeby vypnout. Pracovní plocha je rozdělena na části velikosti A4, kvůli rozvržení pro tisk. Nevýhodou je absence mřížky, ke které by šly objekty přichytávat. Vztahy jsou reprezentovány lomenou čárou, úhly jsou vždy pravé, což považuji za velké plus z pohledu přehlednosti. Do pracovní plochy lze kromě entit a vztahů vkládat také text a základní geometrické tvary. Tato možnost je využitelná pro tvorbu popisků.

3.2.2.3 Nevýhody

Jeden z mála problémů, se kterým jsem se setkal, bylo hromadné přidání všech tabulek a vztahů z existující databáze do diagramu. Tabulky se všechny umístily do jednoho místa a bylo je nutné postupně rozprostřít po větší ploše. Naštěstí výrobce na tento problém nenechal bez povšimnutí a v nabídce je volba *autolayout*. Ta má za následek automatické uspořádání tabulek tak, aby se nepřekrývaly. Rozložení je možné shora dolů, zleva doprava nebo abecedně do obdélníku. Podobně lze znova vykreslit vztahy. Dokonce lze i zakázat křížení vizualizace vztahů, vyhledávání tras je velmi rychlé a dokonce ani po zapnutí volby pro co nejpřesnější trasy nepřesáhl čas řádově jednotky sekund.

Jedinou výtkou k nástroji Toad Data Modeler je nepřehlednost uživatelského rozhraní. Jedná se o opravdu velmi komplexní nástroj a to mělo na GUI pravděpodobně největší dopad. Hlavní menu obsahuje sice optimální počet položek a není překombinované, nicméně o nástrojové liště to říct nelze. Ukázku nástrojové lišty vidíme na obrázku 17. Většina funkcionality je směřovaná zde a novému uživateli zabere nějaký čas než prozkoumá veškeré možnosti, výhodou je velké množství klávesových zkratek, které velmi usnadňují práci. Problém způsobovalo také rozvržení spodních panelů na obrazovce s nižším rozlišením, nicméně přesunutí panelů fungovalo bez problémů a na obrazovce s FullHD rozlišením se problém neprojevil.

Obrázek 17: Nástrojová lišta nástroje Toad Data Modeler

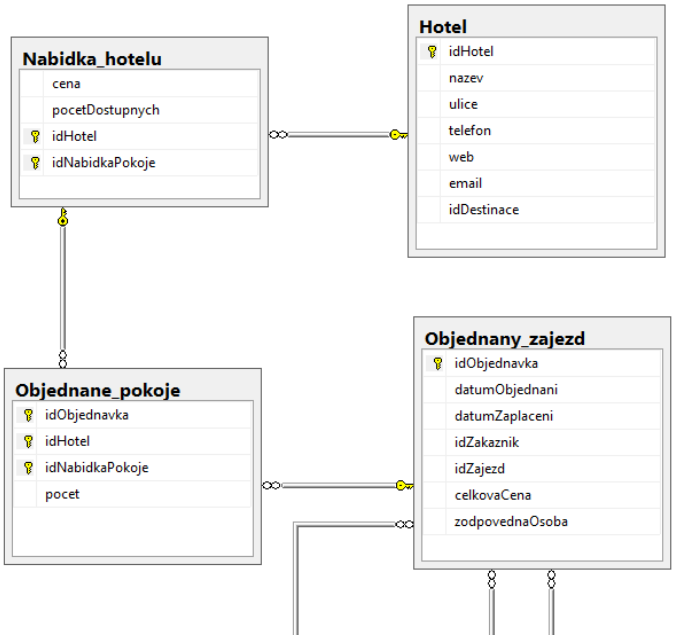
3.2.3 SQL Server Management Studio

Tento nástroj, často označován jen jako SSMS, je vyvíjen společnosti Microsoft. První verze byla vydaná společně s Microsoft SQL Server 2005. Předchůdcem toho nástroje je Enterprise Manager, který se používal ke správě starších verzí SŘBD SQL Server. Jedná se o plnohodnotný nástroj pro administraci SŘBD SQL Server. Většinu potřebných úkonů je možné provést s využitím GUI. Součástí tohoto nástroje je také možnost tvorby ER diagramů a právě na ní se zaměříme. Nástroj je určen pro platformu Microsoft Windows a kromě SQL Serveru nepodporuje žádné další SŘBD.

3.2.3.1 Popis funkcionality

Management studio umožňuje pouze tvorbu fyzického modelu databáze. Nemožnost tvorby modelů na jiných úrovní abstrakce je způsobena primárním zaměřením celého nástroje na správu již existujících databází. Konceptuální model tedy v tomto případě postrádá smysl. Použitá notace je specifická, tento nástroj nepoužívá žádnou z dříve zmíněných notací. Ukázku notace zachycuje

obrázek 18 Povinnost vztahu je znázorněna graficky v look here stylu, kardinalita znázorněna není, předpokládá se 1:N.



Obrázek 18: Notace SSMS

Import diagramů z jiných nástrojů nebo souboru není k dispozici v žádné formě, jedinou možností je vytvoření databáze na serveru a vygenerování diagramu na základě jejího schématu. Stejně tak není podporován ani export do CSV, XLS nebo XML, jako je tomu u konkurenčních nástrojů.

Diagramy jsou uloženy ve speciální tabulce (`sysdiagrams`) přímo v databázi. V práci je tedy možné bez problémů pokračovat z jiného počítače aniž bychom museli kopírovat soubory projektu. Nástroj podporuje tisk diagramů, takto můžeme vytvořit i PDF nebo XPS.

Reverzní generování diagramu umožňuje vizualizovat určitou část schématu, tato možnost je praktická zejména při práci s rozsáhlými systémy, kde lze schéma rozdělit na logické celky. Aktualizace schématu databáze prostřednictvím ER diagramu jsou velmi rychlé, stačí potvrzení změn uložením diagramu a patřičné změny se ihned promítnou v aktuálním schématu. Před potvrzením je možné zobrazit DDL skript obsahující SQL příkazy k modifikaci schématu.

Přepínání mezi diagramy funguje prakticky okamžitě, změny provedené v jednom jsou ihned zobrazeny v druhém, z tohoto pohledu je synchronizace na skvělé úrovni. Problém způsobilo jen přidání atributu do tabulky, ze které byl atribut se stejným názvem v dalším diagramu smazán, změny byly potvrzeny opožděně, nicméně tato situace skončila chybou a pomohlo jen smazání diagramů z databáze a jejich opětovné vygenerování. Tento fakt nepovažuji za zásadní selhání, protože se jedná o zcela neobvyklou operaci.

3.2.3.2 Popis grafického editoru

Grafický editor reaguje ve srovnání s ostatními nástroji nejrychleji z testovaných. Umožňuje přidat kromě objektů z databáze také popisky, stejně tak je možné přidávat popisky vztahům. Režimů zobrazení tabulek editor nabízí také několik, od pouhého názvu tabulky po zobrazení pouze klíčových atributů. V případě potřeby je možné využít automatické rozvržení, podobně jako u nástroje Toad Data Modeler, a celé schéma nechat přehledně uspořádat.

3.2.3.3 Nevýhody

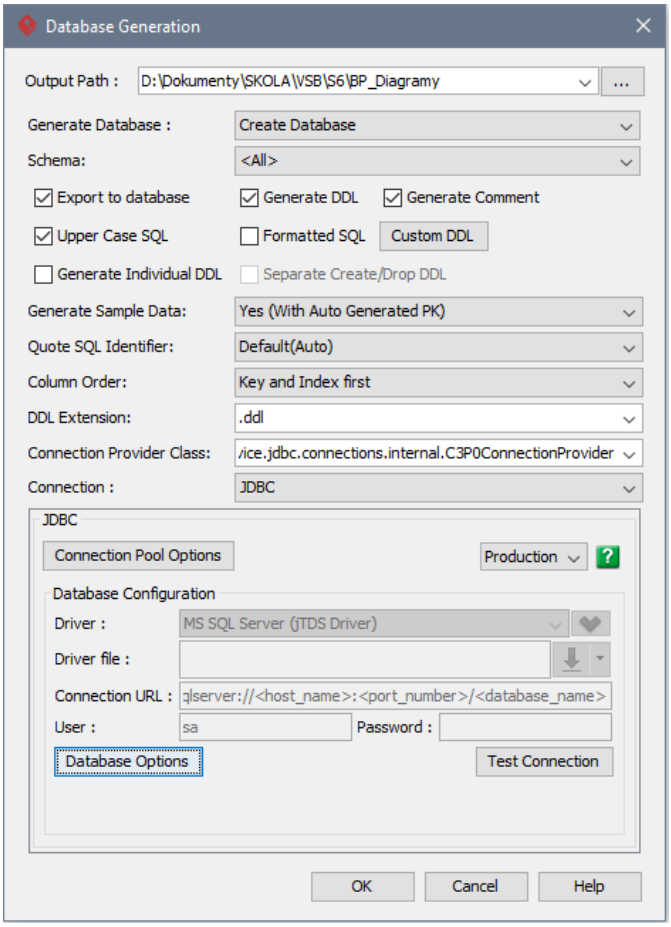
Žádné problémy s grafickým editorem jako je špatné uspořádání čar, problikávání objektů a podobně se nevyskytly. Práce je velmi intuitivní a editor neobsahuje zbytečné funkce. Právě tato jednoduchost má největší vliv na uživatelský komfort, který považuji za nejvyšší z testovaných nástrojů.

3.2.4 Visual Paradigm

Dalším testovaným nástrojem je komerční produkt od firmy Visual Paradigm International, zastřešuje ji Hong Kong Institute of Vocational Education, který je určen primárně pro tvorbu UML diagramů. Nástroj podporuje i modelování relačních databází formou ER diagramu. K dispozici je několik typů placených licencí rozdělených podle nabízených možností a také community edice, která je určena pro nekomerční projekty. Nevýhodou community edice je absence veškerých pokročilých funkcí pro práci s relační databází, jako je např. reverzní generování diagramů.

3.2.4.1 Popis funkcionality

Visual Paradigm podporuje tvorbu logických modelů bez nutnosti specifikovat SŘBD, ten je nutné uvést až při generování skriptu a nástroj podporuje více než deset různých systémů. Diagram používá Crow's foot notaci (viz kapitola 2.7.2), bohužel je jediná podporovaná a nelze ji přepnout jako v případě jiných nástrojů. Možná nastavení pro vygenerovaní relačního schématu zachycuje obrázek 19.



Obrázek 19: Generování schématu databáze z ER diagramu ve Visual Paradigm

Diagramy je možné importovat v rámci projektu, samostatný import není možný. Diagramy můžeme importovat z nezávislých formátů jako je XML nebo i z jiných nástrojů, příkladem je konkurenční Enterprise Architect, Visio a několik dalších.

Nástroj nabízí tisk diagramů, pomocí této volby jsme schopni vytvořit i PDF nebo XPS soubor. Další možností exportu, kromě nativního formátu, je XML soubor a obrázek. Nastavení je poměrně detailní, lze nastavit formát, kompresi a dokonce i rozdělení diagramu na části specifikované svou velikostí. V případě XML můžeme ovlivnit jeho podobu volbou struktury.

Nástroj podporuje generování DDL skriptu vytvořeného diagramu. Diagramy mohou obsahovat i uložené procedury, funkce a triggery, které mohou být také součásti tohoto skriptu. Skript můžeme uložit do souboru nebo jej rovnou spustit nad definovaným připojením a modelované schéma tak fyzicky vytvořit v databázi. Oproti dalším nástrojům umí Visual Paradigm vygenerovat také kostru příkazů pro vložení a aktualizaci záznamů pro jednotlivé tabulky.

Na rozdíl od jiných nástrojů je možné vytvořit z logického modelu také kostru pro objektově-relační mapování. Tuto volbu považuji za jeden z největších přínosů, protože pro rozsáhlé databáze ušetří obrovské množství času. Další užitečnou funkcí je i možnost synchronizace s třídním diagramem. Diagramy můžeme reverzně generovat z existující databáze nebo DDL skriptu.

Synchronizace se schématem databáze není provedena okamžitě, ale je nutné ji spustit prostřednictvím GUI.

3.2.4.2 Popis grafického editoru

Práce s grafickým editorem je intuitivní, reakce působí svižně a nedochází k žádným problémům s výskytem grafických artefaktů. Zobrazení tabulek je možné všemožně upravovat, k dispozici je i mřížka, ke které lze objekty zachytávat. Samozřejmostí je i zobrazení popisků vztahů, případně dalších objektů. Uživatelské rozhání je, vzhledem ke komplexnosti nástroje, přehledně uspořádáno. Nástroj považuj za vhodnou volbu zejména pro rozsáhlejší projekty, kde je výhodné diagram datového modelu využít i v dalších diagramech.

3.2.4.3 Nevýhody

Definování nového připojení k databázi je jedna ze slabých stránek nástroje, pro SQL Server jsou možnosti velmi omezené a chybí např. možnost autentikace pomocí Windows účtu. Pro testování jsem tedy musel vytvořit nový login a databázového uživatele. Samotné připojení k databázi se pravděpodobně provádí na hlavním vlákně, což často způsobuje zamrznutí celého GUI.

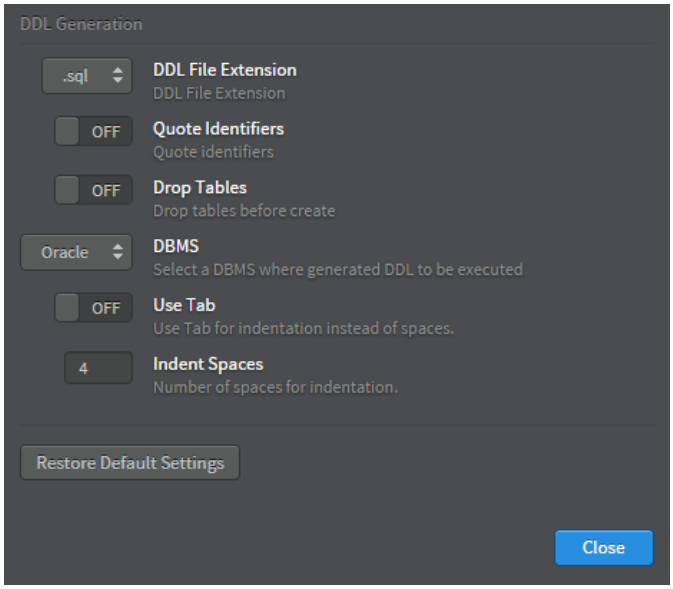
3.2.5 StarUML 2

StarUML je poměrně nový nástroj, první verze byla vydaná roku 2014. Nástroj následuje trend desktopových aplikací vytvořených pomocí webových technologií. Tyto aplikace využívají pro svůj běh NodeJS, což umožňuje nasazení bez ohledu na platformu. Jedná se o komerční nástroj, pro testování jsem použil zkušební verzi. Licence pro studenty není zdarma, lze ji ale pořídit se slevou.

3.2.5.1 Popis funkcionality

Nástroj podporuje pouze logický model, který používá Crow's foot notaci (viz kapitola 2.7.2), jiné typy notací bohužel v nabídce nejsou. Model je vytvářen univerzálně, je možné použití pro různé SŘBD.

Zde se dostáváme k největší výhodě nástroje, kterou je modularita. Použití webových technologií umožňuje velmi snadnou tvorbu doplňků, které lze instalovat z oficiálního repozitáře. Po instalaci dokonce nebyl nutný ani restart aplikace a nové možnosti byly dostupné ihned. Právě doplněk je nutný i pro generování DDL skriptu, protože samotná aplikace tuto možnost neobsahuje. Zmíněný doplněk v současné době podporuje MySQL a Oracle Database, zdrojové kódy jsou dostupné na serveru GitHub, což přináší možnost rozšířit funkcionalitu o další SŘBD. Konfiguraci doplňku pro generování DDL skriptu zachycuje obrázek 20.



Obrázek 20: Konfigurace doplňku pro generování DDL skriptu ve StarUML 2

Importovat diagramy lze pouze z nativního formátu souboru a StarUML 1. Výhodou je, že nástroj ukládá diagramy ve formátu JSON, tento formát je možné využít pro případnou spolupráci s jinými nástroji. Export diagramu je možný ve formátu PNG, JPEG a dokonce SVG, což je vhodné pokud potřebujeme diagram využít i v jiném nástroji, který podporuje vektorové formáty, poslední možností je export do PDF. Nástroj umí vytvořit report ve formě webových stránek, zde nalezneme jak diagramy projektu, tak detailní popis tabulek, atributů a vztahů.

Nástroj bohužel nenabízí možnost reverzního generování diagramů z databáze, ani DDL skriptu. Stejně tak nástroj postrádá synchronizaci s již existujícím schématem. Možným řešením je modul, který by rozšířil nástroj o tuto funkcionalitu, v současné době se mi bohužel nepodařilo takový najít.

3.2.5.2 Popis grafického editoru

Grafický editor nabízí všechny základní funkce pro vytváření ER diagramu a celkově působí uživatelské rozhraní přehledně.

3.2.5.3 Nevýhody

Za největší nedostatek považuji rychlost reakcí editoru. Zde se bohužel projevil nižší výkon použitých technologií oproti klasickému přístupu k vývoji desktopových aplikací. Při přesouvání objektů po plátně se editor velmi znatelně zadrhává. Vzhledem ke zbytku srovnávaných nástrojů je v tomto ohledu StarUML rozhodně nejhorší.

3.3 Shrnutí

Všechny testované nástroje nabízí alespoň základní funkcionalitu nutnou pro úspěšné vytvoření ER diagramu. Většina z nich se nezastavuje u pouhého grafického editoru s možností exportu DDL skriptu, ale nabízí mnoho komplexnějších funkcí. Hodnocen nebyl pouze grafický editor pro tvorbu diagramů, ale také možnosti reverzního generování diagramů z databáze a s tím spojená synchronizace s jejím schématem. Dalšími aspekty v hodnocení byly také možnosti importu a exportu schémat, kdy XML nebo CSV mohou být výhodné z hlediska strojového zpracování nástroji pro formální analýzu, zatímco export do Excelu nebo PDF je využitelný pro tvorbu dokumentace projektu.

Z hlediska grafického editoru považuji za nejlepší SQL Server Management Studio pro jeho rychlost a snadnou aplikaci změn na existující schéma. Bohužel jej nelze použít i pro jiná SŘBD. Stejně kvalitním editorem disponuje např. Visual Paradigm, který je vhodný pro jiné SŘBD, ale aplikace změn není tak přímočará jako u nástroje SQL Server Management Studio. Přehledné srovnání možností z pohledu grafických editorů nabízí tabulka 3. Rychlost odezvy a přehlednost GUI je hodnocena známkou z intervalu 1 - 5 (1 = nejlepší, 5 = nejhorší).

Tabulka 3: Srovnání grafických editorů pro vybrané CASE nástroje

Vlastnost	SQL Dev.	Toad	SSMS	Visual Paradigm	StarUML
Výchozí notace	Barker	IE	Vlastní	Crow's foot	Crow's foot
Možnost volby notace	Ano	Ano	Ne	Ne	Ne
Strojově čitelný export	Ano	Ano	Ne	Ano	Ano
Export do PDS/XPS	Ano	Ano	Ano	Ano	Ano
Export formou obrázku	Ano	Ano	Ne	Ano	Ano
Rychlost odezvy	3	2	1	1	4
Přehlednost GUI	2	3	1	2	1

V oblasti synchronizace diagramu s existující databází nabízí nejširší škálu možností Toad Data Modeler. Ostatní nástroje předčil podporou mnoha SŘBD, porovnáním modelů a detailními možnostmi nastavení generování DDL skriptu. Srovnání všech nástrojů v této oblasti nalezneme v tabulce 4, komfort aktualizací schématu je opět hodnocen známkami 1 - 5, StarUML je z tohoto hodnocení vynechán, protože práci s databází nepodporuje.

Tabulka 4: Srovnání možností synchronizace pro vybrané CASE nástroje

Vlastnost	SQL Dev.	Toad	SSMS	Visual Paradigm	StarUML
Synchronizace s exist. schématem	Ano	Ano	Ano	Ano	Ne
Podpora více SŘBD	Ano	Ano	Ne	Ano	Ano
Reverzní generování - databáze	Ano	Ano	Ano	Ano	Ne
Reverzní generování - DDL	Ano	Ano	Ano	Ano	Ne
Export kompletního DDL	Ano	Ano	Ne	Ano	Ano
Export DDL se změnami	Ano	Ano	Ano	Ano	Ne
Migrace na jiný SŘBD	Ano	Ano	Ne	Ano	Ne
Komparace modelů	Ne	Ano	Ne	Ne	Ne
Procedury v modelu	Ne	Ano	Ne	Ano	Ne
Komfort aktualizace schématu	1	2	1	3	-

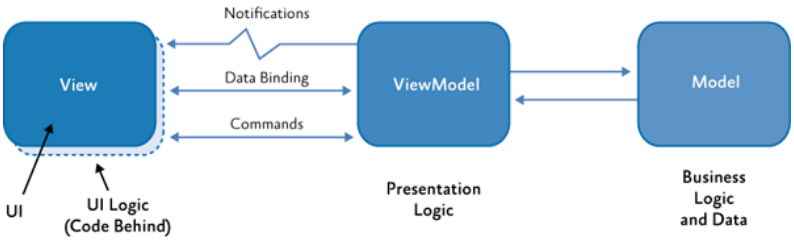
4 Návrh a implementace grafického editoru ER diagramů

Grafický editor představuje jádro celého CASE nástroje, protože většina operací, které nástroj umožňuje, je prováděna prostřednictvím jeho rozhraní. Nástroj je implementován jako desktopová WPF aplikace [10] určena pro operační systém Microsoft Windows.

4.1 Architektura aplikace

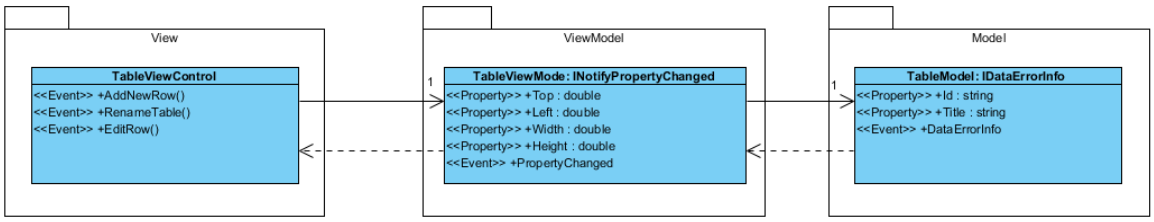
S WPF aplikacemi se váže návrhový vzor MVVM [11], který slouží k oddělení prezentační vrstvy od doménové logiky, včetně přístupu ke zdroji dat. Tento návrhový vzor je výhodný zejména pro vývoj rozsáhlejších aplikací, kdy čistě událostmi řízené programování může vést k obtížím s laděním aplikace, případně pozdějším úpravám v oblasti grafického rozhraní. Jednotlivé části vzoru MVVM jsou znázorněny na obrázku 21. Vzor rozděluje aplikaci na tyto části:

- **Model** slouží pro uchování dat aplikace. Modely obvykle obsahují také doménovou logiku jako na např. validace vstupních dat.
- **View** slouží k prezentaci dat a komunikaci s uživatelem. Veškerá interakce s konkrétními ovládacími prvky uživatelského rozhraní se provádějí v této vrstvě.
- **ViewModel** funguje jako mezivrstva mezi view a modelem. Pokud dojde ke změně vlastností viewmodelu, view je o těchto změnách informováno a změnám se přizpůsobí. Viewmodel je nezávislý na prvcích uživatelského rozhraní. Z tohoto důvodu se zde využívá princip zvaný data-binding, který je popsán v kapitole 4.2.



Obrázek 21: Blokové schéma komponent MVVM (Zdroj: [11])

Aplikaci vzoru MVVM v praxi vidíme na obrázku 22. Příklad zobrazuje strukturu komponenty pro vizualizaci tabulek v grafickém editoru. Plná čára zobrazuje asociaci mezi třídami. Vazba je jednosměrná, přímá komunikace mezi view a modelem tedy není možná, vše se odehrává prostřednictvím viewmodelu. Změny dat jsou přenášeny pomocí událostí, směr jejich propagace je značena čárkovanou čárou. Viemodel informuje view o změně dat pomocí události `PropertyChanged`. Model používá stejný princip při validaci dat, tedy událost `DataErrorInfo`.



Obrázek 22: Blokové schéma komponenty pro tabulky v grafickém editoru

4.2 Klíčové části grafického editoru

Grafický editor je stejně jako ostatní části aplikace navrhnut s ohledem na výše zmíněný návrhový vzor MVVM. Viewmodely ve WPF aplikacích slouží pro uchovávání dat pro jednotlivé komponenty GUI v podobě jaká vyhovuje prezentační vrstvě. Provázaní viewmodelů s prezentační vrstvou je řešeno principem, který se obecně označuje jako data-binding. Standardně se realizuje toto provázání v XAML kódu, který určuje strukturu uživatelského rozhraní aplikace. Ukázku data-bindingu pro textové pole určující přesnost datového typu vidíme na výpisu kódu 2. Grafický editor je z pohledu data-bindingu specifickou komponentou, zejména co se týká povahy prováděných operací, provázaní dat s prezentační vrstvou bylo tedy nutné implementovat alternativním způsobem.

```
<TextBox x:Name="PrecisonTextBox"
   Grid.Column="1"
   IsReadOnly="True"
   controls:TextBoxHelper.UseFloatingWatermark="True"
   controls:TextBoxHelper.Watermark="Precision"
   Text="{Binding SelectedDatatype.Precision, Mode=OneWay,
      UpdateSourceTrigger=PropertyChanged}">
</TextBox>
```

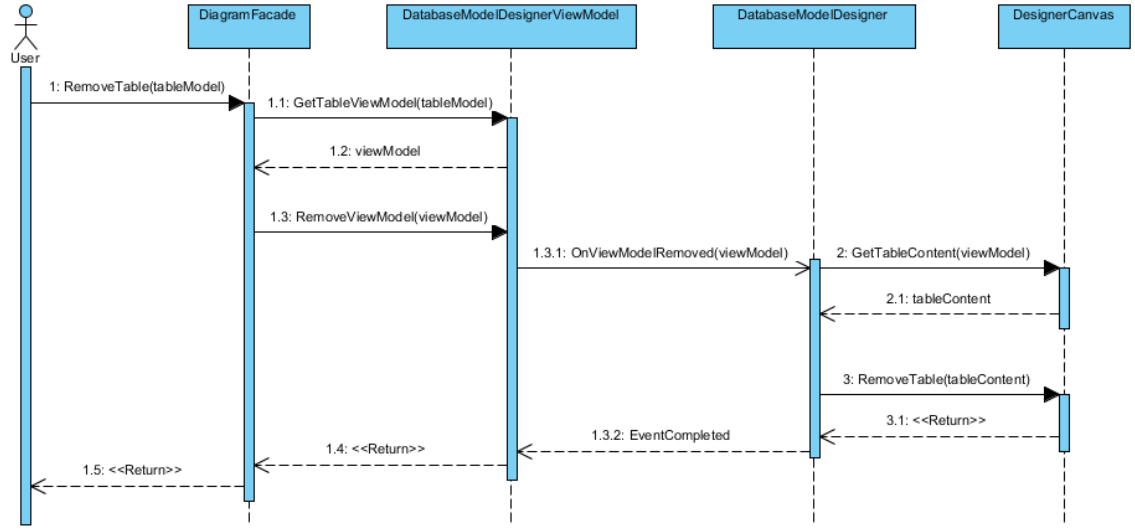
Výpis 2: Příklad data-bindingu v XAML pro textové pole

4.2.1 Provázání editoru s daty

Dle MVVM obsahuje viewmodel grafického editoru trojici kolekcí pro uložení viewmodelů tabulek, vztahů a popisků, tedy objektů, které se vyskytují na plátně. Kolekce po přidání nebo odebraní objektu vyvolají událost, na kterou patřičným způsobem reaguje komponenta editoru. Viewmodel tedy vystavuje rozhraní grafického editoru ostatním částem aplikace. Pro přidání nové tabulky stačí jen vložit její viewmodel, s nastavenými hodnotami, do kolekce tabulek a editor se sám postará o přidání objektu na plátno. Není tedy nutná přímá interakce jiné komponenty s prezentační vrstvou grafického editoru. Tato implementace umožňuje pracovat s grafickým edi-

torem na úrovni viewmodelu, stejným způsobem jako s komponentami využívajícími standardní způsob data-bindingu ve WPF aplikacích.

Pro ilustraci je proces odstranění tabulky z plátna zachycen sekvenčním diagramem na obrázku 23. Třída `DiagramFacade` slouží k zapouzdření operací nad viewmodely, její metody jsou volány z po úspěšném provedení akce uživatelem. Odstranění viewmodelu tabulky z kolekce ve třídě `DatabaseModelDesignerViewModel` vyvolá událost, na tu reaguje prezentační vrstva odstraněním příslušných objektů z plátna (třída `DesignerCanvas`).



Obrázek 23: Sekvenční diagram - odstranění tabulky z plátna

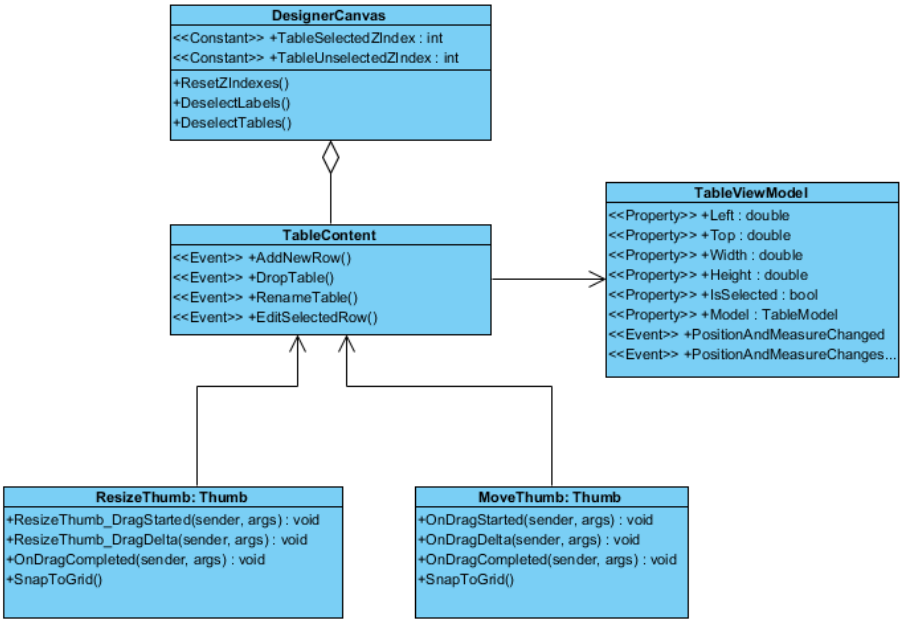
4.2.2 Vizualizace tabulek

Implementace změny rozměrů a pozice tabulky na plátně je úzce spjatá s použitím WPF komponenty `Thumb`, která je nejčastěji využívaná v aplikacích určených pro dotykové displaye. Na první pohled se jeví jako nejsnazší řešení explicitní zachytávaní událostí vyvolaných po stisku tlačítka a tahu myši s následným výpočtem rozdílu mezi aktuálním a počátečním bodem. Problémem této implementace je obtížná generalizace tohoto řešení, protože zde vzniká závislost na implementaci komponenty pro grafické znázornění tabulky. Tuto závislost lze řešit právě použitím již zmíněné komponenty `Thumb` v kombinaci s XAML šablonou.

4.2.2.1 Komunikace komponent grafického editoru s viewmodely

Pro zprostředkování komunikace mezi šablonou a viewmodelem tabulky slouží třída `TableContent`. Strukturu této části aplikace zachycuje třídní diagram na obrázku 24, ten je značně zjednodušený oproti implementaci, obsahuje jen položky důležité pro tuto kapitolu. Třída `TableContent` obsahuje referenci na viewmodel tabulky. Po interakci s `Thumb` objekty tedy zajistí aktualizaci patřičných hodnot v jeho vlastnostech. Třída `TableContent` slouží také k propagaci události pro

přidání atributu, odstranění tabulky z databáze a mnohé další, tak aby komponenta pro grafické znázornění tabulky obsahovala co nejméně doménové logiky.



Obrázek 24: Část třídního diagramu - přesunutí a změna rozměrů tabulky

`Thumb` vyvolává událost na počátku tahu, v jeho průběhu a po ukončení tahu. Explicitní zachytávání událostí myší tedy není nutné, stejně tak rozdíl souřadnic počátečního a aktuálního bodu je součásti argumentu událostí. Šablona umožňuje nastavení společných základních vlastností pro jakoukoli implementaci komponenty pro tabulku, společně s definicí triggeru pro zviditelnění vrstvy umožňující změnu rozměrů tabulky po jejím výběru. Triggery reagují na změny hodnot ve viewmodelech, ukázka triggeru pro ovládání vrstvy umožňující změnu rozměrů je k vidění na výpisu kódu 3. Tímto způsobem je možné provádět úpravy komponenty pro zobrazení tabulky, aniž by došlo k narušení funkcionality přesunu a změny rozměrů.

```
<ControlTemplate.Triggers>
   <Trigger Property="IsSelected" Value="True">
      <Setter TargetName="ItemDecorator" Property="ShowDecorator" Value="
         True"/>
   </Trigger>
</ControlTemplate.Triggers>
```

Výpis 3: Příklad triggeru v XAML pro vrstvu umožňující změnu rozměrů

Změna rozměrů využívá opět komponenty `Thumb`, která je nyní součástí tzv. adorner vrstvy, tato vrstva se zobrazí až po vybrání tabulky. Vybraním dojde k úpravě hodnoty jejího z-indexu

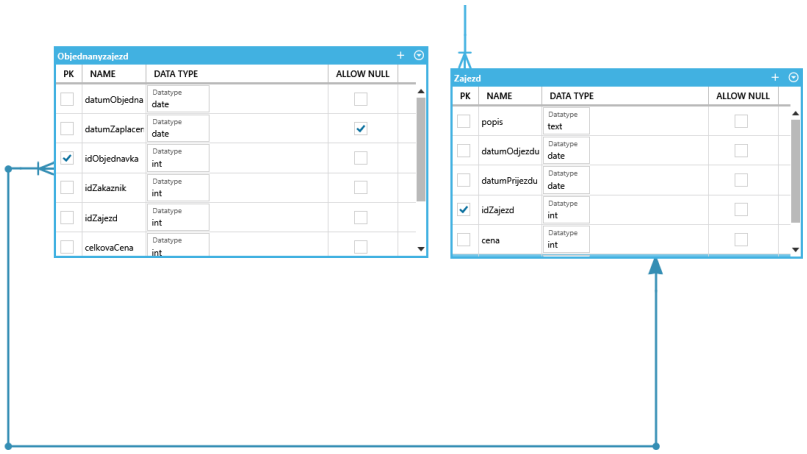
tak, aby překryla veškeré ostatní objekty na plátně a bylo možné s adorner vrstvou komfortně pracovat.

4.2.3 Vizualizace vztahů

Práce se samotnými vztahy probíhá opět na úrovni viewmodelu. Podobně jako je tomu u tabulek, nicméně samotný vztah nyní není reprezentován jediným objektem na plátně, ale objektů se zde vyskytuje celá řada.

4.2.3.1 Reprezentace vztahů pomocí lomených čar

Vztah je reprezentován lomenou čárou. Na obou koncích této čáry se nachází symboly vyjadřující kardinalitu a povinnost. Na obrázku 25 vidíme ukázku lomené čáry spojující dvě tabulky. Tato lomená čára je reprezentována seznamem segmentů, kde sousední segmenty svírají vždy pravý úhel. V grafickém editoru je použita vlastní implementace lomené čáry, protože třída `Polyline`, která je pro práci s lomenými čárami určená, nabízí velmi omezené možnosti interakce ze strany uživatele a hodí se tedy spíše pro pouhou vizualizaci dat např. formou grafu.

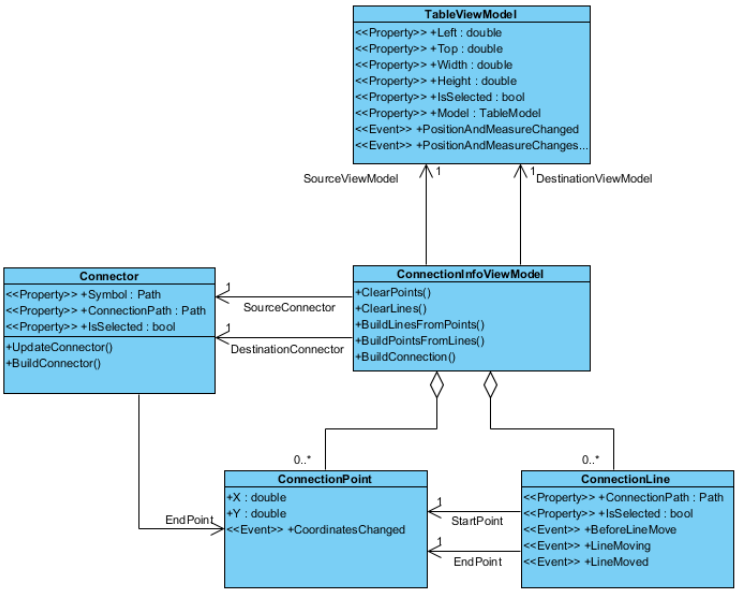


Obrázek 25: Vizualizace vztahu mezi dvěma tabulkami

4.2.3.2 Interakce s lomenými čárami

Závislosti mezi třídami pro vizualizaci vztahů popisuje třídní diagram 26. Viewmodel pro vztah obsahuje kolekci lomových bodů čáry, společně s kolekcí čar, které tvoří jednotlivé segmenty lomené čáry. V první fázi konstrukce lomené čáry jsou segmenty vytvořeny na základě lomových bodů, které jsou po přidání nového vztahu určeny algoritmem pro vyhledání trasy. Detailní popis algoritmu nalezneme v kapitole 4.3. Editor reaguje na události vyvolané při práci s kolekcí čar tak, aby byla zajištěna integrita dat viewmodelu s objekty na plátně.

Všechny segmenty lomené čáry reagují na interakci pomocí myši. Pro přenos informací o provedených změnách do viewmodelu je využit návrhový vzor Pozorovatel, který využívá pro tuto



Obrázek 26: Část třídního diagramu - vizualizace vztahu mezi tabulkami

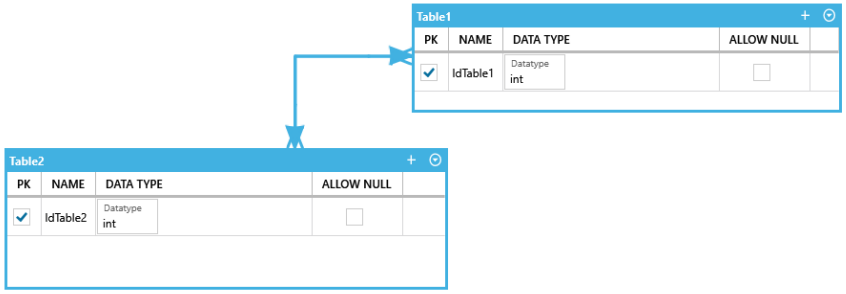
činnost události. Viewmodel vztahu je informován o průběhu interakce, tím je tedy možné provádět aktualizaci pozice přilehlých segmentů lomené čáry včetně ukončujících symbolů v závislosti na pohybu určitého segmentu. Po ukončení interakce je opět vyvolána událost, která spustí post-processing, jehož součástí je např. spojení stejně orientovaných segmentů čáry lišících se pozicí v rámci tolerance nebo změna orientace ukončujících symbolů na základě orientace krajních segmentů čáry.

4.3 Vyhledání trasy po přidání vztahu

Po přidání nového vztahu mezi tabulkami je nutné tento vztah vykreslit na plátno. Tento krok byl v první fázi vývoje nástroje implementován metodou, která nejdříve zjistila vzájemnou polohu obou tabulek a následně vztah vykreslila jako spojnici předefinovaných lomových bodů. Ukázku této situace vidíme na obrázku 27. Tento způsob se neosvědčil když tabulka figurovala v několika vztazích, protože často docházelo k překrývaní čar. Největším problémem tohoto řešení byl fakt, že nejsou zohledněny další tabulky na plátně a dochází ke křížení vztahů s tabulkami, které se vyskytují v předdefinované trase. Z toho důvodu je diagram transformován na graf a problém je řešen jako vyhledávání cesty v grafu.

4.3.1 Sestavení grafu

Prvním krokem při hledání vhodné trasy pro vztah je převod plátna na graf. Vrcholy grafu jsou tvořeny obrazovými body plátna. Mezi vrcholy existuje hrana pokud se jedná o sousední



Obrázek 27: Ukázka překrytí lomených čar

obrazové body. Nejprve bylo nutné převést každý obrazový bod na samostatný uzel grafu. Tento uzel obsahuje souřadnice bodu na plátně, společně s informací zda je bod volný nebo jej překrývá tabulka. Výhodou této reprezentace je její přesnost, kdy nedochází k zanedbání informací z plátna. Problémem je ovšem negativní dopad na výkon nástroje, protože pro plátno o šířce a výšce 4000 obrazových bodů je nutné alokovat 16 milionů uzlů, které v průměru zabírají více než 1GB operační paměti. Počet uzlů grafu je nutné zredukovat tak, aby zabíraly řádově méně prostoru operační paměti. Z tohoto důvodu je plátno nejprve rozděleno na čtvercové úseky o konstantních rozměrech. Pokud tabulka alespoň částečně zasahuje do některého z úseku, je úsek označen jako obsazený. Z této mřížky je následně vytvořen graf, který místo 16 milionů uzlů jich pro dříve zmíněné rozměry obsahuje pouze 1600. Alokace operační paměti v tomto případě trvá jen několik milisekund a zabraná paměť se pohybuje řádově v jednotkách MB.

4.3.2 Vyhledávání cesty v grafu

Tento problém je v práci řešen dvěma algoritmy: prohledávání do šířky a A*. Vstupem obou algoritmů je prohledávány graf, počáteční a cílový vrchol. Výstupem je kolekce uzlů reprezentující nalezenou cestu. Dle této kolekce je možné sestavit vztah (viz obrázek 26).

4.3.2.1 Prohledávání grafu průchodem do šířky

Princip prohledávání popisuje algoritmus 1. Prohledávání grafu do šířky využívá pro svou činnost frontu. V každém kroku jsou do fronty přidány sousední volné uzly vzhledem k aktuálně zpracovávanému uzlu. Pokud je tento uzel cílovým, prohledávání končí a cesta je nalezena. Pokud dojde k vyprázdnění fronty aniž by byl nalezen cíl, tak cesta v grafu neexistuje. Funkce `BacktrackNodes` slouží pro následování ukazatelů na rodičovské uzly pro vytvoření kolekce uzlů v nalezené cestě. Tento způsob hledání trasy je velmi snadno implementovatelný, jeho výkon bohužel není ideální, protože vyhledávání trvá řádově desítky až stovky milisekund pro jedinou

trasu.

Vstup : Graf G, Počáteční vrchol S, Cílový vrchol E
Výstup: Cesta ve formě kolekce uzlů

```
1  uzavřené_uzly ← prázdná množina;
2  otevřené_uzly ← prázdná fronta;
3  otevřené_uzly.ZařaditNaKonecFronty(S);
4  while otevřené_uzly is not empty do
5  |  aktuální ← otevřené_uzly.VybratPrvníVeFrontě();
6  |  uzavřené_uzly.Přidat(aktuální);
7  |  if aktuální == E then
8  |  |  výsledek ← BacktrackNodes(aktuální);
9  |  end
10 |  foreach node N that is adjacent to aktuální do
11 |  |  if N is not in uzavřené_uzly then
12 |  |  |  N.Parent ← aktuální;
13 |  |  |  otevřené_uzly.ZařaditNaKonecFronty(N);
14 |  |  end
15 |  end
16 end
```

Algoritmus 1: Prohledávání grafu průchodem do šířky

4.3.2.2 Prohledávání grafu algoritmem A*

Druhým implementovaným algoritmem je A* [8], používaný k nalezení nejkratší cesty v grafu. Tento algoritmus kombinuje výhody hladového BFS a Djikstrova algoritmu [8]. Priorita prohledávání uzlů je určena na základě heuristiky, která slouží k odhadu vzdálenosti z aktuálního uzlu k cíli. Metod jak tuto hodnotu určit je několik, její volba závisí na reprezentaci prohledávaného prostoru a možných směrech. Pro mřížku se 4 směry pohybu je nejvhodnější metoda Manhattan 1. Metoda Manhattan odhaduje vzdálenost z aktuálního uzlu k cíli jako součet absolutních hodnot vertikálního a horizontálního rozdílu bodů.

$$H = \mid aktualni.x - cilovy.x \mid + \mid aktualni.y - cilovy.y \mid \tag{1}$$

Uzly jsou uchovány v prioritní frontě [9], priorita je určena hodnotou vypočtenou pomocí metody Manhattan 1. Možností implementace prioritní fronty je celá řada, pro tento účel bylo nejvhodnější použít binární haldu, pro co nejlepší výkon algoritmu. Jedná se o stromovou strukturu, kde prvek s nejvyšší prioritou nalezneme v kořeni stromu. Open-source knihovny bohužel obvykle neimplementují operaci pro aktualizaci priority, která vyvolá nutnost úpravy stromové

struktury. V nástroji bylo tedy nutné tuto strukturu pro správnou funkci A* algoritmu implementovat také.

```
   Input  : Graph G, Start vertex S, End vertex E
   Output: Path as a vertex collection
 1 closed ← empty set;
 2 open ← empty priority queue;
 3 open.Enqueue(S);
 4 while open is not empty do
 5 |   current ← open.Dequeue();
 6 |   closed.Add(current);
 7 |   if current == E then
 8 |   |   Result ← BacktrackNodes(current);
 9 |   end
10 |   foreach node N that is adjacent to current do
11 |   |   if N is in closed or N is obstacle then
12 |   |   |   continue;
13 |   |   end
14 |   |   N.Parent ← current;
15 |   |   N.Manhattan ← Manhattan(N, E);
16 |   |   if N is not in open then
17 |   |   |   open.Enqueue(N);
18 |   |   |   continue;
19 |   |   end
20 |   |   open.UpdatePriority(N);
21 |   end
22 end
```

Algoritmus 2: Prohledávání grafu algoritmem A*

Algoritmus 2 demonstruje princip A*. Ten spočívá v použití prioritní fronty pro ještě nezpracované uzly a množiny pro uzly zpracované. Algoritmus v každém kroku vezme z fronty nezpracovaných uzlů uzel s nejvyšší prioritou. Pokud je zpracovávaný uzel cílový, prohledávání končí. V opačném případě je uzel vložen do množiny zpracovaných uzlů, pro sousední uzly je určena priorita, ukazatel na rodiče je nastaven na aktuální uzel a tyto uzly jsou vloženy do množiny nezpracovaných uzlů. Po nalezení cíle je nutné následovat ukazatele na rodiče každého uzlu pro rekonstrukci nalezené cesty.

4.4 Automatické rozvržení tabulek

Přidávání již existujících tabulek v databázi je možné dvěma způsoby: automaticky nebo metodou drag and drop. Drag and drop umožňuje exaktně určit pozici přidané tabulky v diagramu. Ovšem pokud je nutné přidat větší množství tabulek, stává se tento způsob poněkud nepraktickým. Z toho důvodu je zde možnost zvolit pozici tabulky na plátně automaticky. Implementace

využívá grafové reprezentace, stejně jako výpočet trasy. Tabulky se rozmísťují kolem středu plátna bez vzájemného překrývání.

4.4.1 Sestavení a příprava grafu

Algoritmus pro rozvržení tabulek na plátno, aniž by došlo k jejich překrytí, využívá opět reprezentaci plátna pomocí mřížky, stejně jako tomu je u vyhledávání trasy (viz kapitola 4.3.1). Převod na graf ovšem není pro správnou funkci vysloveně nutný, celý postup by bylo možné realizovat pomocí matice obsahující jen čísla.

```
   Input  : Grid G, Width of grid W, Height of grid H
   Output: Preprocessed grid G for free rectangle search
 1 for i ← 0 to W − 1 do
 2 |   counter ← 1;
 3 |   for j ← H − 1 to 0 do
 4 |   |   current ← G[i, j];
 5 |   |   if current is obstacle then
 6 |   |   |   counter ← 1;
 7 |   |   |   continue;
 8 |   |   end
 9 |   |   current.Metric ← counter;
10 |   |   counter ← counter + 1;
11 |   |   j ← j − 1;
12 |   end
13 |   i ← i + 1;
14 end
```

Algoritmus 3: Příprava mřížky pro vyhledávání

Před samotným vyhledáváním volné pozice je nutné provést přípravu dat v mřížce. Postup přípravy mřížky demonstruje algoritmus 3. Mřížku je nutné projít nejprve po sloupcích, vždy směrem vzhůru pro jednotlivé sloupce, a každou volnou buňku opatřit číslem vyjadřujícím její vzdálenost od nejbližší nižší překážky. Začínáme číslem 1, počítadlo zvyšujeme s každou volnou buňkou ve sloupci, pokud narazíme na překážku, počítadlo opět nastavíme na počáteční hodnotu.

4.4.2 Nalezení volné pozice

V připravené mřížce je nyní možné vyhledávat volné obdélníky. Proces hledání popisuje algoritmus 4, vstupem je připravená mřížka a minimální rozměry volné plochy, výstupem je souřadnice levého horního rohu tabulky pro umístění do volné plochy. Mřížku procházíme po řádcích zleva a hledáme posloupnost čísel v řádku, pro která platí, že všechny z nich jsou rovny nebo vyšší minimální výšce hledané plochy. Počet takových čísel v řádku musí být roven nebo vyšší než je minimální šířka hledané plochy. Nyní zbývá pouze zvolit jednu z nalezených volných ploch. Pro tento účel byla opět zvolena metoda Manhattan 1 k určení metriky levého horního bodu

obdélníkové plochy vzhledem ke středu plátna. Bod s hodnotou metriky lepší než je průměrná hodnota je zvolen jako výsledný počáteční bod hledaného obdélníku.

Input : Grid G, Width of grid W, Height of grid H, Minimal width MW, Minimal height MH
Output: Top-left point of free rectangles

```
1  nodes ← empty collection;
2  for i ← 0 to H − 1 do
3  |  for j ← 0 to W − 1 do
4  |  |  current ← G[i, j];
5  |  |  line ← empty collection;
6  |  |  for k ← j to j + MW − 1 do
7  |  |  |  line.Add(G[k, j])
8  |  |  end
9  |  |  if ∀c ∈ line c.Metric >= MH and c is free then
10 |  |  |  nodes.Add(current);
11 |  |  end
12 |  |  j ← j + 1;
13 |  end
14 |  i ← i + 1;
15 end
16 foreach node N of nodes do calculate Manhattan(N, center);
17 Result ← cell where Manhattan(cell, center) < average value;
```

Algoritmus 4: Nalezení volné plochy o minimálních rozměrech

4.5 Export diagramů

Grafickou podobu diagramu je možné exportovat ve formátu PNG a XPS. Každý formát má své výhody, pokud uložíme diagram do PNG, není nutné exportovat celé plátno, ale jen využitou plochu. Ve formátu XPS dojde k exportu celého plátna, nicméně tento typ exportu je velmi rychlý, z důvodu vektorové reprezentace plátna ve WPF aplikacích. Nástroj umožňuje také tvorbu DDL skriptu pro objekty v diagramu, tato volba je využitelná v kombinaci s dalšími nástroji, které umožňují použít DDL skript jako jeden z možných formátů pro import.

5 Návrh a implementace datové vrstvy

Grafický editor nabízí mnoho operací pro úpravu relačního schématu skrze GUI. Pro provedení těchto akcí je nutné nejprve sestavit validní DML příkaz, který změnu ve schématu provede. Nástroj aktuálně podporuje dva SŘBD: Oracle Database a Microsoft SQL Server. Navzdory tomu, že SQL je standardizované, syntaxe některých příkazů se liší. Z důvodů těchto odlišností bylo potřeba navrhnout způsob synchronizace s relačním schématem tak, aby změny provedené v diagramu mohly být promítnuty do relačního schématu pomocí jednotného rozhraní. Datová vrstva je navrhnuta s ohledem na možné budoucí rozšíření o další SŘBD.

5.1 Zdroj informací o relačním schématu

Informace o podobě relačního schématu nalezneme v tzv. systémovém katalogu. Systémový katalog je souhrn tabulek, které uchovávají metadata popisující strukturu databáze, nalezneme zde informace o veškerých databázových objektech. Pro uživatele je zpřístupněn systémový katalog pouze pro čtení použitím pohledů nad samotnými tabulkami. Pokud chceme upravovat strukturu databáze, je nutné využít DDL příkazů. Systémový katalog jednotlivých SŘBD nabízí podobné možnosti, ale struktura a názvy pohledů se mohou lišit. Přístup k systémovému katalogu je implementován pomocí vlastních SQL příkazů. Pro tuto činnost tedy nejsou využity externí knihovny.

5.1.1 Systémový katalog Microsoft SQL Server

Pohledy nad systémovými tabulkami jsou součástí schématu `sys`. V aplikaci se pracuje pouze s některými z těchto pohledů. Pro zjištění často používaných informací jsou zde implementovány uložené procedury, které práci značně ulehčují. Pro zjištění informací o primárním klíči tabulky je možné použít proceduru `sp_pkeys`, pro cizí klíče existuje procedura `sp_fkeys`. V aplikaci se pracuje s těmito pohledy:

- `databases` - Pohled obsahuje informace o databázích.
- `tables` - Tento pohled slouží pro zjištění informací o tabulkách, jako je např. název a identifikátor.
- `columns` - Zde zjistíme informace o sloupcích tabulky. Často se používá v kombinaci s pohledem `types`.
- `types` - Tento pohled obsahuje informace o datových typech.
- `foreign_keys` - Pohled obsahuje informace o cizích klíčích.

5.1.2 Systémový katalog Oracle Database

Tabulky a pohledy systémového katalog jsou vlastněny uživatelem `SYS`. Oracle Database rozlišuje tři typy pohledů, pomocí těchto prefixů:

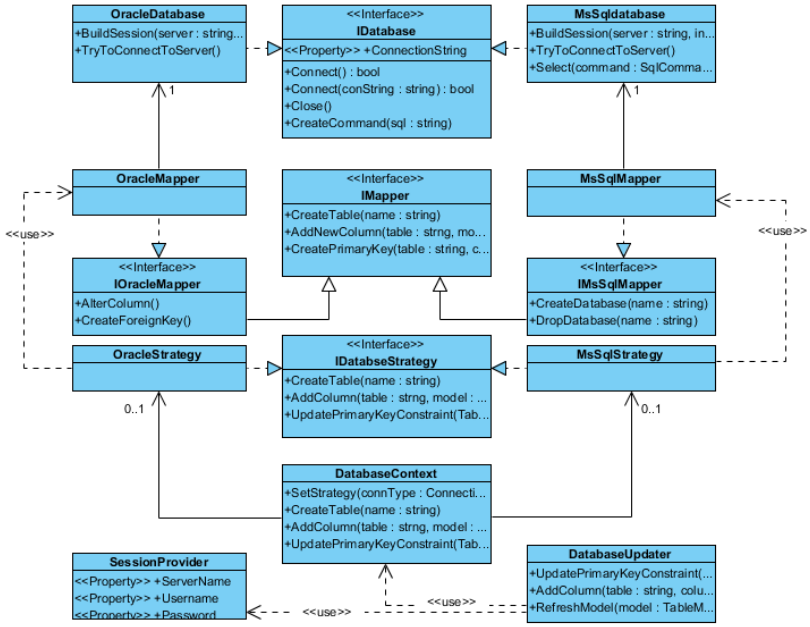
- `USER` - Pohledu uvozeny tímto prefixem obsahují informace o objektech, které vlastní přihlášený uživatel.
- `ALL` - Tyto pohledy obsahují veškeré objekty, ke kterým má přihlášený uživatel přístup, nemusí být jejich vlastníkem.
- `DBA` - Zde nalezneme informace o veškerých objektech databáze. Pohledy uvozeny tímto prefixem mohou obsahovat detailnější informace.

V nástroji jsou jako zdroj informací použity pohledy uvozené prefixem `ALL`. Pro získání informací o schématu databáze jsou použity tyto pohledy:

- `ALL_OBJECTS` - Pohled slouží k získání informací o veškerých objektech, které jsou uživateli přístupné. Používá se pro zjištění informací o tabulkách.
- `ALL_TAB_COLUMNS` - Tento pohled slouží pro zjištění informací o sloupcích tabulky včetně informací o datovém typu.
- `ALL_CONSTRAINTS` - Zde zjistíme informace o integritních omezeních.
- `ALL_CONS_COLUMNS` - Tento pohled obsahuje informace o sloupcích, které figurují v integritních omezeních.
- `FOREIGN_KEYS` - Pohled obsahuje informace o integritních omezeních v podobě cizího klíče.

5.2 Struktura datové vrstvy

Strukturu datové vrstvy popisuje diagram na obrázku 28. Jádrem datové vrstvy jsou třídy `MsSqlMapper` a `OracleMapper`. Tyto třídy (tzv. *mappery*) realizují sestavení DML příkazů pro úpravu schématu, stejně tak jsou zodpovědné za přístup k datům systémového katalogu. Připojení k databázi realizují třídy `OracleDatabase` a `MsSqlDatabase`, tyto třídy nabízejí jednotné rozhraní.



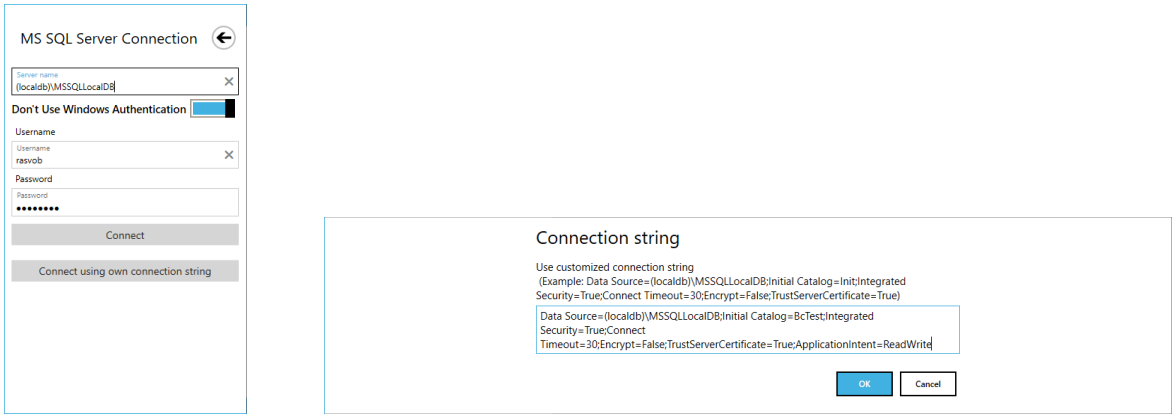
Obrázek 28: Část třídního diagramu - struktura datové vrstvy

V nástroji se obvykle nepřistupuje k mapperům přímo, ale kvůli nutnosti přípravy dat před některými operacemi v závislosti na SŘBD, se kterým aktuálně pracujeme, je přístup zapouzdřen pomocí návrhového vzoru Strategy. Strategie (třídy implementující rozhraní `IDatabaseStrategy`) představuje primární rozhraní přístupu k mapperům pro provedení změn v relačním schématu. Každému mapperu odpovídá samostatná strategie, kontext je definován typem aktuální relace ve třídě `DatabaseContext`. Struktura se samostatnými strategiemi pro jednotlivé mappery je zvolena z důvodu nutného předzpracování dat v případě některých operací. Pokud by k tomuto jevu nedocházelo, bylo by možné namísto strategie použít vzor Factory a s data mappery pracovat přímo.

Datová vrstva je využívaná v doménové logice prostřednictvím třídy `DatabaseUpdater`. `DatabaseUpdater` pracuje se strategiemi pomocí třídy `DatabaseContext`. Nejprve je nutné strategii zvolit pomocí metody `SetStrategy` ve třídě `DatabaseContext`, poté může následovat volání metod, které jsou strategiemi vykonávány (viz kapitola 5.4.2). Informace o relaci uchovává třída `SessionProvider`, která je třídou `DatabaseUpdater` využívána. Toto uspořádání tříd pro provádění změn v relačním schématu eliminuje nutnost rozhodovacích bloků při každém volání metody data mapperu, stejně tak nabízí snadný způsob rozšíření nástroje o nové SŘBD. Stačí pouze přidat nový data mapper, který implementuje požadované rozhraní a vytvořit pro něj novou strategii.

5.3 Sestavení aktuální relace

Informace nutné pro připojení k databázi se částečně liší pro různé SŘBD. Pro vytvoření relace je nutné zkonstruovat řetězec pro připojení na základě přihlašovacích údajů. Za tuto činnost je zodpovědná třída `SessionProvider` (viz obrázek 28). Kromě údajů potřebných k úspěšné autentizaci je zde uveden také SŘBD, ke kterému jsme v aktuální relaci připojeni. Tato informace je zásadní pro konstrukci řetězce s přihlašovacími údaji k serveru, protože ty se liší pro každý SŘBD. Stejně tak se liší i třídy pro jeho sestavení z uvedených informací. Standardně jsou přihlašovacími údaje zadávány pomocí GUI (viz obrázek 29a), nicméně může dojít k situaci, kdy potřebujeme využít parametry, které GUI nenabízí (viz obrázek 29b). V tomto případě je možné zadat řetězec vlastní a třída `SessionProvider` s informacemi o relaci zprostředkovává tento řetězec pro připojení, aniž by jej bylo nutné sestavovat, jako je tomu v předchozím případě.



(a) Dialog pro nastavení relace

(b) Dialog pro zadání vlastního řetězce pro připojení

Obrázek 29: Zadání přihlašovacích údajů

5.4 Synchronizace s relačním schématem

Aktualizace existujícího schématu je prováděna okamžitě po provedení změny v ER diagramu. Nevyužívá se tedy odložená aktualizace, jako u většiny testovaných nástrojů. Nástroj podporuje práci se schématem použitím několika diagramů současně, není nutné diagram generovat pro všechny tabulky, ale stačí přidat ty, se kterými potřebujeme pracovat. Tento způsob je výhodný pro rozsáhlá schémata, kde rozdělení schématu na určité logické celky zpřehlední vizualizaci a usnadní tak proces aktualizace. Okamžité provedení aktualizace je důležité právě z důvodu práce s několika diagramy najednou. Jinak by mohlo dojít k situaci, kdy pro provedení změn záleží na pořadí jejich potvrzení, což může mít za následek nemožnost vykonání některých z nich, jako je tomu např. v nástroji Microsoft SQL Server Management studio.

5.4.1 Problém odložené aktualizace

K problémům s odloženou aktualizace zpravidla dochází vlivem konfliktu integritních omezení nebo odstraněním atributu. Příklad takového problému vidíme v tabulce 5.

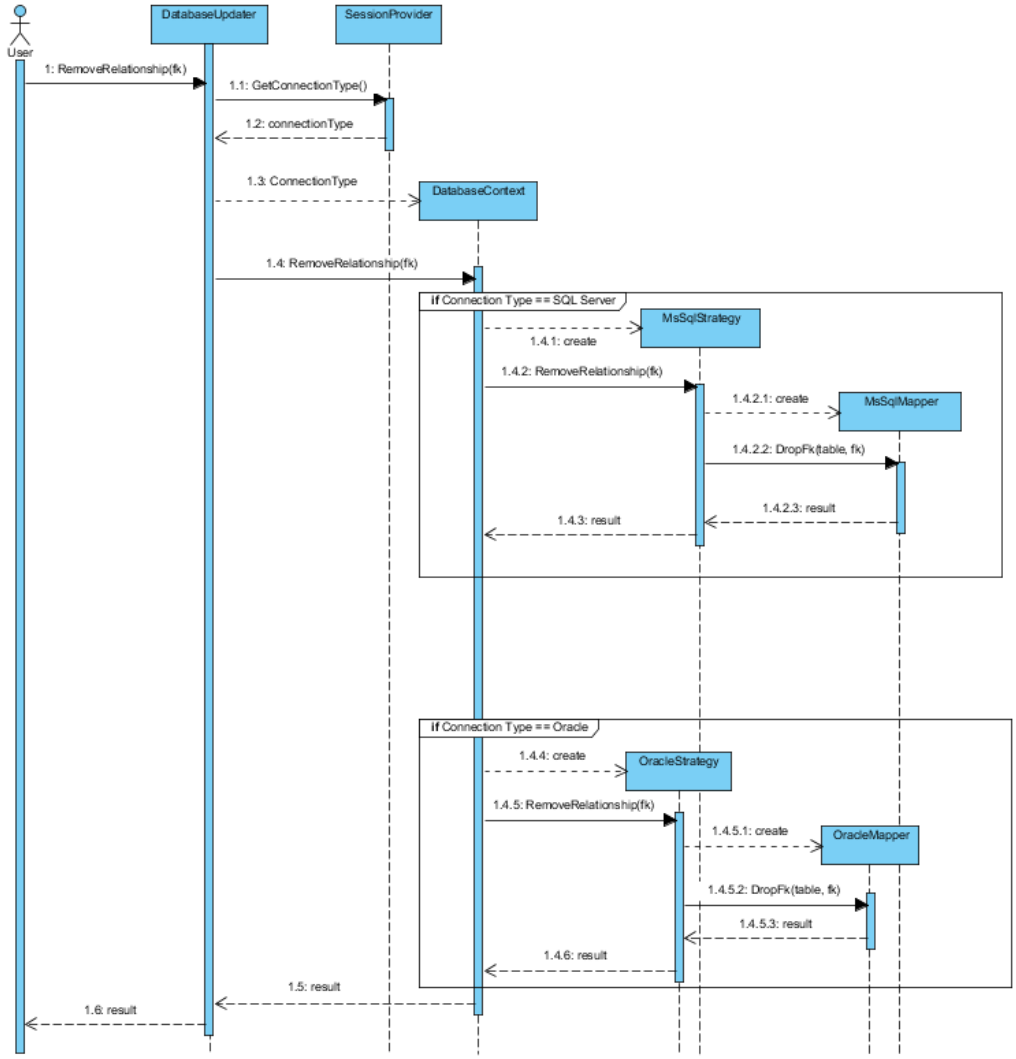
Tabulka 5: Příklad problému odložené aktualizace

Čas	Diagram 1	Diagram 2
`t1`	Povolení `NULL` hodnot atributu `X`	
`t2`		Nastavení atributu `X` primárním klíčem
`t3`		Potvrzení změny
`t4`	Změnu není možné potvrdit	

V čase `t4` není možné změnu potvrdit, diagram 1 navíc není v souladu s existujícím schématem. Odstranění problému závisí na možnostech použitého nástroje.

5.4.2 Aktualizace schématu

Celý proces aktualizace schématu začíná provedením změny v diagramu pomocí grafického editoru. Po potvrzení změny uživatelem je nutné provést adekvátní SQL příkaz. Sekvenční diagram na obrázku 30 zachycuje odstranění vztahu mezi tabulkami z pohledu datové vrstvy. Na základě SŘBD (informaci poskytuje třída `SessionProvider`), se kterým aktuálně pracujeme, je zvolen data mapper, který příkaz provede. Ten je zodpovědný i za sestavení SQL příkazu. Operace data mapperu je volána prostřednictvím rozhraní třídy `DatabaseContext`, který vybere konkrétní strategii pro práci s příslušným data mapperem.

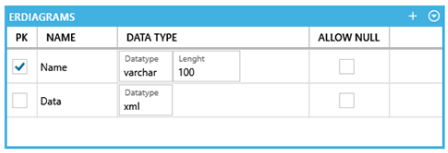


Obrázek 30: Sekvenční diagram - odstranění vztahu

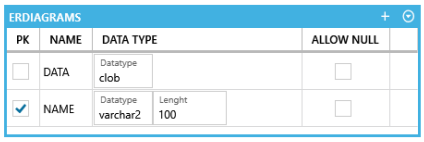
Při vykonávání příkazu může samozřejmě dojít k chybě, ta může být zaviněna jak chybou vstupu na straně uživatele, tak např. chybou sítě. Tyto chyby vyvolávají výjimky v kódu, proto probíhá přístup k metodám třídy `DatabaseContext` přes třídu `DatabaseUpdater`. Tato třída má na starosti zachycení výjimek. O chybách je uživatel informován formou dialogového okna a také jsou zapsány do logu, který se nachází v samostatném panelu. Do tohoto logu jsou vypisovány také prováděné SQL příkazy pro snazší identifikaci chyb, pokud změna neproběhla úspěšně. V případě chyby je nutné diagram uvést do původního stavu. Pokud se jedná o chybu při úpravě struktury tabulky, dojde k aktualizaci dat doménové vrstvy opětovným načtením hodnot ze systémového katalogu. Pokud dojde k chybě při provádění operace, která způsobuje odstranění nebo přidání objektu na plátno, není opětovné načtení dat potřeba, protože k operacím s objekty na plátně dochází jen po úspěšném dokončení operace.

5.5 Serializace diagramů

Nástroj podporuje ukládání rozpracovaných diagramů přímo do speciální tabulky v databázi, ke které se diagram vztahuje. Oproti klasickému ukládání na pevný disk ve formě souboru umožňuje tento způsob zpřístupnění diagramu dalším osobám pracujícím s databází, aniž by bylo nutné přenášet soubory projektu. Navíc je možné pokračovat v práci z jiného počítače, protože je potřeba jen připojení k databázi, kde se diagram nachází. Výhodou je také zálohování, protože diagramy je možné zálohovat v rámci záloh databáze. Strukturu tabulky pro Microsoft SQL Server zachycuje obrázek 31a, pro Oracle Database pak obrázek 31b.



(a) Tabulka v Microsoft SQL Server



(b) Tabulka v Oracle Database

Obrázek 31: Struktura tabulky pro ukládání diagramů

5.5.1 Uložení diagramu

Serializace diagramu probíhá na úrovni viewmodelů, protože ty obsahují veškerá potřebná data týkající se struktury diagramu. Serializaci podléhají veškeré vlastnosti diagramu, které nelze získat z existujícího schématu. Jedná se tedy o souřadnice objektů na plátně, jejich velikost a samozřejmě také názvy a identifikátory, aby bylo možné provést načtení příslušných polí ze systémového katalogu. Tato struktura je uložená ve formátu XML. Příklad serializovaného vztahu ve formě XML elementu vidíme na výpisu kódu 4.

```
<ConnectionInfoViewModel SourceConnectorOrientation="Down"
    DestinationConnectorOrientation="Down">
  <RelationshipModel Id="661577395" Name="Objednany_zajezd_Zajezd_FK"
      LastModified="2017-01-21T15:17:53.68">
    <Source>
      <TableModel Title="Zajezd" Id="453576654" />
    </Source>
    <Destination>
      <TableModel Title="Objednany_zajezd" Id="421576540" />
    </Destination>
  </RelationshipModel>
  <Points>
```

```
        <ConnectionPoint X="1060" Y="1173" />
        <ConnectionPoint X="1060" Y="1200" />
        <ConnectionPoint X="1500" Y="1200" />
        <ConnectionPoint X="1500" Y="832" />
    </Points>
</ConnectionInfoViewModel>
```

Výpis 4: Serializovaný vztah ve formátu XML

5.5.2 Načtení diagramu

Načtení diagramu je rozděleno na dvě fáze. V první fázi je nutné provést kontrolu existence objektů v databázi, aby se nezobrazovaly již neexistující tabulky nebo integritní omezení v podobě cizích klíčů, po této kontrole jsou načtena data ze systémového katalogu. Tímto je zamezeno nekonzistenci načteného diagramu a relačního schématu, stejný princip je využit i při změně aktivního diagramu při práci s několika diagramy zároveň, aby zobrazený diagram ukazoval aktuální podobu schématu. V druhé fázi je potřeba přidat objekty na plátno a nastavit jim požadované vlastnosti tak, aby byla struktura načteného diagramu stejná jako při ukládání.

6 Závěr

Cílem této práce bylo vytvořit aplikaci umožňující návrh schématu relační databáze formou ER diagramu, která je schopna pracovat s SŘBD Microsoft SQL Server a Oracle Database. Práce je rozdělené na dvě části, první část se zaměřuje na problematiku notací ER diagramů z hlediska využívaných konstrukčních prvků, jejich rozdílů a využití samotných ER diagramů společně s nevýhodami. Součástí první části práce je také porovnání existujících nástrojů pro návrh relačního schématu databáze, nástroje jsou porovnány nejen na základě možností grafického editoru pro práci s ER diagramy, ale porovnání se zaměřuje také na oblast synchronizace diagramu s existujícím relačních schématem.

Druhá část bakalářské práce se zabývá návrhem a implementací grafického editoru vyvíjeného nástroje, včetně synchronizace diagramů s relačním schématem. V práci je detailně popsáno fungování klíčových částí grafického editoru pro práci s ER digramy, včetně problematiky algoritmizace některých činností. Datová vrstva pro přístup k systémovému katalogu SŘBD je navržena s ohledem na možná budoucí rozšíření o podporu dalších SŘBD. Návrh obou částí je doplněn o části třídního diagramu umožňující snazší popis struktury těchto částí.

V rámci implementace jsem narazil na několik problémů souvisejících s limitujícím výkonem WPF aplikací, nicméně se tyto problematické částí podařilo zoptimalizovat a nástroj tak, dle mého názoru, splnil cíle stanovené na počátku vývoje. Aplikace byla navrhnuta s ohledem na možná rozšíření, kterých se nabízí celá řada, nejzásadnějším z nich je podpora práce s více SŘBD, než je tomu doposud. Za další užitečná rozšíření považuji návrhář pohledů, možnost práce v offline režimu nebo export do formátů, které podporují další nástroje v této oblasti.

Téma bakalářské práce pro mne bylo velice přínosné, protože jsem se díky němu naučil pracovat s mnoha technologiemi v oblasti vývoje aplikací pro platformu Microsoft Windows. Díky této práci jsem nabyl nové znalosti také v oblasti správy databázových systémů a využití návrhových vzorů v rámci návrhu a implementace rozsáhlejší aplikace.

Literatura

[1] CHEN, Peter Pin-Shan. *The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems* [online]. 1(1), 9-36 [cit. 2017-03-10]. Dostupné z: `http://dspace.mit.edu/bitstream/handle/1721.1/47432/entityrelationshx00chen.pdf`

[2] KRÁTKÝ, Michal a Radim BAČA. *Databazové systémy* [online]. [cit. 2017-03-11]. Dostupné z: `http://dbedu.cs.vsb.cz/SubPages/OpenFile.aspx?file=book/dbcb.pdf`

[3] SONG, Il-Yeol, Mary EVANS a E.K. PARK. *A Comparative Analysis of Entity-Relationship Diagrams* [online]. [cit. 2017-03-11]. Dostupné z: `http://www.cci.drexel.edu/faculty/song/publications/p_Jcse-erd.PDF`

[4] BACHMAN, C. W. *Data structure diagrams* [online]. [cit. 2017-03-11]. Dostupné z: `http://www.minet.uni-jena.de/dbis/lehre/ws2005/dbs1/Bachman-DataStructureDiagrams.pdf`

[5] Data structure diagram. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-11]. Dostupné z: `https://en.wikipedia.org/wiki/Data_structure_diagram`

[6] Lucidchart. *What is an Entity Relationship Diagram* [online]. [cit. 2017-03-10]. Dostupné z: `https://www.lucidchart.com/pages/er-diagrams`

[7] Lucidchart. *ER Diagram Symbols and Notation* [online]. [cit. 2017-03-10]. Dostupné z: `https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning`

[8] Amit's A* Pages. *Pathfinding* [online]. [cit. 2017-03-21]. Dostupné z: `http://theory.stanford.edu/~amitp/GameProgramming/`

[9] Visual Studio Magazine. *Priority Queues with C#* [online]. [cit. 2017-04-06]. Dostupné z: `https://visualstudiomagazine.com/Articles/2012/11/01/Priority-Queues-with-C.aspx`

[10] Microsoft. *Introduction to WPF in Visual Studio 2015* [online]. [cit. 2017-04-07]. Dostupné z: `https://msdn.microsoft.com/en-us/library/aa970268.aspx`

[11] Microsoft. *Developer's Guide to Microsoft Prism Library 5.0 for WPF* [online]. [cit. 2017-04-07]. Dostupné z: `https://msdn.microsoft.com/en-us/library/gg406140.aspx`

A Struktura přiloženého optického média

Tabulka 6: Obsah optického média

Adresář	Obsah
\src	Projekt s aplikací ve Visual Studiu 2015
\bp	PDF soubor s textem bakalářské práce
\setup	Instalační soubory aplikace

B Použité knihovny třetích stran

- MahApps.Metro - `https://github.com/MahApps/MahApps.Metro`
- Extended WPF Toolkit Community Edition - `http://wpftoolkit.codeplex.com/`

C Testované CASE nástroje a SŘBD zmíněné v textu

Tabulka 7: Stávající CASE nástroje

Název	Výrobce	Oficiální web
Oracle SQL Developer Data Modeler	Oracle Corporation	`http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/index.html`
Toad Data Modeler	Quest Software	`https://www.quest.com/products/toad-data-modeler/`
SQL Server Management Studio	Microsoft	`https://www.microsoft.com/en-us/sql-server/default.aspx`
Visual Paradigm	Visual Paradigm International	`https://www.visual-paradigm.com/`
StarUML 2	MKLab	`http://staruml.io/`

Tabulka 8: Zmíněné SŘBD

Název	Výrobce	Oficiální web
Oracle Database	Oracle Corporation	`https://www.oracle.com/database/`
Microsoft SQL Server	Microsoft	`https://www.microsoft.com/en-us/sql-server/default.aspx`
DB2	IBM	`https://www.ibm.com/analytics/us/en/technology/db2/`
PostgreSQL	The PostgreSQL Global Development Group	`https://www.postgresql.org/`
MySQL	Oracle Corporation	`https://www.mysql.com/`