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 |
| CASE | – | Computer-Aided Software Engineering |
| ERD | – | Entity-Relationship Diagram |
| SŘDB | – | 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.

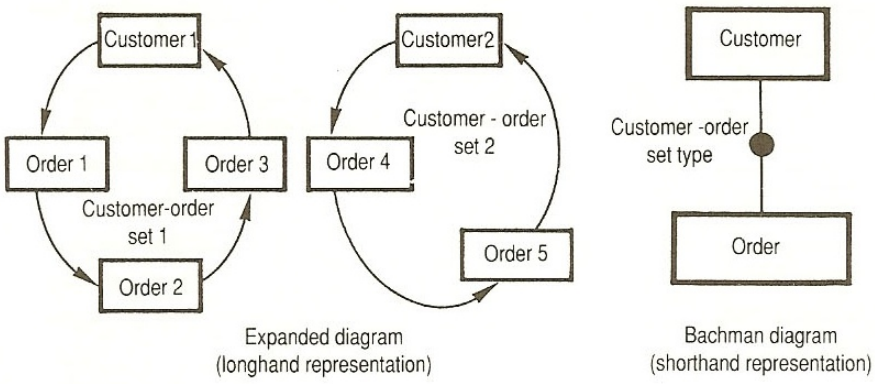
Tato bakalářská práce popisuje návrh a implementaci aplikace pro návrh a aktualizaci schématu relační databáze. Aplikace podporuje SŘDB Microsoft SQL Server a Oracle Database, z toho důvodu je první kapitola věnována popisu notace ER diagramu, který se používá pro vizualizaci entit a vztahů mezi nimi v relačním schématu 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 systémy řízení báze dat.

# 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.

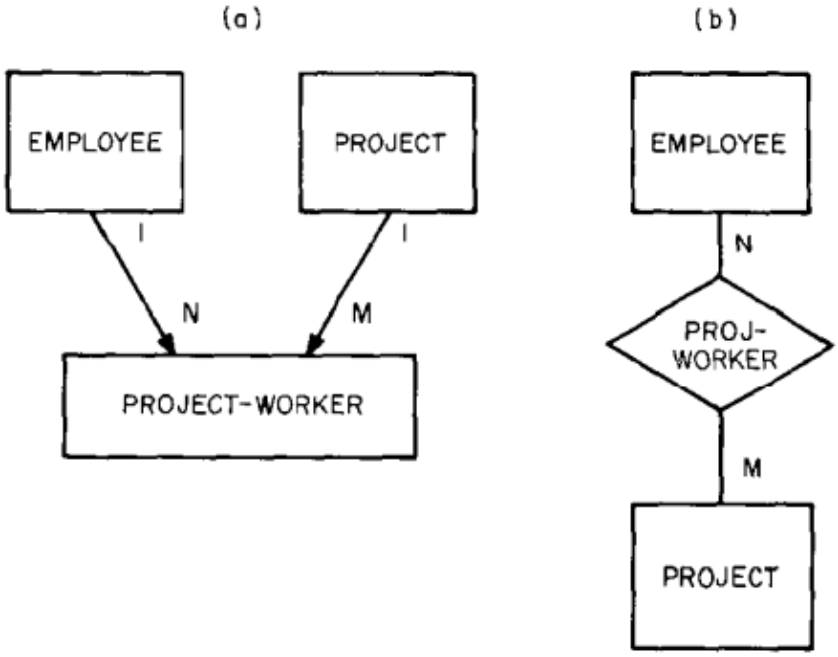
## 2.1 Historie

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 SŘDB.




Obrázek 1: Bachmanův diagram (Převzato z [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. Srovnání těchto dvou typů diagramů nabízí obrázek 2.




(a) Data structure diagram (b) ER diagram

Obrázek 2: Srovnání DSD a ERD (Převzato z [1])

## 2.2 Využití ER diagramu

Nejčastější použití ER diagramů představuje tvorba nových relačních schémat při návrhu databází, jednotlivé typy modelů popisuje kapitola 2.3. Rozhodně se ale nejedná o jedinou oblast využití. Tvorba ER diagramu velmi často stojí na počátku analýzy a návrhu mnoha informačních systému, protože relační databáze jsou obvykle primárním datovým zdrojem těchto systémů pro uložení dat transakčního charakteru. Časté je i využití ER diagramu ve spojitosti s diagramy datových toků pro znázornění uložení dat business procesu.

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 došlo ke změně některých business procesu. Tyto změny obvykle vyvolají nutnost přizpůsobení relačního modelu, ER diagram je tedy vhodný prostředek nejen k analýze stávajícího schématu pro návrh změn, ale také pro jejich provádění použitím CASE nástrojů.

## 2.3 Kategorie modelů

Podle úrovně abstrakce rozlišujeme tři typy datových modelů [6], které využíváme při návrhu databázového schématu. Nejvyšší úroveň abstrakce nabízí konceptuální model, který zachycuje

pouze entity a vztahy mezi nimi, je nejméně detailní. Více detailů nabízí logický model, poskytuje nižší úroveň abstrakce, ale stále je nezávislý na použitém SŘDB. Posledním typem je fyzický model, ten vychází z logického modelu, nicméně se 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 SŘDB fyzické modely více či méně lišit. Přehledné shrnutí rozdílů mezi modely nabízí tabulka 1.

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 modelu 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 vhodný je zejména pro modelování komplexních databází pro geografické informační systémy nebo telekomunikace.

## 2.5 Prvky ER diagramu

V této části jsou popsány prvky používané při tvorbě ER diagramu, včetně speciálních případů těchto prvků a příkladů. Ilustrační grafické znázornění odpovídá notaci podle P. Chena, nicméně v další části budou ostatní používané notace dále rozvedeny [7].

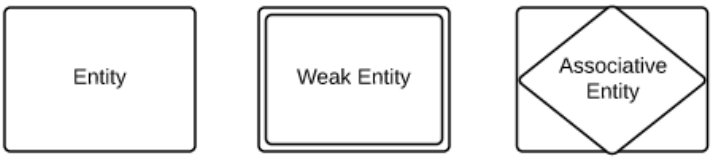
### 2.5.1 Entitní typ

Entitní typy (často označovány v souvislosti s ERD jen jako entity) jsou jednoznačně identifikovatelné objekty vyskytující se v problémové doméně, které jsou pro nás zajímavé svými

atributy, daty, které chceme uchovávat např. *zákazník, oddělení, automobil*. Obvykle se označují podstatným jménem v jednotném čísle.

Rozlišujeme následující typy entit:

- **Silný entitní typ** - muže existovat nezávisle na ostatních entitních typech, protože obsahuje alespoň jeden atribut, který entitu jednoznačně odlišuje od ostatních, značí se obdélníkem
- **Slabý entitní typ** - jeho existence je závislá na jiném entitním typu, protože neobsahuje atribut umožňující jednoznačnou identifikaci, značí se obdélníkem s dvojitým okrajem
- **Asociativní entitní typ** - slouží k vyznačení vztahu mezi entitami, obsahuje atributy identifikující tento vztah, značí se kosočtvercem v obdélníku




Obrázek 3: Znázornění entit podle Chenovy notace [7]

### 2.5.2 Vztah

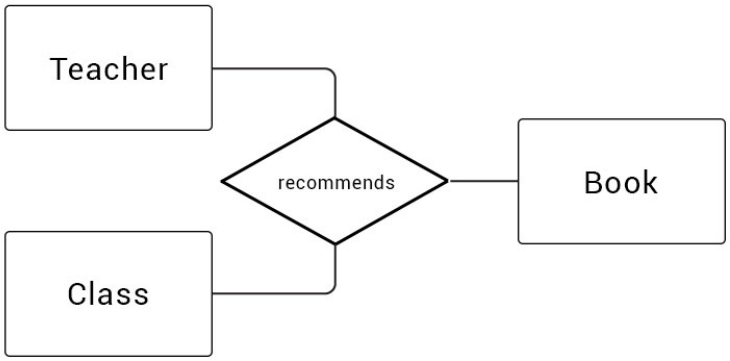
Vztah slouží k vyjádření asociace mezi entitními typy, obecně vyjadřuje informace, které nelze vyjádřit pouhými entitními typy, např. student *studuje* předmět, novinář *napsal* článek. Označuje se slovesem. Znázorňuje se kosočtvercem.

Identifikující vztah je speciální typ vztahu, který spojuje slabou entitu s entitou, na které je závislá, ta se označuje jako vlastník. Značí se kosočtvercem s dvojitým okrajem.

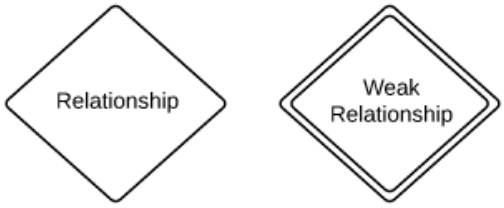
Vztahy dělíme podle počtu entit, které se v nich vyskytují na vztahy:

- **Unární** - ve vztahu vystupuje pouze jedna entita, ta je ve vztahu sama se sebou, např. zaměstnanec *je nadřízeným* jiného zaměstnance
- **Binární** - vztah mezi dvěma entitami, např. zaměstnanec *je vedoucím* oddělení
- **Ternární** - vztah mezi třemi entitami najednou, např. učitel *doporučuje* knihu třídě
- **N-ární** - vztah mezi $n$ entitami zároveň

Obecně se ternární vztahy vyskytují v ER diagramech velmi málo, protože vedou k problémům s dekompozicí na několik binárních vztahů při přechodu z konceptuálního modelu na relační (logický), kdy může dojít ke ztrátě některých informací.




Obrázek 4: Příklad ternárního vztahu




Obrázek 5: Znázornění vztahů podle Chenovy notace [7]

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 následující typy kardinality:

- **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 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.

### 2.5.3 Atribut

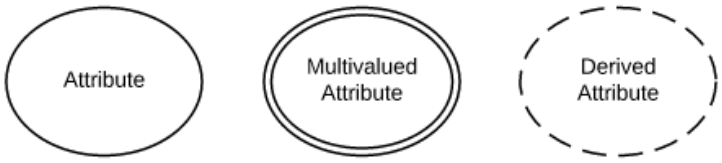
Atributy jsou vlastnosti charakterizující entitní typ. Označují se podstatným jménem, znázorňují se elipsou.

Rozlišujeme následující kategorie atributů:

- **Jednoduchý** - atomický atribut, nelze dále dělit, např. *značka automobilu*
- **Složený** - dělí se na více atomických atributů, např. *adresa* se dělí na *město, ulici a PSČ*
- **Odvozený** - hodnota je vypočtena na základě hodnot ostatních atributů, fyzicky nemusí v relační databázi existovat, znázorněn je elipsou s čárkovanou hranou, např. *průměrná mzda*

Další rozdělení je podle počtu hodnot atributu na:

- **Jednohodnotové** - atribut obsahuje pouze jednu hodnotu, např. *rodné číslo*
- **Vícehodnotové** - atribut může obsahovat více hodnot, značí se elipsou s dvojitým okrajem,např. *telefonní číslo na pevnou linku i mobilní telefon*




Obrázek 6: Znázornění atributů podle Chenovy notace [7]

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

## 2.6 Rozdíly 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.

### 2.6.1 Binární a n-ární modely

Nejzásadnějším rozdílem mezi notacemi je podpora n-árních modelů, z toho důvodů rozlišujeme dva typy modelů, a to 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 je možné 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.

V případě n-árních modelů není nutné vytvářet další entity k přidání atributů pro vztah. N-ární vztahy jsou z hlediska sémantiky výhodné pro modelování některých situací oproti vyjádření pomocí několika binárních vztahů. Při přechodu na modely nabízející nižší úroveň abstrakce, je často nutné tyto n-ární vztahy převést na sekvenci binárních vztahů, což může vést ke ztrátě

cenných informací, protože už i pro ternární vztah M:N:P může způsobů dekompozice na binární vztahy existovat více a na první pohled nemusí být patrná ztráta části informace.

Podpora n-árních vztahů vede k nekompatibilitě mezi jednotlivými notacemi, kdy nemusí být snadné převést model používající jednu notaci na model jiné notace. Z tohoto důvodu dnešní CASE nástroje obvykle podporují pouze jeden typ notace, případně typů podporují více, ale pouze ze stejné kategorie s ohledem na podporu arity vztahů.

### 2.6.2 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 vztahů 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. Tento způsob je použit např. u Min-Max notace, která podle tohoto způsobu převzala i název.

Jednotlivé notace se neliší vyjádřením kardinality a povinnosti pouze graficky, případně textově, ale také tím, na které straně je informace vyjádřena. Rozlišujeme proto dva styly zápisu, první z nich se označuje jako look here, druhý jako look across [3].

Pokud modelujeme situaci, ve které jeden zaměstnanec pracuje v právě 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, tedy 1 zapíšeme na stranu zaměstnance a N na stranu oddělení. To, na které straně je vyjádřena kardinalita a povinnost, nemusí být pro notaci jednotné. Chenova notace používá look across přístup pro vyjádření kardinality, ale povinnost vyjadřuje stylem look here, oproti tomu u notace Min-Max se obvykle dodržuje pouze jeden způsob pro kardinalitu i povinnost.

## 2.7 Nejčastěji používané notace

Jak už bylo zmíněno, notací existuje celá řada. Jednotlivé notace se v mnoha směrech liší, v následující kapitole jsou vybrané notace popsány, včetně příkladu. Příklad modeluje situaci, kdy jeden člověk má právě jedno místo narození a na jednom místě narození se mohlo narodit více lidí.

### 2.7.1 Chenova notace

Jedná se o notaci představenou poprvé roku 1976 Peterem Chenem 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. Grafické znázornění jednotlivých prvků této notace je popsáno v kapitole 2.5.


Obrázek 7: Příklad Chenovy notace [6]

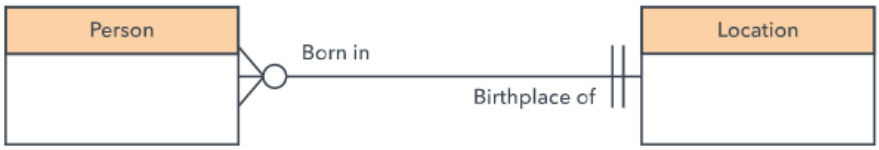
### 2.7.2 Bachmanova notace

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 modeluje cizí klíče už na konceptuální úrovní, 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.


Obrázek 8: Příklad Bachmanovy notace [6]

### 2.7.3 Crow's foot notace

Tato notace se označuje také jako Information engineering nebo Martinova notace. 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 nich 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, detailní popis nalezneme v kapitole 2.7.6. Tato notace je populární zejména v oblasti CASE nástrojů a nalezneme ji v mnohých z nich.




Obrázek 9: Příklad Crow's foot notace [6]

### 2.7.4 Barkerova notace

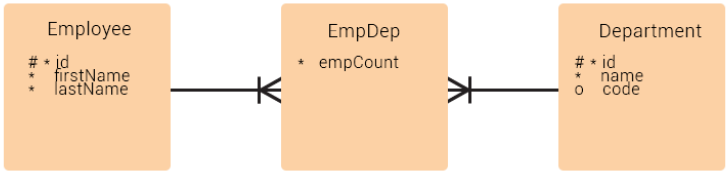
Barkerova notace je velmi populární v oblasti tvorby datových modelů pro Oracle Database, vznikla roku 1981 a po příchodu Richarda Barkera do Oraclu, se stala v této oblasti velmi populární. Jedná se o notaci podporující nejvýše binární vztahy, které jsou vyjádřeny čárami. Plná čára symbolizuje povinný vztah, čárkovaná pak vztah nepovinný. Pro vyjádření kardinality je zde použít také symbol vraní nohy. Povinnost používá styl look here, kardinalita pak styl look across. Atributy jsou zde trojího typu, každý typ označuje kategorii atributu.

Notace používá tyto znaky pro kategorizaci atributů:

- # - identifikující atribut
- ∗ - povinný atribut
- ○ - nepovinný atribut


Obrázek 10: Příklad Barkerovy notace [6]

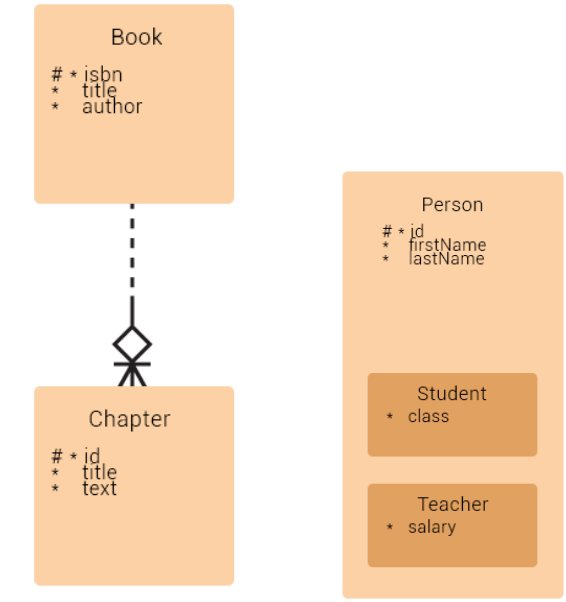
Oproti ostatním notacím zde nalezneme určitá specifika. Prvním z nich je tzv. UID čára, obr. 11, která se kreslí pouze u slabých entitních typů a vyjadřuje, že je identifikován atributy silných entitních typů.




Obrázek 11: Specifika Barkerovy notace - UID

Dalším specifikem, je možnost vyznačit nepřenositelný vztah, obr. 12a, 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 12: Specifika Barkerovy notace

Notace podporuje také dědičnost, kdy podtypy dědí atributy supertypu, obr. 12b, graficky je tento jev vyjádřen přidání obdélníků jednotlivých podtypů do společného obdélníku, který odpovídá supertypu.

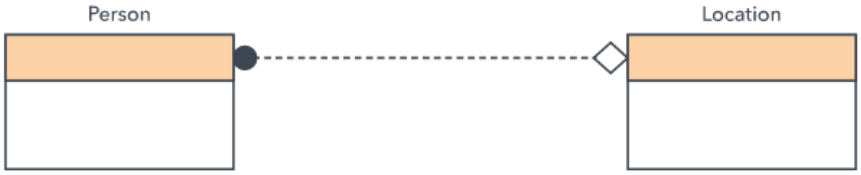
### 2.7.5 Min-Max notace

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, které musí ve vztahu participovat, určuje povinnost, maximální počet určuje kardinalitu. Styl pro kardinalitu a povinnost se používá jednotný, zpravidla look here.


Obrázek 13: Příklad Min-Max notace [6]

### 2.7.6 IDEF1X notace

IDEF1X 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, a nejčastěji používá styl look across. Tato notace rozlišuje graficky silné entitní typy, značí se obdélníkem, a slabé entitní typy, značí se obdélníkem se zakulacenými rohy. 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, zbytek prvků IDEF1X je ponechán.




Obrázek 14: 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 vhodný nástroj k vizualizaci jejich struktury. Toto omezení platí i pro částečně strukturovaná data, nejznámějšími zástupci jsou XML nebo JSON, protože navzdory tomu, že jsme schopni rozlišit jednotlivé entity, může dojít k situaci, kdy se entity téhož typu liší svými atributy.

## 2.9 Shrnutí

Každá notace má své klady i zápory, nelze tedy říci, že je některá notace je lepší než jiná, 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, dojde sice k snížení úrovně abstrakce, nicméně konverze na fyzický model je poté jednodušší. Posledním kritériem je možnost modelování subtypů 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 softwarovém 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, podporujících 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 na základě vytvořeného diagramu.

## 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 nich se nezastavuje u pouhého generování skriptů pro nasazení nové databáze, ale využívá reverzního inženýrství k 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. Tyto možnosti obvykle vedou ke značnému zvýšení produktivity.

Použití CASE nástrojů s sebou nenese jen samá pozitiva, ale skýtá i řadu nedostatků. Největším problémem je obvykle nulová podpora pro provedení formální analýzy vytvořeného modelu, nástroje tedy 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 SŘDB se stala primárním požadavkem při volbě porovnávaných nástrojů. Každý z následujících nástrojů tedy podporuje alespoň jeden z dvojice dříve zmíněných databázových systémů. Nástroje jsou porovnány na základě mnoha kritérií, mezi jinými např. schopnost reverzního generování ER diagramů nebo způsob synchronizace se stávajícím schématem databáze. Každý z následujících nástrojů nabízí alespoň trial verzi pro otestování jeho funkcionality, v ideálním případě je možné pořídit licenci pro studenty zdarma nebo využít bezplatné community edice.

### 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, ovšem ten je zaměřen na správu existujících Oracle databázi, takže možnosti modelování jsou oproti samostatné instalaci Data Modeleru mírně limitovány.

Data Modeler nabízí tvorbu modelů na dvou úrovních abstrakce, prvním je logický model a druhým 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, která je v oblasti Oracle databází nejpoužívanější, nicméně ji lze přepnout na Bachmanovu nebo Information engineering. 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 mírně modifikovaná Barkerova 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 Data Modeleru, můžeme exportovat do CSV, v tomto případě vznikne hned několik CSV souborů, rozdělených podle jednotlivých prvků diagramu. Dalšími možnostmi je XMLA nebo Cube View Metadata, používané v kombinaci s DB2.

Nástroj podporuje vygenerování DDL skriptu na základě relačního modelu. Před exportem je nutné nastavit požadované SŘDB, v nabídce je Oracle Database, Microsoft SQL Server a IBM DB2, všechny zmíněné jsou podporovány v několika verzích z důvodu překlenutí specifik jednotlivých verzí. Exportování DDL skriptu je doprovázeno kontrolou chyb, ať už se jedná o varování nebo errory, 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 nativního formátu Data Modeleru, ale také na základě DDL skriptu. Tato možnost je výhodná v případě vizualizace již existujících schémat. Ovšem za nejvíce praktickou 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.

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 je nutné synchronizaci spustit ručně. Tento způsob implementace umožňuje nepotvrzené změny jednoduše vrátit zpátky synchronizací diagramu s aktuálním schématem databáze.

Grafický editor reaguje při manipulaci s objekty poměrně rychle, nicméně při přesunu entity nepříjemně blikají, stejně tak korekce čar, po dokončení manipulace s entitou, způsobuje problémy. Konce čar se přemísťují aniž by k tomu byl zjevný důvod, entita byla posunuta jen o

několik pixelů, případně jsou vlivem otočení špatně rozlišitelné symboly pro znázornění kardinality.

V průběhu používání jsem se setkal s několika problémy, nejčastěji souvisely s grafickým editorem diagramů. Nepříjemná je drag and drop funkcionalita pro přidání existujících tabulek do diagramu, místo 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, sice došlo k vytvoření asociační tabulky, ale bohužel přibyly atributy cizích klíčů také v obou tabulkách z logického modelu. V logickém modelu je tedy nutné nejprve vytvořit ručně asociační entitu, byť neobsahuje žádné neklíčové atributy, a vztah převést na dvojici vztahů 1:N. Po tomto opatření již proběhne převod správně.

Nepříjemná je také nutnost vytvoření nového relačního schématu ještě před forward engineeringem, 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 *undo* tento problém neřeší.

### 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 SŘDB v několika verzích, mezi nimi např. Oracle Database, Microsoft SQL Server nebo PostgreSQL.

Toad umožňuje vytvořit 3 typy modelů, nejpoužívanější je fyzický model, který je cílen na konkrétní SŘDB, 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 SŘDB, případně pokud určený SŘDB není mezi podporovanými. Notace jsou tentokrát v nabídce pouze dvě, implicitně nastavený Information Engineering a volitelná IDEF1X.

Export je možné provést do CSV, podle mého názoru je export řešen lépe než v Data Modeleru, protože CSV soubor je exportován pouze jeden a prvky jsou odděleny vnitřní strukturou souboru. Další možnosti je export do Excelu, který prvky rozděluje do samostatných sešitů. 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 SŘDB 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 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 SŘDB než z jakého byl model importován.

Import modelu je možný nejen z CSV, Excelu 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 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.

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, samozřejmě je 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.

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 jejich křížení, vyhledávání tras je velmi rychle a dokonce ani po zapnutí volby pro co nejpřesnější trasy nepřesáhl čas, dle mého odhadu, jednu sekundu.

Jediná věc co bych Toad Data Modeleru vytkl 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. 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 displayi s nižším rozlišením,

nicméně přesunutí docků fungovalo bez problémů a na displayi s FullHD rozlišením se problém neprojevil.

### 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. Jedná se o plnohodnotný nástroj pro administraci SQL Serveru, 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 SŘDB.

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

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, Excelu nebo XML, jako je tomu u konkurenčních nástrojů.

Diagramy jsou uloženy ve speciální tabulce 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 pouze vybranou 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, kterou bych mimo testování prováděl jen těžko.

Vybraná část schématu, která je součásti diagramu, bohužel nejde exportovat ve formě DDL skriptu, je možné vytvořit pouze DDL skript pro jednotlivé tabulky a tyto dílčí části zkompletovat v textovém editoru podle potřeby.

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 autolayout, podobně jako u Toad Modeleru, a celé schéma nechat přehledně uspořádat.

Žádné problémy s grafickým editorem jako je špatné uspořádání čar, blikání objektů a podobně se nevyskytly. Práce je velmi intuitivní a editor neobsahuje zbytečné funkce. Právě tato jednoduchost má největší vliv na uživatelský komfort, který je podle mě na nejlepší úrovní 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 samozřejmě 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..

Visual Paradigm podporuje tvorbu logických modelů bez nutnosti specifikovat SŘDB, ten je nutné uvést až při generování skriptu a nástroj podporuje více než deset různých systémů. Diagram používá Crow's foot notaci, bohužel je jediná podporovaná a nelze ji přepnout jako v případě jiných nástrojů.

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, dále Excel, 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 rozřezání diagramu na části specifikované svou velikostí. V případě XML můžeme ovlivnit jeho podobu volbou struktury, na výběr máme reprezentaci vlastností objektu samostatnými elementy nebo atributy příslušného elementu.

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 dalších 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.

Definování nového připojení k databázi je bohužel 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, to způsobilo v případě pomalé odezvy serveru zamrznutí GUI i na několik sekund.

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.

### 3.2.5 StarUML 2

StarUML je poměrně nový nástroj, první verze byla vydaná roku 2014, který 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 trial verzi, licence pro studenty není zdarma, lze ji ale pořídit se slevou.

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

Zde se dostáváme k největší výhodě nástroje, kterou je modularita. Použití webových technologií umožňuje velmi snadnou tvorbu doplňků, ty 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 GitHubu, což přináší možnost rozšířit funkcionalitu o další SŘDB.

Importovat diagramy lze pouze z nativního formátu souboru a StarUML 1. Výhodou je, že nástroj ukládá diagramy v podobě JSONu, 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 softwaru, 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.

Grafický editor nabízí všechny základní funkce pro vytváření ER diagramu a celkově působí uživatelské rozhraní přehledně. Ovšem 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, rychlost odezvy při modelování rozsáhlých schémat si raději ani nepředstavuji. 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 daleko 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, jen je škoda, že jej nelze použít i pro jiná SŘDB. Stejně kvalitním editorem disponuje např. Visual Paradigm, který je vhodný pro jiné SŘDB. Přehledné srovnání možností z pohledu grafických editorů nabízí tabulka 3. Rychlost odezvy a přehlednost GUI je hodnocena známkou z intervalu 1–5, stejně jako ve škole.

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 SŘDB, 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 SŘDB | 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 SŘDB | 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 určena pro operační systém Microsoft Windows. S WPF aplikacemi se váže návrhový vzor MVVM, který slouží k oddělení prezentační vrstvy od doménové logiky, včetně přístupu k zdroji dat. Tento přístup 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.

## 4.1 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 agregování dat, pro jednotlivé komponenty, v podobě jaká vyhovuje prezentační vrstvě. Provázaní těchto dat 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, nicméně tato praktika se používá zpravidla pro komponenty jako je tabulka nebo seznam, kdy hlavním úkolem je prezentace dat ze zdroje. Grafický editor je z tohoto pohledu specifickou komponentou, zejména co se týká povahy prováděných operací, provázaní dat s prezentační vrstvou bylo tedy nutné implementovat alternativním způsobem.

### 4.1.1 Provázání editoru s daty

Každá komponenta má, pro správnou implementaci vzoru MVVM, vlastní viewmodel, jenž je vlastněn výhradně touto komponentou. Kromě, výše zmíněného, uchovávaní dat slouží viewmodely také ke komunikaci mezi jednotlivými komponentami.

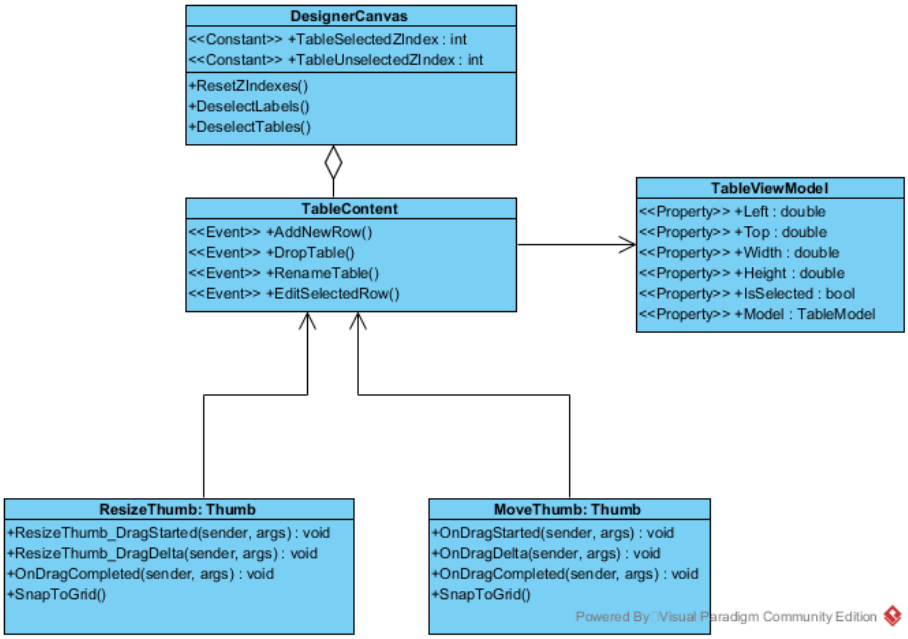
Stejně je tomu i v případě grafického editoru, viemodel obsahuje trojici kolekcí obsahující viewmodely tabulek, vztahů a popisků, tedy objektů, které se vyskytují na plátně. Kolekce po přidání nebo odebraní objektu vyvolají událost, na kterou patřičným způsobem reaguje komponenta editoru. Viewmodel tedy vystavuje rozhraní grafického editoru ostatním částem aplikace, kdy 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 editorem na úrovni viewmodelu, stejným způsobem jako s komponentami využívajícími standardní způsob data-bindingu ve WPF aplikacích.

### 4.1.2 Přesouvaní a změna rozměrů tabulek v diagramu

Implementace obou operací je úzce spjatá s použitím WPF komponenty Thumb, která je nejčastěji využívaná v aplikacích, které pro své fungování využívají výhod dotykového 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. 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. Tímto způsobem je možné provádět úpravy komponenty pro zobrazení tabulky, aniž by došlo k narušení funkcionality přesunu a změny rozměrů.




Obrázek 15: Část třídního diagramu - přesunutí a změna rozměrů tabulky

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 ní komfortně pracovat.

Pro zprostředkování komunikace mezi šablonou a viewmodelem tabulky slouží třída TableContent. Tato třída 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.

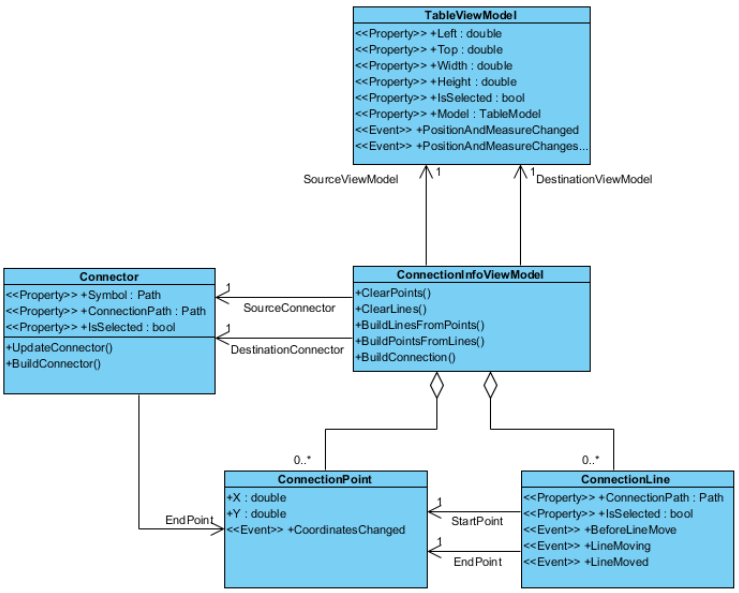
Vztahy mezi zmíněnými třídami zachycuje třídní diagram na obr. 15, ten je značně zjednodušený oproti implementaci, obsahuje jen položky důležité pro tuto kapitolu.

### 4.1.3 Úpravy čar pro vizualizaci vztahu

Práce se samotnými vztahy probíhá opět na úrovní viewmodelu, podobně jako je tomu u tabulek, nicméně samotný vztah nyní není reprezentován jediným objektem na plátně, ale objektů se zde vyskytuje celá řada. Grafický editor používá Oracle CASE notaci, vztah je reprezentován lomenou čárou, na obou koncích této čáry se nachází symboly vyjadřující kardinalitu a povinnost.

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, kde jsou zobrazovaná data aktualizována pomocí jiných prvků uživatelského rozhraní a přímá interakce není nutná.

Viewmodel pro vztah tedy, kromě samotného datového modelu s parametry vztahu, obsahuje kolekci lomových bodů čáry, společně s kolekcí čar, které tvoří jednotlivé segmenty lomené čáry. V první fázi jsou segmenty vytvořeny na základě lomových bodů, které jsou po přidání nového vztahu určeny algoritmem pro vyhledání trasy, detailní popis této činnosti je popsán v kapitole 4.2.1. Editor reaguje na události vyvolané při práci s kolekcí čar tak, aby byla zajištěna integrita dat viewmodelu s objekty na plátně. Závislosti mezi třídami pro zajištění této funkcionality nástroje popisuje třídní diagram 16.




Obrázek 16: Část třídního diagramu - vizualizace vztahu mezi tabulkami

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 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.

## 4.2 Algoritmy pro automatické rozvržení objektů

Vztahy může uživatel libovolně upravovat dle své potřeby, nicméně po přidání nového vztahu mezi tabulkami je nutné tento vztah nejprve vykreslit pomocí lomené čáry na plátno. Tento krok byl v první fázi tvorby nástroje implementován metodou, která nejdříve zjistila vzájemnou polohu obou tabulek a následně vztah vykreslila jako spojnici předefinovaných lomových bodů, jejich souřadnice byly posunuty na základě aktuální polohy tabulek. Tento způsob se neosvědčil v situacích, ve kterých tabulka figuruje v několika vztazích, protože často docházelo, minimálně k částečnému, překrývaní vztahů. 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, což vedlo u rozsáhlých diagramů k naprosté nepřehlednosti. Z toho důvodu je diagram transformován na graf a problém je řešen jako vyhledávání cesty v grafu, který jej umožňuje řešit v krátkém čase se snadnou definicí omezení, která jsou na požadovanou trasu kladeny. Použité algoritmy jsou rozebrány v kapitole 4.2.1.

Přidávání již existujících tabulek v databázi z příslušného panelu je možné dvěma způsoby. Prvním z nich je klasická drag and drop varianta, známá z mnoha dalších nástrojů, která 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 druhá možnost, která po dvojitém stisku levého tlačítka myši umístí tabulku na plátno automaticky. Prvním způsobem implementace bylo umístění tabulky na náhodné místo v diagramu tak, aby se levý horní roh tabulky vyskytoval ve volném místě, nicméně toto řešení není zdaleka ideální, protože tabulky se velmi často překrývají a navíc může dojít k situaci, kdy jen několik málo tabulek je rozprostřeno po celém plátně, což není pro komfortní práci s nástrojem žádoucí. Aktuální implementace využívá grafové reprezentace, stejně jako výpočet trasy. Tabulky se rozmísťují kolem středu plátna bez vzájemného překrývání. Použitá metoda je popsána v kapitole 4.2.2.

### 4.2.1 Výpočet trasy po přidání vztahu

Prvním krokem bylo transformace plátna na graf, ve kterém je cesta vyhledávaná. Graf je reprezentován pomocí mřížky, toto vyjádření nejlépe odpovídá reálné reprezentaci plátna pixely. Prvním krokem bylo převedení každého pixelu plátna na samostatný uzel grafu. Tento uzel

obsahuje souřadnice bodu na plátně, společně s informací zda je bod volný nebo jej překrývá tabulka, dále jsou zde data důležitá pro vyhledávací algoritmus, ale tyto položky budou vysvětleny později.

Výhodou této reprezentace je její přesnost, kdy nedochází k zanedbání žádné informace z plátna. Problémem je ovšem katastrofální dopad na výkon nástroje, protože pro plátno o šířce a výšce 4000 pixelů je nutné alokovat 16 milionů uzlům, které v průměru zabírají více než 1GB operační paměti a jejich alokace trvá přes 15 sekund.

Počet uzlů grafu bylo nutné zredukovat na co nejnižší počet při optimálním zanedbání informací. Z tohoto důvodu je plátno nejprve rozděleno na čtvercové úseky o konstantních rozměrech, pokud tabulka alespoň částečně zasahuje do některého z úseku, je tento úsek označen jako obsazený. Z této mřížky je následně vytvořen graf, který místo 16 milionů uzlů, jako tomu bylo původně, obsahuje uzlů, pro dříve zmíněné rozměry, pouze 1600. Alokace operační paměti v tomto případě trvá jen několik milisekund a zabraná paměť se pohybuje řádově v jednotkách MB.

Nyní bylo možné přistoupit k samotnému vyhledávání trasy. Prvním implementovaným algoritmem je prohledávání grafu do šířky. Tento algoritmus 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, cesta je nalezena. Pokud dojde k vyprázdnění fronty aniž by byl nalezen cíl, tak cesta v grafu neexistuje. Tento způsob hledání trasy je velmi snadno implementovatelný, jeho výkon bohužel není ideální, protože vyhledávání trvá řádově desítky až stovky milisekund pro jedinou trasu, nalezení více tras postupně tedy může trvat několik sekund, což je nežádoucí jev.

Druhým implementovaným algoritmem je A* [8], který se používá k nalezení nejkratší cesty v grafu. Tento algoritmus kombinuje výhody hladového BFS a Djikstrova algoritmu. 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, která je v implementaci použita.

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

Uzly jsou uchovány v prioritní frontě, priorita je určena hodnotou vypočtenou pomocí metody Manhattan. 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 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.

Princip A* algoritmu spočívá v použití dvou množin, první je pro ještě nezpracované uzly, druhá pro již zpracované uzly. Algoritmus v každém kroku vezme z množiny nezpracovaných

uzlů uzel s nejvyšší prioritou, tento prvek tvoří hlavu prioritní fronty, 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.

Moje implementace namísto množiny nezpracovaných uzlů používá, již zmíněnou, prioritní frontu a množina zpracovaných uzlů je nahrazena příznakem přímo v uzlu, který indikuje jeho stav. Tento způsob implementace značně optimalizuje základní algoritmus a výkon je tedy mnohem lepší než při použití množin.

### 4.2.2 Automatické rozvržení tabulek

Implementace algoritmu pro rozvržení tabulek na plátno, aniž by došlo k jejich překrytí, využívá opět reprezentaci plátna pomocí mřížky, stejně jako tomu je u vyhledávání trasy. Převod na graf ovšem není pro správnou funkci vysloveně nutný, celý postup by bylo možné realizovat pomocí matice obsahující jen čísla, z hlediska výkonu, je ale, vzhledem k množství dat, tato skutečnost zanedbatelná.

Po připravení mřížky na základě rozvržení objektu na plátně je nutné provést přípravu dat v mřížce pro samotné vyhledávání. 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.

V další fázi už je možné vyhledat volné obdélníky v takto připravené mřížce. Nyní procházíme mřížku po řádcích, zleva, a v případě, že narazíme na posloupnost čísel v řádku, pro které platí, že všechny z nich jsou rovny nebo vyšší výšce obdélníku, o celkovém počtu rovném alespoň jeho šířce, můžeme index pole prvního prvku z této posloupnosti přidat do kolekce možných počátečních bodů hledaného obdélníku.

Nyní zbývá pouze zvolit jeden z těchto bodů, osobně jsem využil opět metodu Manhattan 1 pro určení metriky daného bodu vzhledem ke středu plátna, bod s hodnotou metriky lepší než je průměrná hodnota je zvolen jako výsledný počáteční bod hledaného obdélníku. Implementace celé metody je k vidění ve výpisu zdrojového kódu 1.

```
public Point? FindFreeRectangle(Rectangle rect)
{
   PreprocessGrid();
   var nodes = GetListOfFreePoints(rect);

   if (!nodes.Any())
   {
      return null;
   }

   var center = new Point(_grid.Width/2, _grid.Height/2);
   var avg = (int)nodes.Select(t => Manhattan(t, center)).Average(t => t);
   var res = nodes.FirstOrDefault(t => Manhattan(t, center) <= avg);

   if (res != null)
      return new Point(res.X, res.Y);

   return null;
}
```

Výpis 1: Nalezení volného obdélníku v mřížce

# 5 Návrh a implementace synchronizace ER diagramů s relačním schématem

Grafický editor nabízí mnoho operací pro úpravu relačního schématu skrze GUI. Pro provedení těchto akcí je nutné nejprve sestrojit validní SQL dotaz, který změnu ve schématu provede. Nástroj aktuálně podporuje dva SŘDB, Oracle Database a Microsoft SQL Server. Navzdory tomu, že SQL je standardizované, se syntaxe některých příkazů liší. Odlišnosti SQL syntaxe jsou nejčastěji způsobeny různorodými vlastnostmi těchto SŘDB.

Z důvodů těchto odlišnosti bylo potřeba navrhnout způsob synchronizace s relačním schématem tak, aby změny provedené v diagramu mohly být promítnuty do relačního schématu pomocí jednotného rozhraní, byť se vnitřní implementace pro oba SŘDB liší. Synchronizace je navrhnuta s ohledem na možné budoucí rozšíření o další SŘDB.

## 5.1 Použité návrhové vzory

Použití návrhových vzorů má dobrý vliv nejen na přehlednost a udržovatelnost kódu, ale také často usnadňuje rozšíření stávající implementace o novou funkcionalitu. V této kapitole je popsána funkce návrhových vzorů, které jsou pro synchronizaci zásadní, první dva z nich jsou využity v datové vrstvě, další jsou použity v doménové logice.

### 5.1.1 Data Mapper

Tento vzor vychází z obecného vzoru Mapper, který slouží pro zprostředkování komunikace mezi, na sobě, nezávislými systémy. Data mapper je často používán v případech, kdy potřebujeme přistupovat k různým datovým zdrojům, které se liší implementaci přístupu. Typ datového zdroje je sice jeden, tedy systémový katalog relační databáze, nicméně pro různé SŘDB se katalogy liší, což představuje ideální situaci pro použití tohoto vzoru. Další výhodou je i nezávislost podoby dat ve zdroji a doménové vrstvě, tímto je tedy možné reprezentovat data pro grafický editor jednotným modelem nezávisle na struktuře systémového katalogu.

V nástroji se obvykle nepřistupuje k mapperům přímo, ale kvůli nutnosti přípravy dat před některými operacemi v závislosti na SŘDB, se kterým aktuálně pracujeme, je přístup zapouzdřen pomocí návrhového vzoru Strategy. Toto provázání je popsáno v kapitole 5.1.3. K systémovému katalogu je přístup implementován pomoci vlastních SQL příkazů, pro tuto činnost tedy nejsou využity externí knihovny z důvodů větší variability při práci s katalogem.

### 5.1.2 Layer Supertype

Layer Supertype slouží k sjednocení společného chování určité skupiny objektů nebo celé vrstvy, výhodou je zamezení duplikace metod se stejnou signaturou. Tento vzor je využít v datové vrstvě na dvou místech. První použitím je definice rozhraní pro třídy zodpovědné za přístup k databázi,

zde jsou definovány základní metody pro otevření nového připojení, jeho uzavření a tvorbu SQL příkazu pro konkrétní mapper.

Druhým použitím je sjednocení operací pro data mappery. Vrstva definuje operace, které musí umět příslušný mapper vykonávat, aby jej bylo možné použít pro komunikaci grafického editoru s SŘDB. Jak již bylo zmíněno, SŘDB se svými funkcemi liší, takže tato vrstva nedeklaruje všechny dostupné operace, ale spíše nezbytný základ pro použití v nástroji. Každý mapper obsahuje navíc ještě operace specifické pro SŘDB, se kterým komunikuje, ale ty už součásti supertypu nejsou.

### 5.1.3 Strategy

Strategie je jedním z kategorie návrhových vzorů chování a slouží k definici skupiny tříd, které obvykle řeší stejný problém, ale jejich implementace se liší. Cílem tohoto vzoru je tuto skupinu tříd zastřešit společným rozhraním a umožnit záměnu těchto tříd v závislosti na situaci, ve které je provedení operace požadováno. Jádrem je třída pro udržování kontextu, která nabízí rozhraní pro nabízené operace. Na základě specifikovaného kontextu je vybraná strategie, která operaci provede. Jednotlivé strategie nabízejí jednotné rozhraní, což umožňuje v třídě pro kontext pracovat se strategiemi na úrovni supertypu. Tímto způsobem se vyhneme rozhodovacímu bloku v situacích, kdy je implementace závislá na SŘDB.

V nástroji představuje strategie primární rozhraní přístupu k data mapperům pro provedení změn v relačním schématu. Každému mapperu odpovídá samostatná strategie, kontext je definován typem aktuální relace. 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 mappery pracovat přímo.

### 5.1.4 Singleton

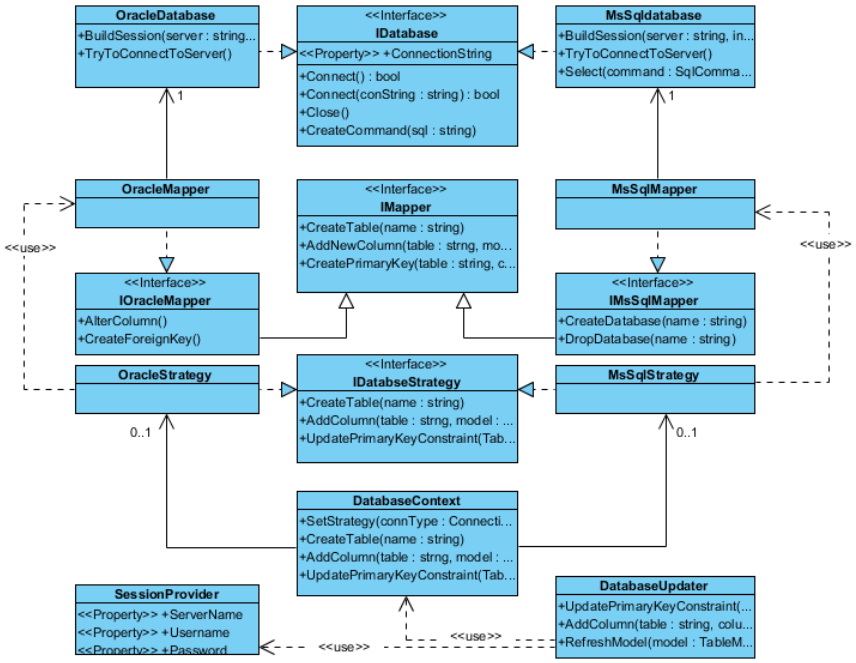
Návrhový vzor singleton je vhodné použít v situaci, kdy potřebujeme ke stejné instanci objektu přistupovat z více míst aplikace. V nástroji je tento vzor využíván pro uchovávání a zprostředkování informací o aktuální relaci.

Uchovávané informace se částečně liší pro různé SŘDB, proto je zde uveden, kromě údajů potřebných k úspěšné autentifikaci, také SŘDB, 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 SŘDB, stejně tak se liší i třídy pro jeho sestavení z uvedených informací. Standardně jsou přihlašovacími údaje zadávány pomocí GUI, nicméně může dojít k situaci, kdy potřebujeme využít parametry, které GUI nenabízí. V tomto případě je možné zadat řetězec vlastní a singleton s informacemi o relaci zprostředkovává tento řetězec pro připojení, aniž by jej bylo nutné sestavovat, jako je tomu v předchozím případě.

## 5.2 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. Strukturu datové vrstvy nalezneme v diagramu 17.

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 Management studiu. Příkladem takové situace je umožnění ukládání NULL hodnot do určitém atributu v diagramu č. 1, změnu nepotvrdíme a v diagramu č.2 nastavíme tentýž atribut primárním klíčem tabulky, nyní změnu potvrdíme ihned, změnu v diagramu č. 1 teď už není možné provést, protože ji brání nově vzniklé integritní omezení.




Obrázek 17: Část třídního diagramu - struktura datové vrstvy

Celý proces aktualizace schématu začíná provedením změny v diagramu pomocí grafického editoru. Po potvrzení změny uživatelem je nutné provést adekvátní SQL příkaz. Na základě SŘDB, se kterým aktuálně pracujeme, je zvolen data mapper, který příkaz provede. Operace mapperu je volána prostřednictvím rozhraní kontextu, který vybere konkrétní strategii pro práci s příslušným mapperem. Tento proces je popsán v kapitole 5.1.3. Toto uspořádání tříd pro

provádění změn v relačním schématu eliminuje nutnost rozhodovacích bloků při každém volání metod data mapperu, stejně tak nabízí snadný způsob rozšíření nástroje o nové SŘDB. Stačí pouze přidat nový data mapper, který implementuje požadované rozhraní a vytvořit pro něj novou strategii. 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 kontextu přes tzv. fasádu, která má na starosti zachycení těchto 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 není změna úspěšná.

## 5.3 Serializace a export 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, diagramy je možné zálohovat v rámci záloh databáze.

Serializace diagramů probíhá na úrovní viewmodelů, protože obsahují veškerá potřebná data týkající se struktury diagramu. Serializaci podléhají veškeré atributy diagramu, které nelze získat z existujícího schématu. Jedná se tedy zpravidla 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.

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.

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.

# 6 Závěr

Cílem této práce bylo vytvořit aplikaci umožňující návrh schématu relační databáze formou ER diagramu, která je schopna pracovat s SŘDB 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 SŘDB je navržena s ohledem na možná budoucí rozšíření o podporu dalších SŘDB. 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 SŘDB, 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/`

# A Struktura přiloženého optického média

Tabulka 5: 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 Instalace programu

Popis instalace.