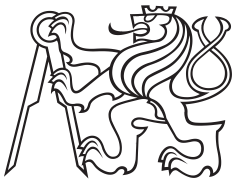
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Automatické vyhodnocování úloh v předmětu Databázové systémy

Adam Škarda

Školitel: RNDr. Ingrid Nagyová, Ph.D.
Květen 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Škarda** Jméno: **Adam** Osobní číslo: **498871**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Automatické vyhodnocování úloh v předmětu Databázové systémy

Název bakalářské práce anglicky:

Automatic evaluation of tasks in the subject Database systems

Pokyny pro vypracování:

Cílem práce je analyzovat možnosti automatického vyhodnocování úloh v předmětu Databázové systémy se zaměřením na úlohy modelování databáze pomocí ER diagramů. Účelem analýzy je specifikovat parametry úlohy a komponenty ER diagramů, které lze počítačově rozeznat a zpracovat. Navržený systém pro automatické zpracování a evaluaci ER diagramů bude implementován do systému BRUTE a výsledky práce budou prakticky ověřeny.
Požadavky projektu:
1. Specifikujte komponenty používané při modelování databází. Zaměřte se na komponenty, z nichž se sestavují ER diagramy.
2. Analyzujte nástroje pro tvorbu ER diagramů se zaměřením na možnosti automatické identifikace jeho komponent a vyberte vhodný nástroj pro výuku v předmětu Databázové systémy.
3. Navrhněte systém automatického zpracování ER diagramů a systém implementujte a otestujte.
4. Systém bude využit pro automatickou evaluaci v BRUTE. Výsledky automatického zpracování ER diagramů vytvořených studenty analyzujte a zhodnoťte přínos nástroje.

Seznam doporučené literatury:

Svoboda, Martin. 2021. „Přednášky z předmětu Databázové systémy.“ Dostupné 31. led. 2023 https://www.ksi.mff.cuni.cz/~svoboda/courses/182-B0B36DBS/
Pokorný, Jaroslav. 2020. „Databázové systémy.“ Praha: Nakladatelství ČVUT.
Stejskal, Petr. 2021. „Modelovací nástroj pro ER konceptuální návrh databází.“ Bakalářská práce. Praha, ČVUT FEL.
Chen, Peter Pin-Shan. 1979. „The entity-relationship model—toward a unified view of data.“ ACM Transactions on Database Systems. 1, č. 1. s. 9–36.
Song, Il-Yeol, Mary Evans a E K. Park. 1995. „Comparative Analysis of EntityRelationship Diagrams.“ Journal of Computer and Software Engineering. 3, č..4. s. 427-459

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ingrid Nagyová, Ph.D. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2023** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **22.09.2024**

RNDr. Ingrid Nagyová, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

V první řadě chci poděkovat své vedoucí práce Ingrid Nagyové za její čas a pomoc při psaní této bakalářské práce. Dále chci poděkovat svým blízkým za jejich podporu během studia a pomoc při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pouze s použitím literatury, kterou deklaruji v kapitole Literatura.
V Praze dne 23. 05. 2023

Abstrakt

Tato práce se zabývá problematikou automatického opravování úkolů z předmětu Databázové systémy, tedy vyhodnocování správnosti entitně vztahového diagramu (Entity relationship diagram, ERD) a jeho následný převod na relační model.

Práce obsahuje shrnutí teorie o ERD a relačních modelech (RM), porovnání nástrojů pro tvorbu diagramů a návrh s popisem implementace řešení pro problém automatizovaného opravování úkolů z předmětu Databázové systémy.

Z práce vyplynulo řešení v podobě aplikace, která dokáže přečíst ERD, najít v něm chyby podle specifikovaných požadavků a následně diagram převést do relačního schématu.

Klíčová slova: Entitně vztahový diagram, XML, java, relační model, databázové systémy

Školitel: RNDr. Ingrid Nagyová, Ph.D.

Abstract

The topic of this bachelor thesis is the automatic correction of assignments from subject Database systems (DBS). Specifically, the evaluation of a Entity relationship diagram (ERD) and its mapping to a relational model (RM).

This paper is comprised of a summary of basic ERD and relational model theory, a comparison of ERD modeling tools and a description of the design and implementation of a solution for the automatic evaluation and correction of assignments from subject DBS.

The result of this project is an application that can correct and evaluate ERD based on predefined requirements and finally map it to a relational model.

Keywords: Entity relationship diagram, XML, java, relational model, database systems

Title translation: Automatic evaluation of assignments in the database systems subject

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V rámci předmětu Databázové systémy (dále DBS) mají studenti za úkol vytvořit entitně vztahový diagram (dále ERD). Tento úkol je opraven a ohodnocen učiteli předmětu. Ti musí projít desítky takových úkolů každý rok a u každého hledat stejnou sadu nedostatků.

Pro hledání těchto nedostatků se nabízí automatizace. Ta by snížila čas strávený opravováním úkolů a redukovala risk přehlédnutí některé chyby či nedostatku.

1.1 Cíl práce

Cílem této práce je analyzovat možnosti automatického vyhodnocování úloh v předmětu Databázové systémy [10] se zaměřením na tvorbu entitně vztahových diagramů. Účelem analýzy je specifikovat parametry této úlohy, které lze počítačově rozeznat, zpracovat a vyhodnotit. Systém navržený na základě této analýzy bude implementován do systému pro kontrolu úloh BRUTE a výsledky práce budou prakticky ověřeny.

1.2 Tvorba ERD - zadání úlohy

Prvním úkolem v předmětu je tvorba ERD, ze kterého vycházejí další úlohy v průběhu výuky. Úkol má řadu kritérií a požadavků pro své hodnocení. [12]

V rámci této práce chceme navrhnout nástroj, který by tyto požadavky dokázal kontrolovat a hodnotit i v případě jejich úpravy či změny v budoucích letech. Kritéria pro hodnocení úkolu v rámci DBS jsou:

- Vytvoření daného množství [1] entit a vztahů (5-8 entity a alespoň 5 vztahů)
- Použití všech druhů atributů (jednoduché, vícehodnotové, strukturované)
- Použití všech druhů identifikátorů (jednoduché, složené, několik identifikátorů)

[1] Počet použití jednotlivých prvků diagramu se může v průběhu let měnit.

- Použití všech druhů vztahů a kardinalit
- Použití alespoň jednoho rekurzivního vztahu
- Použití alespoň jednoho několikanásobného vztahu
- Použití ISA dědičnosti s omezeními (celková/částečná, exkluzivní /překrývající se)
- Použití alespoň jedné slabé entity

Kapitola 2
Konceptuální schéma

V této kapitole uvedeme definici konceptuálního schématu. Dále se budeme soustředit na entitně vztahový model jeho druhy a datovou reprezentaci.

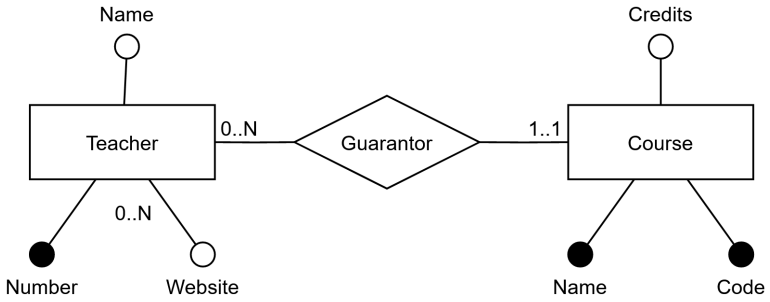
2.1 Definice

Konceptuální schéma je první fáze datového modelování informačního systému. Jedná se o věrnou reprezentaci pro nás důležitých informací a vztahů mezi nimi s určitou mírou abstrakce.

Systém konceptuálních objektů tak popisuje tu část dynamiky reálného světa, která vyplývá z podstaty objektů (konkrétně jejich životních cyklů) a jejich vztahů. Avšak již nemodeluje tu část dynamiky reálného světa, která vyplývá z věcné podstaty potřeby informací - z podstaty věcných (tzv. "podnikových") procesů. [9]

2.2 Entitně vztahový model

Základním nástrojem pro modelování konceptuálního schématu je Chenův entitně vztahový diagram [5] (viz obr. 2.1). ERD slouží k vizuální reprezentaci informací a vztahů mezi nimi [1] a je s ním spojena řada pravidel a návodů jak jej tvořit, které považujeme za techniky a principy datového modelování. [9]



Obrázek 2.1: Entitně vztahový diagram

2.2.1 Stavební bloky ER diagramu

Každý ER diagram tvoří řada prvků. Základními prvky diagramu jsou:

- Entita (Entity, obdélník)
- Vztah (Relationship, kosočtverec)
- Atribut (Atribute, elipsa)
- Spoj (Connection, čára spojující ostatní prvky)

Pro úplný diagram je také vhodné a v některých případech nutné (např. identifikátor) použít další prvky odvozené z výše zmíněných základních. Jsou to:

- Identifikátor (Identifier)
- Složený identifikátor (Composed identifier)
- Strukturovaný atribut (Structured atribute)
- Vícehodnotový atribut (Multivalued atribute)
- Slabá entita (Weak entity)
- Isa Hierarchie (Isa hierachy)
- Rekurzivní vztah (Recursive relationship)
- Kardinalita na spojích (Cardinality)

2.2.2 Validace ERD

Existují pravidla a doporučení pro modelování ERD.[5] Naším cílem je vyhodnotit, zda-li tato pravidla byla při tvorbě diagramu dodržena, a kde byla případně porušena. Při modelování existují dva druhy chyb: syntaktické a sémantické.

Syntaktická chyba se týká nesprávného použití prvků ER. Příkladem jsou:

- Oddělené komponenty diagramu
- Chybějící názvy entit a vztahů
- Chybějící klíčové atributy u entit, které nejsou slabé
- Slabá entita není napojena na entitu s vlastním klíčem, chybí atribut k identifikaci v rámci vztahu
- Chybějící kardinality
- Spoje, které nikam nevedou
- Spoj mezi dvěma entitami

- Spoj mezi dvěma vztahy

Tyto chyby se dají automaticky opravit, neboť nezávisí na chápání modelovaného světa.

Sémantickou chybou se myslí nesprávné převedení informací do ER. Například:

- Zavádění nepotřebných entit
- Nesprávné určení kardinality
- Záměna atributu za entitu (a naopak)
- Nesmyslné vazby mezi entitami
- Nadměrná míra detailu
- Nedostatek detailu

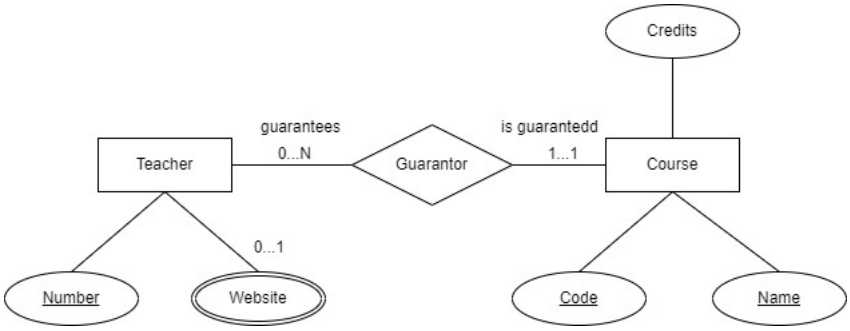
Tyto chyby se pochopitelně nedají kontrolovat automaticky, neboť závisí na modelovaném tématu a subjektivním pohledu na věc.

V rámci předmětu DBS existuje ještě jeden druh chyby a to dodržení zadání úkolu, specifikované v kapitole 1.2. Zde nestačí zkontrolovat jestli jsou kritéria zadání dodržena, je také nutné zjistit do jaké míry a na základě toho je bodově ohodnotit.

2.2.3 Druhy ERD diagramu

Existuje řada notací pomocí, kterých je možné ER diagram vytvořit. V této sekci se zmíním o několika z nich. [14]

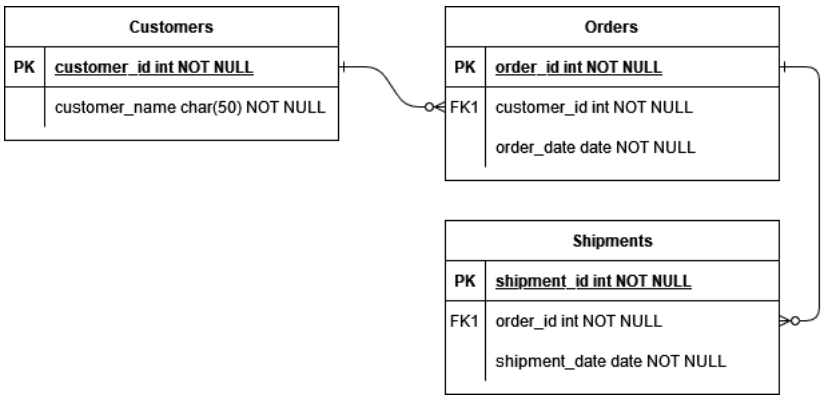
První notace, kterou vytvořil a popsal Peter Pin-shan Chen, se nazývá Chenova notace (viz Obr. 2.2). Je velice rozšířená, dodnes se často používá, avšak oproti jiným je výsledný diagram příliš roztahaný a velké množství prvků modelu vede k nepřehlednosti. Důležité pro nás je, že z této notace vychází i notace, která se používá v předmětu DBS.



Obrázek 2.2: Chenova notace

Mezi další používané notace patří Martinova (viz Obr. 2.3), která používá takzvané vraní nohy (Crow's Foot) pro grafické znázornění kardinalit. Tato

změna vede ke stručnějšímu a jasnějšímu diagramu. V rámci předmětu DBS se tato notace nepoužívá.



Obrázek 2.3: Martinova notace

V poslední řadě se k tvorbě ERD používá i grafický jazyk UML (viz 2.4). Ten má výhodu ve srozumitelnosti díky své standardizaci, avšak modelování popsaných vztahů mezi entitami je v něm o něco méně srozumitelné.

Obrázek 2.4: UML notace

2.2.4 Reprezentace ERD

Vzhledem k tomu, že se ERD používá hlavně pro znázornění modelovaných informací pro člověka, je nejčastěji reprezentován obrázkem (např. Obr. 2.1).

To však vede k obtížné strojové práci s modelovanými daty. Potřebujeme tedy další reprezentace diagramu, abychom ho mohli řádně analyzovat, zpracovat a vyhodnotit jeho správnost.

Nástroje, určené k tvorbě ERD, využívají pro reprezentaci, ukládání a export diagramu řadu formátů, kromě už výše zmíněného obrázku, který je výstupem každého nástroje. Bohužel tyto formáty nejsou nijak standardizované a liší se napříč nástroji. Jsou to:

- XML/HTML dokument
- VSDX - zkomprimovaný XML pro Microsoft aplikace
- JSON
- Grafika - SVG

- Binární soubor (vpd)
- Tabulka (slsx, csv)

Z těchto možností jsou nejpřívětivější formáty XML, JSON a tabulky, neboť je jejich zpracování nejjednodušší - jsou strukturované a data jsou lehce přístupná. XML má dále tu výhodu, že existují doporučené standardy pro ukládání ERD do XML schématu [6].

Dále by se dalo uvažovat nad zpracováním SVG (je to technicky XML). Naopak s binární reprezentací diagramu není možné pracovat, neboť je čitelná téměř výhradně jen nástrojem, který ji vygeneroval.

Nakonec je možné pomocí umělé inteligence zpracovat i obrázek ER diagramu, ale je to zdaleka nejnáročnější možnost.

Kapitola 3
Nástroje pro modelování ER

V této kapitole porovnáme možné aplikace pro tvorbu ER diagramů. Následně z tohoto porovnání vybereme jeden nástroj, se kterým budeme dále pracovat.

3.1 Kritéria výběru

Ve výběru porovnávaných aplikací jsou z velké části jen volně dostupné webové aplikace. Důvodem tohoto výběru je už zmíněná dostupnost pro studenty předmětu DBS, kteří nemohou (a nechtějí) platit za nástroj pro jeden předmět. Těchto nástrojů je však stále mnoho a porovnávat je všechny je nad rámec této práce. Vybrali jsme tedy aplikace doporučené předmětem [12], učiteli a dalšími zdroji. [7]

Každý nástroj porovnáváme na základě následujících kritérií:

- Kvalita - jestli je možné nástrojem namodelovat vše potřebné pro úspěšné splnění zadání úkolu v rámci předmětu DBS
- Dostupnost - zda je nástroj volně dostupný
- Intuitivnost - uživatelská přívětivost pro studenty předmětu DBS
- Formát dat - nabízené možnosti ukládání a exportu diagramu z aplikace. Preferované formáty jsou diskutovány v kapitole 2.2.4 .

3.2 Draw.io

Doporučený nástroj pro modelování konceptuálního schématu v předmětu Databázové systémy. Je robustní, jak v množství různých diagramů, které se dají pomocí něho vytvořit, tak i v možnostech vizualizace, ukládání a exportu dat (XML, HTML i vsdx). Pro naše účely je téměř ideální, avšak není přímo určený pro tvorbu ERD.[1]

[1] Nástroj Draw.io je dostupný na portálu https://www.diagrams.net/

3.3 ERdia

ERdia[11] je jednoduchá, volně dostupná webová aplikace výhradně pro tvorbu Chen ER diagramů. Byla vytvořena v rámci bakalářské práce na ČVUT. Je používána učiteli při výuce předmětu DBS. Pro účely předmětu je výborná, ač mnohdy neintuitivní. Ukládá data do přívětivého XML dokumentu. Její největší výhodou je, že se v ní student může dopustit menšího počtu chyb, protože mu nástroj některé chyby nedovolí vytvořit. Například nelze použít vícehodnotový atribut jakožto klíč. [2]

3.4 ERDplus

ERDplus je nástroj, který je téměř identický výše zmíněné aplikaci ERdia, je však srozumitelnější. Jeho nevýhodou je, že data diagramu lze ukládat do formátu JSON (preferovaný formát dat, viz kapitola 2.2.4) pouze pokud se uživatel nepřihlásí. Přihlášený nemá možnost exportovat diagram do přívětivého datového formátu. [3]

3.5 Visio

Placený nástroj pro tvorbu diagramů od Microsoft. Je avšak součástí balíčku Microsoft Office 365 a je tak dostupný pro studenty ČVUT zdarma v rámci školní licence. Data diagramu jsou ve formátu vsdx, což je zkomprimovaná sada XML dokumentů. Pro tvorbu ER diagramů je však Visio příliš málo uzpůsobeno a modelování je komplikované a neintuitivní (kardinality vazeb, druhy šipek atd.). [4]

3.6 Creately

Creately je nástroj doporučený na stránkách předmětu DBS. Je dobrý pro tvorbu různých diagramů, avšak pro účely úkolu není ideální, neboť nabízí pouze omezenou podporu pro styl ER diagramu, který je požadovaný v zadání. Dále je nástroj těžko srozumitelný a některé funkce jsou přístupné jen pro prémiové uživatele. Data diagramu je možné exportovat ve formátu csv tabulky. [5]

[2] Nástroj ERdia je dostupný na portálu https://erdia.stejspet.cz/

[3] Nástroj ERDplus je dostupný na portálu https://erdplus.com/

[4] Informace o nástroji Visio jsou dostupné na stránkách https://www.microsoft.com/cs-cz/microsoft-365/visio/visio-in-microsoft-365

[5] Nástroj Creately je dostupný na adrese https://creately.com/

3.7 Shrnutí

Účelem této kapitoly bylo vybrat nejvhodnější nástroj pro modelování ER diagramu pro úkol předmětu DBS.

Důvod, proč nástroj vůbec vybíráme je dvojí. Jednak chceme doporučit program vhodný pro studenty a ulehčit jim hledání nejlepší možné aplikace. Hlavně chceme, aby výstup úkolu byl u každého studenta standardizovaný a v přívětivém formátu.

	Kvalita	Dostupnost	Intuitivnost	Formát dat
Draw.io	✓	✓	✓	✓
ERdia	✓	✓	✗	✓
ERDplus	✓	✓	✓	✗
Visio	✗	✓	✗	✓
Creately	✓	✓	✗	✗

Tabulka 3.1: Srovnání ER nástrojů

Z tabulky 3.1 vybraných nástrojů se nám nabízí Draw.io a ERdia jako nejlepší kandidáti.

Abychom však mohli správně zpracovat výstup Draw.io museli bychom definovat striktní postup modelování a tento postup by studenti museli dodržovat. Důvodem je rozmanitost programu a svoboda modelování, kterou nabízí.

Nejlepší názor pro naše účely je tedy nástroj ERdia. Už se v rámci předmětu používá, umí modelovat vše co je potřebné, je volně a jednoduše dostupný a výstupem modelování je přívětivý datový formát XML.

Kapitola 4
Relační model

V procesu získávání požadavků pro tuto práci, byl vznesen návrh na přetvoření opraveného ERD do odpovídajícího relační modelu (dále RM). Tento návrh vychází ze skutečnosti, že druhým úkolem pro studenty v předmětu DBS je právě převod vlastního ERD do relačního modelu.

V této kapitole popíšeme co je to RM a jak se dá vytvořit převodem z ERD.

4.1 Definice

Relační model je způsob ukládání dat založený na predikátové logice a matematických relačních množinách. S touto představou o struktuře databází přišel poprvé informatik E. F. Codd v roce 1969. [2]

Relační schéma databáze je sada tabulek a omezení. Každá tabulka je pojmenovaná sada sloupců (atributů), kde každý sloupec, nebo sada sloupců, která odpovídá minimálnímu klíči, je podtržená. [13]

Tabulka v kontextu relačního schématu vypadá následovně:

`Tabulka(klíč1, attribut1, attribut2, ...)`

Tabulky však často nestojí samy o sobě a odkazují se na další tabulky. Například vztahy mezi entitami v ERD jsou často reprezentovány tabulkou, která používá klíče těchto entit jako vlastní sloupce. Těmto sloupcům, nebo sadám sloupců říkáme *cizí klíče*.

Cizí klíče se v relačním schématu zapisují pod tabulku, ve které jsou použity následujícím způsobem:

`Vztah(cizí klíč 1, cizí klíč 2)`
`FK:(cizí klíč 1)` $\subseteq$ `entita 1(klíč)`
`FK:(cizí klíč 2)` $\subseteq$ `entita 2(klíč)`

4.2 Převod z ERD

Existuje řada pravidel pro mapování ERD do relačního modelu, například Kolp a Zimanyi navrhují způsob s minimální redundancí informace[4] a Pieris

et al. používají způsob pro maximální zachování informací[3].

V této práci však využijeme způsob mapování ERD do relačního schématu z předmětu DBS.[12] Základní pravidla jsou následující:

- Každá entita je vlastní tabulka a její atributy jsou sloupce v této tabulce. Klíč entity je podtržen.
- Vícehodnotové atributy jsou vlastní tabulky, které se odkazují na entitu, či vztah, ke kterému patří cizím klíčem. Mají také sloupec, který drží původní hodnotu atributu.
- Vztahy, které spojují více než dvě entity, jsou rovněž vlastní tabulky. Atributy asociované s tímto vztahem se stávají sloupci a entity, ke kterým se vztah vázal, se stávají cizími klíči.
- Vztahy, které spojují právě dvě entity, se mohou obecně zapsat jako samotná tabulka s cizími klíči. Existují však další speciální pravidla pro jejich tvorbu.

  Například má-li tento vztah kardinalitu k jedné entitě 1..1 a k té druhé 0..N či 1..N, poté se dá tento vztah zredukovat do entity, s kterou má vazbu 1..1.
- Slabá entita je entita, která se váže přes vztah pomocí cizího klíče na jinou entitu. V tomto případě vztah mezi těmito entitami se nestává vlastní tabulkou.
- V případě ISA hierarchie se každá entita stane standardně vlastní tabulkou. Dále se vazba dítěte na předka zapíše jako cizí klíč v tabulce dítěte.

Kapitola 5
Návrh řešení

V této kapitole shrneme požadavky na aplikaci, která bude automaticky vyhodnocovat úlohy z předmětu DBS. Dále navrhneme procesy a datové struktury, které budou požadavky splňovat. Nakonec projdeme technologie, které použijeme k implementaci řešení.

5.1 Požadavky

Požadavky na aplikaci vycházejí ze zadání této práce a konverzace s vyučujícím předmětu DBS.

5.1.1 Popis uživatelů

- **Učitel** - Učitel předmětu DBS, který opravuje jednotlivé úlohy. Hlavní uživatel aplikace.

5.1.2 Business cíle

- **BG01 - Tvorba aplikace pro automatické vyhodnocování "Tvorba ERD" z předmětu DBS**
  - Cílem projektu je tvorba aplikace, která dokáže automaticky najít chyby v řešení úlohy "Tvorba ERD" předmětu DBS a na základě nich je ohodnotit.

5.1.3 Business požadavky

- **BRQ101** - Hodnocení ERD
  - Jako učitel chci, aby aplikace dokázala najít chyby v ERD a na základě nich vytvořila hodnocení řešení úlohy.
- **BRQ102** - Nastavení kritérií hodnocení
  - Jako učitel chci, aby mi aplikace umožnila měnit kritéria hodnocení jednotlivých úloh.

5.1.4 Funkční požadavky

- **FRQ101** - Načtení ERD
  - Aplikace umožní uživateli načíst ERD ve formátu XML.
- **FRQ102** - Extrahování dat ERD z XML
  - Aplikace umožní uživateli extrahovat informace o ERD ze souboru XML bez ztráty dat.
- **FRQ103** - Vyhledání chyb v ERD
  - Aplikace umožní uživateli vyhledat chyby v ERD.
- **FRQ104** - Vyhodnocení ERD na základě přítomných chyb
  - Aplikace umožní uživateli vytvořit hodnocení odevzdaného ERD na základě přítomných chyb.
- **FRQ201** - Nastavení bodového ohodnocení
  - Aplikace umožní uživateli bodově ohodnotit správnost ERD na základě přítomných chyb a předem daných kritérií.
- **FRQ202** - Specifikace chyb
  - Aplikace umožní uživateli specifikovat a upřesnit kritéria jednotlivých chyb.

5.1.5 Kvalitativní požadavky

- **NRQ01** - Integrace do systému BRUTE
  - Aplikace bude fungovat v rámci systému BRUTE pro automatické zpracování odevzdaných úkolů.
- **NRQ02** - Čitelný výstup na standardní výstup
  - Výstup aplikace bude zapisován v čitelném a srozumitelném formátu na standardní výstup.
- **NRQ03** - Výstup ve formátu HTML
  - Aplikace umožní formátovat svůj výstup do HTML pro lepší čitelnost.

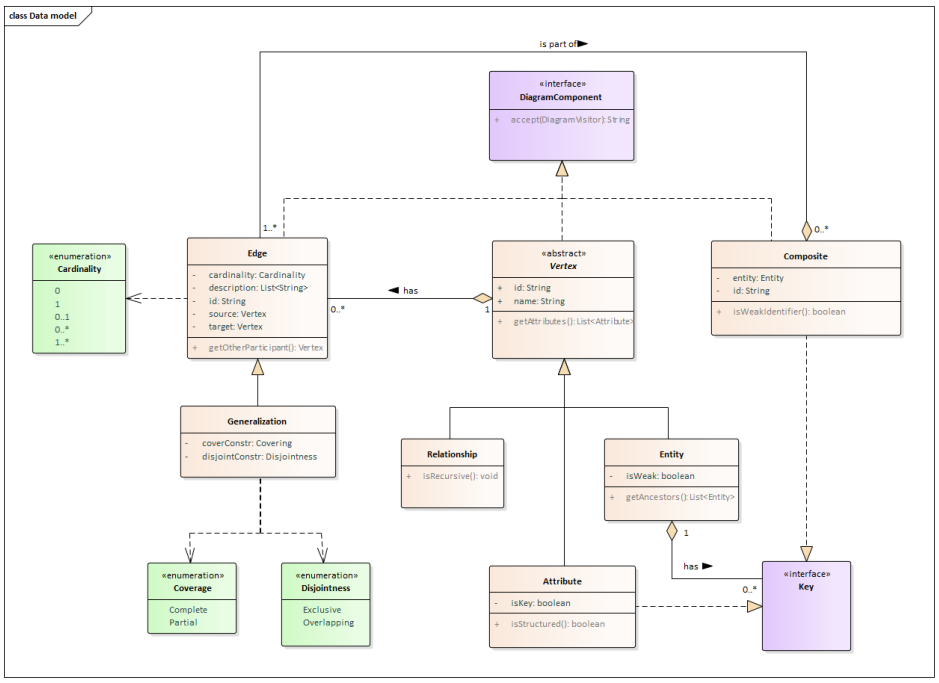
5.2 Data

V rámci aplikace je nutné mít připravené datové struktury, které budou uchovávat informace o ERD a o chybách, které se v něm vyskytují. V této části předvedeme a popíšeme tyto struktury.

5.2.1 Datový model ERD

Diagram tříd na obrázku 5.1 popisuje strukturu pro ukládání dat o ERD bez ztráty informace. Vychází z meta-modelu ERD. Tento návrh už předpokládá, že implementace řešení bude provedena objektově orientovaným způsobem.

Celá struktura je dále uchovávána ve třídě `Diagram`, která uchovává kolekce každého prvku pro ulehčení práce a přístupu k datům ERD.



Obrázek 5.1: Diagram tříd ERD

5.2.2 Datový model chyb

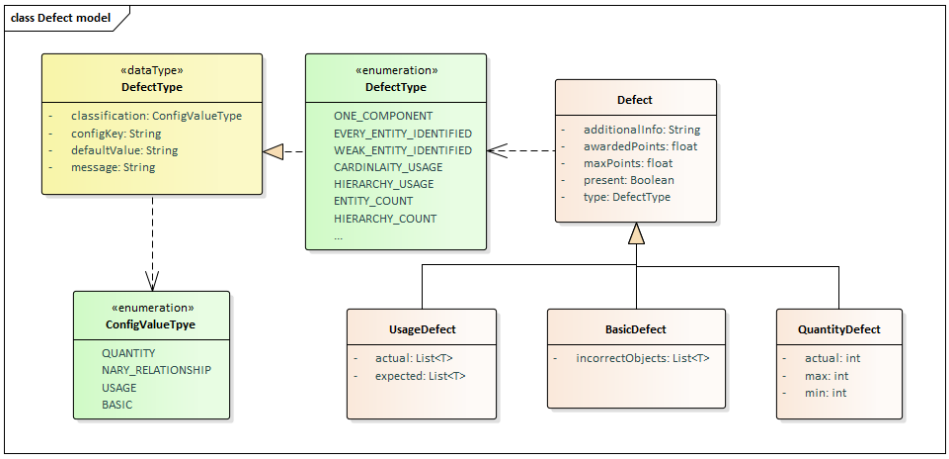
Na obrázku 5.2 je diagram tříd, který popisuje způsob ukládání chyb, které se nacházejí v ERD.

Třída `Defect` reprezentuje chybu nacházející se v ERD. Tyto chyby se dělí do tří kategorií:

- `UsageDefect` slouží pro reprezentaci chyb, které popisují využití některých prvků, které jsou povinné v rámci úlohy. Příkladem je využití všech kardinalit v diagramu.
- `QuantityDefect` slouží k reprezentaci chyb, které popisují počet využití některých prvků, například počet entit přítomných v diagramu.
- `BasicDefect` znázorňuje základní modelovací chybu (viz syntaktické chyby v 2.2.2) a na kterých objektech se tato chyba nachází.

Enumerační třída `DefectType` je založena na enumeracích v programovacím jazyku Java, kde jednotlivé položky enumerace mohou mít vlastní atributy.

Tyto atributy jsou znázorněné ve žluté `<dataType>` třídě. Celá enumerace slouží jako seznam všech možných chyb, které se mohou v ERD nacházet.



Obrázek 5.2: Chyby - Diagram tříd

5.3 Procesy

V této části se věnujeme tomu, jak vypadá práce s aplikací a jak jsou rozděleny a definovány procesy, které vedou od spuštění až k výstupu.

5.3.1 Celkový proces

Základním procesem v rámci aplikace je jeho celkový průběh, jak je znázorněno na diagramu 5.3.

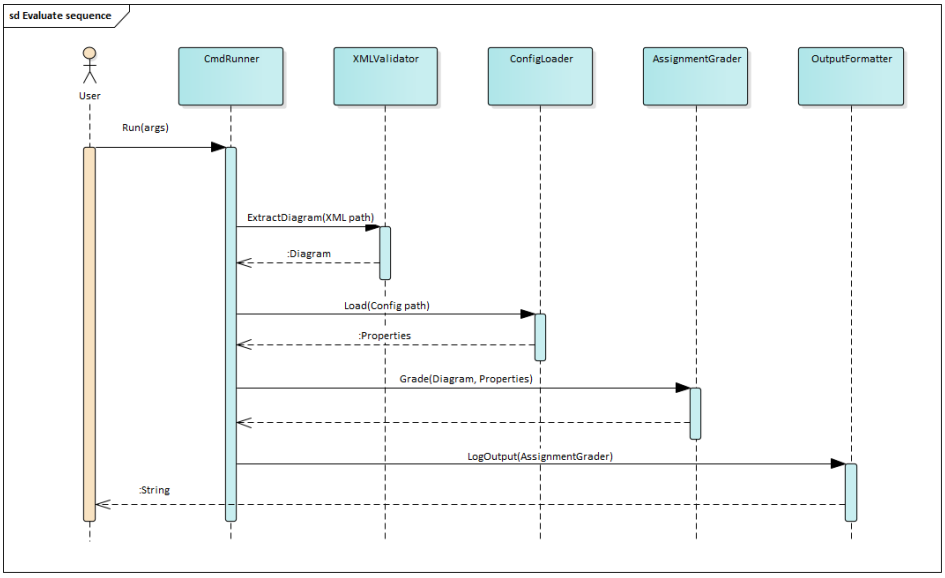
Nejdříve uživatel spustí program s potřebnými argumenty přes příkazovou řádku. Argumenty jsou poté zpracovány pomocí třídy `CmdRunner`, která dále volá všechny další podprocesy.

Jako první zavolá třídu `XMLValidator`, která z uvedeného XML souboru vyextrahuje potřebný ERD v podobě třídy `Diagram` (viz 5.2.1), pokud je to možné. Podrobněji je tento proces popsán v sekci 5.3.2.

Dále se najde a načte konfigurace (`Properties`) pro hledání chyb pomocí třídy `ConfigLoader`. Pokud není cesta ke konfiguračnímu souboru specifikována, nebo konfigurační soubor nelze načíst, použije se výchozí konfigurace, která je definována přímo v programu.

V dalším kroku se pomocí parametrů `Properties` prohledá `Diagram` pomocí třídy `AssignmentGrader`. Cílem této operace je najít všechny přítomné chyby a na základě nich vytvořit bodové ohodnocení. Podrobněji se tomuto procesu věnujeme v sekci 5.3.3.

Nakonec se výstup `AssignmentGrader` převede do znakového řetězce, který se v požadované formě předá uživateli. Více o tomto procesu se dozvíte v sekci 5.3.4.



Obrázek 5.3: Celkový proces

5.3.2 Parsing

Proces parsování ERD ze souboru probíhá ve dvou krocích a je znázorněn na obrázku 5.4.

Nejprve se vytvoří dokumentová struktura z XML, nad kterým proces probíhá, za pomocí DOM parseru ve třídě `DocumentBuilder`. Tento dokument je stromová struktura, která nám umožňuje procházet a vybírat relevantní informace z XML.

Následně tento strom XML uzlů procházíme v třídě `Parser`. Ta vytvoří třídu `Diagram`, do které postupně vkládá všechny entity, vztahy, atributy a hrany mezi nimi. Tyto jednotlivé prvky diagramu `Parser` identifikuje pomocí tokenů v atributech XML uzlů. Posledním úkonem v rámci tohoto kroku je proces identifikace klíčů a slabých entit v `Diagramu`.

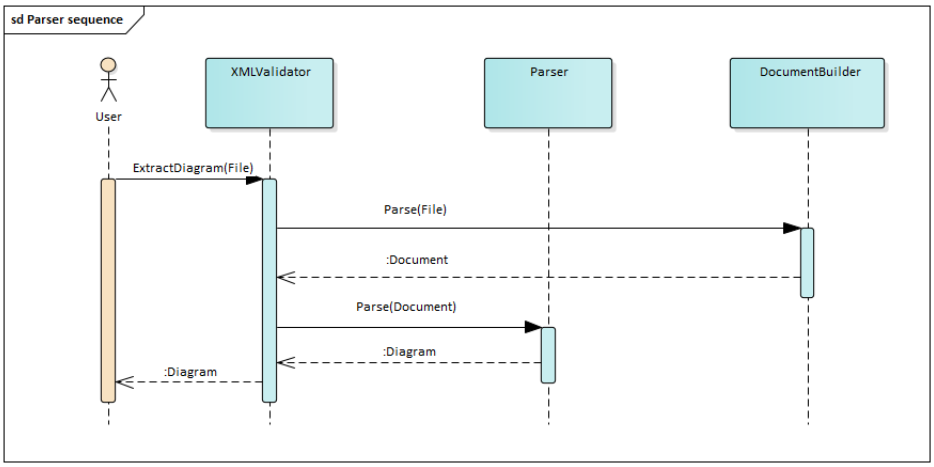
5.3.3 Detekce chyb

Stěžejní částí aplikace je proces detekce chyb v ERD, který probíhá v rámci třídy `AssignmentGrader`. K jeho nejlepšímu pochopení slouží diagram datového modelu chyby 5.2 a obrázek 5.5, který popisuje jeho průběh.

Detekce chyb probíhá ve třídě `DefectChecker`. Ta u každé možné chyby, která by se mohla v ERD vyskytovat, nejdříve extrahuje její konfiguraci z `ConfigExtractor`.

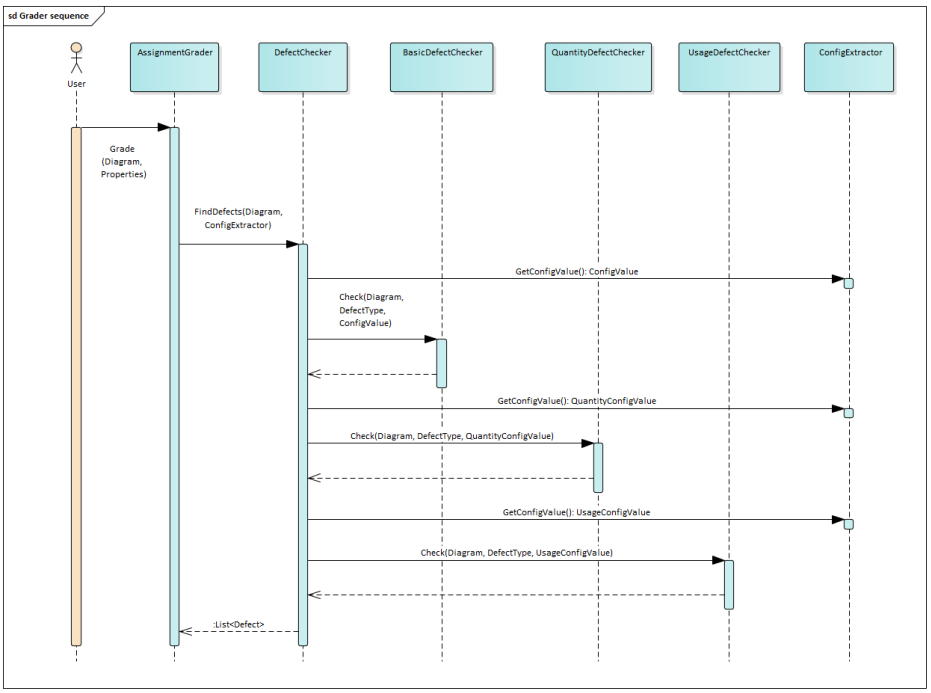
Poté zavolá funkci na detekci této chyby, jejíž implementace se nachází v jedné ze tříd `BasicDefectChecker`, `QuantityDefectChecker` a `UsageDefectChecker` v závislosti na tom jakého druhu konkrétní chyba je (viz obr. 5.2).Při zavolání této funkce použije jako parametr extrahovanou konfiguraci.

Nakonec, když `DefectChecker` projde celý seznam všech možných chyb, vrátí třídě `AssignmentGrader` seznam všech zkontrolovaných `Defectů`. Na



Obrázek 5.4: Proces parsingu

základě tohoto seznamu vytvoří `AssignmentGrader` hodnocení ERD.



Obrázek 5.5: Proces detekce chyb

5.3.4 Výstup

Výstup programu tvoří tři části, které dohromady informují uživatele o běhu a výsledcích programu.

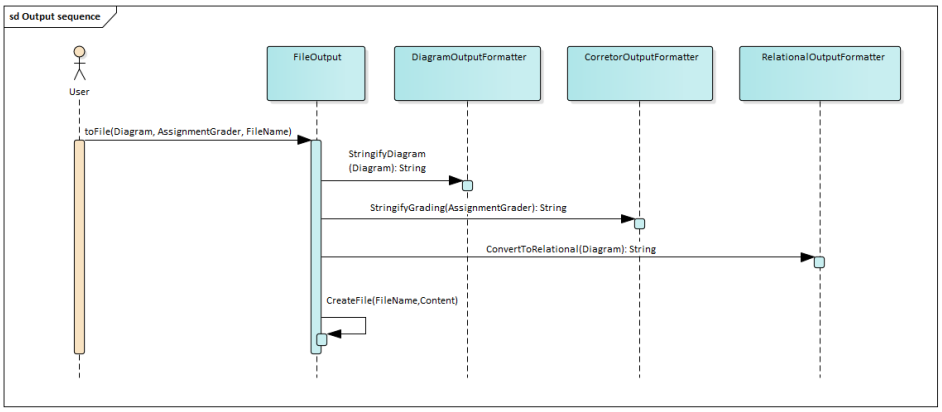
Nejdříve program vytvoří slovní reprezentaci ER diagramu, který opravuje. Tento úkol odvede `DiagramOutputFormatter`, který definuje způsob jakým

se ERD vypisuje do textové formy. Využívá k tomu návrhový vzor visitor, pomocí kterého přidává každé komponentě ERD další `toString` metodu.

Další krok, tedy převedení nalezených chyb do textové formy, zpracovává `CorrectorOutput formatter`, který funguje velice podobně jako `DiagramOutputFormatter` z předchozího kroku, akorát pomocí vzoru visitor přetváří třídy `Defect`.

Poslední z kroků, je převod ERD na RM. Tímto procesem se budeme podrobně zaobírat v kapitole 6.4. Zde stačí vědět, že funguje na podobném principu visitorů jako předcházející kroky.

Nakonec se celkový výstup programu vytiskne na požadované místo, buďto do souboru ve formátu HTML nebo na standardní výstup.



Obrázek 5.6: Proces tvorby výstupu

5.4 Technologie

Tuto část kapitoly věnuji technologiím, využitým k implementaci navrhované aplikace.

5.4.1 Jazyk

K implementaci řešení jsem si zvolil programovací jazyk Java. Vybral jsem ho z několika důvodů, které však nejsou žádným způsobem vitální pro implementaci aplikace.

Hlavním takovým důvodem je nezávislost Javy na prostředí a jeho přítomnost v systému BRUTE, což nám umožní jednoduché nasazení aplikace. Dalším důvodem je má znalost jazyka a jeho možností a limitací.

Verze Javy pro spuštění programu je JDK 8, kvůli její obecné rozšířenosti a dostupnosti v BRUTE.

5.4.2 Parser

Pro čtení ERD z XML dokumentu jsem použil DOM parser. Document Object Model je rozhraní, které umožňuje číst a měnit obsah XML dokumentů.

Jeden z důvodů pro tuto volbu je, že při čtení ERD procházím jednotlivé elementy na přeskáčku a ne tak, jak jsou řazené v původním dokumentu. DOM je navíc rychlejší, i když náročnější na paměť, protože načítá celý dokument najednou, paměť nás však v této aplikaci neomezuje.

Posledním důvodem je jednoduchost implementace oproti SAX, což je parser založený na zpracovávání událostí při procházení XML.

Kapitola 6

Implementace řešení

Tuto kapitolu věnujeme způsobu implementace navržené aplikace. Projdeme jednotlivé části programu, a u každé uvedeme jejich způsob implementace a problémy, s kterými jsme se setkali.

6.1 Parsování

Pro účely extrahování ERD z XML jsme vytvořili rozhraní `Parser`, které má jedinou metodu `Parse`. Rozhraní slouží k jednoduché rozšiřitelnosti aplikace v případě opravování ERD vymodelovaných jiných nástrojích než je ERdia (viz kapitola 3). Cílem je vždy získat z procesu parsování odpovídající `Diagram` bez ohledu na implementaci parseru.

Pro XML soubor vygenerovaný z ERdia je způsob extrakce ERD následující:

1. Na základě obsahu svých atributů a přítomnosti identifikačních *tokenů* se jednotlivé XML elementy roztřídí do dvou kategorií. Jednu kategorii tvoří entity, vztahy a atributy ERD, které společně označujeme jako vrcholy. Druhou kategorií jsou poté ERD spoje. Tuto kategorií označujeme jako hrany.

2. Kategorii vrcholy dále třídíme pomocí *tokenů* ve *style* atributu XML elementu na entity, atributy a vztahy, které vložíme do prázdného `Diagramu`.

3. Kategorii hrany roztřídíme podobným způsobem jako předchozí a přidáme do `Diagramu`. Přidáním hrany do `Diagramu` myslíme jednak přidání spoje do kolekce a jednak spojení a provázání asociovaných vrcholů.

4. Na základě informací z předchozích kroků identifikujeme klíče (jednoduché i složené) a přiřadíme je jednotlivým entitám.

5. V posledním kroku označíme entity, které jsou slabé. Tedy entity se složeným klíčem jehož člen je vztah.

Pokud výše zmíněné kroky proběhnou bez chyby pak máme vytvořený `Diagram`, s kterým budeme pracovat v dalších částech. V případě chyby se chyba vypíše a program skončí.

6.1.1 Problémy

Jedním z problémů, s kterým jsme se setkali při tvorbě parseru pro ERdia i pro Draw.io je heterogenita jejich výstupu. Respektive rozhraní `Parser` a datová struktura `Diagram` nám nijak nezajišťují, že výsledek parsování stejného ERD v různých nástrojích různými `Parsery` bude ten samý `Diagram`.

Tento problém ve výsledku znamená, že bez dalších opatření (např. validace), není program spolehlivě rozšiřitelný na jiné programy než ERdia.

6.2 Hledání chyb

Základní funkcionalitu pro vyhledávání plní dvě třídy `AssignmentGrader` a `DefectChecker`.

`AssignmentGrader` slouží k operacím prováděným nad kolekcemi chyb přítomných v `Diagramu`. Například výpočet celkového počtu bodů, které úloha získala, se odehrává v této třídě.

Kolekce chyb vytváří `DefectChecker` podle vstupní, nebo výchozí konfigurace. V takové kolekci jsou přítomny všechny druhy chyb, které `DefectChecker` vyhledává na daném `Diagramu`.

Detekce jednotlivých chyb se odehrává v unikátních metodách pro každou z nich. Všechny tyto metody mají podobný průběh. Ten se skládá ze tří jednoduchých kroků:

1. Zjištění konfigurační hodnoty pro daný druh chyby
2. Zavolání metody pro detekci chyby
3. Přidáním výsledného `Defectu` do výsledné kolekce chyb

6.2.1 Konfigurace

Zjištění konfigurační hodnoty probíhá v `ConfigExtractoru` a je založené na interakci s `Properties` třídou. Ta je základním způsobem čtení konfigurace v jazyce Java a je vytvořena ze souboru, který byl aplikaci předán při její inicializaci.

Z `Properties` se tedy na základě klíče, specifikovaného v enumerační třídě `DefectType`, vyhledá pro každý druh chyby konfigurační hodnota `ConfigValue`. Ta popisuje omezení a požadavky pro danou chybu.

Například pro chybu “Počet entit v diagramu”, je v konfigurační hodnotě specifikován minimální a maximální počet entit spolu s body, které budou přiděleny, pokud se chyba v diagramu nevyskytuje.

6.2.2 Základní chyby

Základní chyby nebo také chyby typu `BasicDefect` (viz obr. 5.2) patří mezi nejdůležitější a nejtěžší druhy chyb ke kontrole. Jsou to chyby, které vznikají

z nedodržení fundamentálních pravidel pro modelování ERD (viz 2.2.2). V rámci konfigurace je možné pouze změnit počet bodů za jejich absenci.

Jedná se o chyby vycházející z nedodržení následujících pravidel:

- Každá entita, atribut a vztah je pojmenována
- Anotované hierarchie - specifikované pokrytí a překrytí
- Platné kardinality na vícehodnotových atributech (Kardinalita 1..1 je neplatná)
- Vícehodnotový atribut nesmí být použit jako klíč entity
- Klíčový atribut není součástí složeného klíče
- Neexistují duplikátní názvy vztahů a entit
- Neexistují duplikátní jména atributů na stejné entitě, či atributu
- Každá silná entita má klíč nebo složený klíč
- Každou slabou entitu je možné identifikovat
- Každý spoj entita - vztah musí mít specifikovanou kardinalitu
- Celý diagram je v jedné komponentě souvislosti
- Klíčové atributy nejsou umělé (název Id, či identifikátor)
- V diagramu se nesmí nacházet cyklické hierarchie

Detekce těchto chyb probíhá ve třídě `BasicDefectChecker`. Ta u každé chyby projde diagram, najde entity, atributy a vztahy, u kterých byla chyba detekována, a vytvoří výsledný `BasicDefect`, který chybu popisuje.

6.2.3 Chyby v rámci zadání

Chybami v rámci zadání myslíme nedostatky, které vznikly nedodržením zadání úlohy. Jelikož se zadání úlohy může rok od roku měnit. Mohou některé z chyb, které detekujeme, být redundantní, nebo naopak můžou chybět.

Nedostatky ERD plynoucí ze zadání dělíme do dvou skupin `QuantityDefects` a `UsageDefects`. Toto rozdělení zjednodušuje implementaci konfigurace a umožňuje nám využít návrhový vzor *Template* k redukci a zjednodušení detekce chyb.

`QuantityDefects` popisují dodržení následujících množstevních požadavků:

- Počet entit
- Počet strukturovaných atributů
- Počet atributů

- Počet vícehodnotových atributů
- Počet vztahů
- Počet rekurzivních vztahů
- Počet ternárních nebo složitějších vztahů
- Počet slabých entit
- Počet entit s několika identifikátory
- Počet složených klíčů
- Počet hierarchií
- Počet hierarchií s více než jedním potomkem

Detekce těchto chyb probíhá v `QuantityDefectChecker` a jejím výsledkem je `QuantityDefect`, který popisuje požadované minimum, maximum a skutečný počet výskytu jednotlivých jevů.

`UsageDefect` popisuje použití modelovacích prvků z předepsané množiny. Existují čtyři druhy této chyby. Jsou to:

- Využití požadovaných kardinalit
- Využití požadovaných párů kardinalit
- Využití požadovaných kardinalit na vícehodnotových atributech
- Využití druhů hierarchií (kombinací pokrytí a překrytí)

Detekce těchto chyb probíhá v `UsageDefectChecker` jejím výsledkem je `UsageDefect`, který popisuje požadovanou množinu k využití a množinu prvků, které byly využity.

6.2.4 Problémy

Zdaleka největším problémem implementace této části programu, byla detekce správnosti slabých entit. Komplexita této funkcionality tkví v počtu všech způsobů, kterými se dá slabá entita vymodelovat špatně. Jsou jimi:

- Slabá entita má vlastní klíč
- Předek slabé entity je silná entita
- Složený slabý klíč slabé entity je špatně pokud:
  - Neobsahuje vztah
  - Obsahuje pouze jeden vztah a žádný atribut
  - Kardinalita u vztahů je jiná než 1..1

  - Obsahuje vícehodnotový atribut
- Slabá entita je rekurzivně identifikována jen slabými entitami a žádnou entitou s vlastním klíčem
- Slabá entita je spojena s identifikační entitou dvojicí kardinalit 1..1 a 1..1

Řešením tohoto problému byla jednoznačná identifikace výše vyčtených způsobů.

6.3 Výstup programu

Výstup programu se skládá ze tří částí:

1. Výpis dat o ERD z procesu parsování - `DiagramOutputFormatter, DiagramComponentStringifier`
2. Výpis chyb a celková tabulka bodů - `CorrectorOutputFormatter`, `DefectStrigifier`
3. Výpis relačního modelu (viz předchozí sekce 6.4) - `RelationalOutputFormatter, DiagramRelationalModelStringifier`

Každé části odpovídají dvě třídy, které jsou odpovědné za jejich vznik.

Z těchto dvou `Stringifier` odpovídá za přepis vnitřních datových struktur na odpovídající textové reprezentace. Jedná se o implementaci návrhového vzoru Visitor, který nám umožňuje rozšířit chování původní objektu.

Druhá třída `Formatter` odpovídá za celkový vzhled a kompozici výstupu. K tomuto účelu volá metody třídy `Stringifier` a formátuje jejich výstupu do cíleného vzhledu.

6.3.1 Problémy

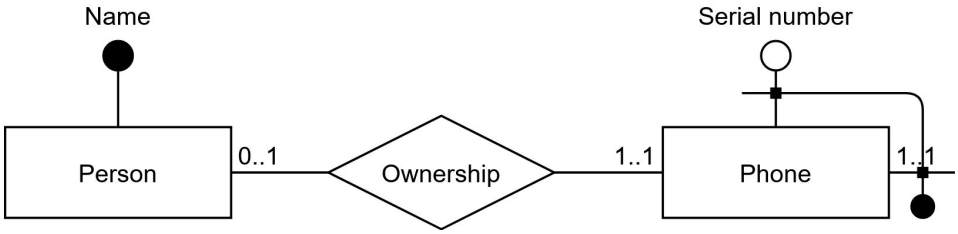
Největší výzvou této části implementace, bylo vytvoření přehledného a flexibilního formátovacího nástroje. Tento nástroj by se měl nabízet úpravám k přizpůsobení výstupu programu.

V ideálním případě, by se formátování výstupu dalo upravit pomocí argumentu při spuštění či pomocí další konfigurace. Tato provázanost však nakonec nebyla implementována.

Změny se tak dají provádět pouze z kódu pomocí formátovacích funkcí tříd `Stringifier`.

6.4 Převod ERD do relačního schématu

Vzhledem k tomu, že už jsme vytvořili vnitřní datovou strukturu pro ERD a umíme ho také parsovat a vypsat jeho části, napadlo nás implementovat i převod ERD do RM.



Obrázek 6.1: Slabá entita s redukovatelným vztahem

Převod se řídí pravidly popsanými v kapitole 4.2 a je založen na stejném principu návrhového vzoru *Visitor* jako výstup z předchozí kapitoly 6.3.

Převod datové reprezentace ERD (viz obr. 5.1) probíhá ve třídě `DiagramRelationalModelStringifier`. V té se nachází metody pro převod jednotlivých elementárních prvků - entit, vztahů, atributů, cizích klíčů.

Dále bylo k tvorbě cizích klíčů RM třeba vytvořit vnitřní tříd `RelationalId`, která slouží k jednoznačné identifikaci RM tabulky. Tato třída reprezentuje strom, ve kterém jsou k identifikaci kořene použity jeho potomci. Ve většině případů je kořenem entita a následovníkem některý její klíčový atribut.

V případech vtahů a slabých entit je poté nutné použít strom s větší hloubkou. Například v případě řetězce slabých entit, kde jsou slabé entity identifikovány jinými slabými entitami, musíme k vytvoření cizího klíče postupně projít a vypsat tento strom.

6.4.1 Problémy

Základní pravidla pro převod ERD do RM jsou vcelku jasná a jednoduchá. Avšak kombinovanou aplikací těchto pravidel vznikají problémové situace, které jsou značně složitější. Navíc se ukázalo, že v reálných ERD tyto situace nastávají poměrně často.

Například v situaci na obr. 6.1 by mohl být vztah zredukován podle pravidel převodu ERD vztahů do RM. [8] Na to by však musel být v entitě *Phone* vytvořen cizí klíč *Name*. Ten by však v kontextu *Phone* byl také klíčem a tudíž by entita přestala být slabou. Tudíž bude mít vztah *Ownership* vlastní tabulku.

Dalšími příklady komplikovaných kombinací pravidel jsou:

- Vícehodnotový atribut navázaný na vztah, který bude při převodu do RM zredukován.
- Rekurzivní vztahy
- Už výše zmíněný řetězec slabých entit.

Kapitola 7
Testování

V této kapitole popíšeme proces testování navrženého programu a přínosy aplikace, které byly s jeho pomocí odhaleny.

V rámci automatizovaného testování aplikace byly implementovány jednotkové testy komponent. Integrační a akceptační testy byly prováděny manuálně na dvou sadách ER diagramů.

Jednu sadu tvořily diagramy, které byly vytvořeny k otestování velkého množství krajních případů. Druhou sadu tvořili diagramy, které byly založeny na řešeních studentů předmětu DBS.

7.1 Testování v rámci BRUTE

Vývoj aplikace probíhal v iteracích. V rámci těchto iterací byl program pravidelně nasazován do systému BRUTE. Tím byla otestována základní funkcionalita v cílovém prostředí.

Dále byl program otestován na skutečných řešeních úlohy "Tvorba ERD" od studentů předmětu DBS. Toto testování probíhalo srovnáním výstupu programu se seznamem chyb, které odhalil učitel předmětu. Na základě výsledků těchto testů byla postupně odhalena a opravena řada nedostatků aplikace.

Zároveň byl také ověřen přínos aplikace. Prokázalo se, že aplikace nebude schopná zkontrolovat všechny možné druhy chyb (viz *Sémantické chyby* v kapitole 2.2.2). Avšak k opravě základních modelovacích chyb ERD a zejména ke kontrole dodržení zadání úlohy, slouží aplikace velice dobře.

Například v případě kontroly použitých kardinalit v ERD by musel učitel procházet celý diagram a dělat si výčet přítomných kardinalit, aby odhalil dvojice, které se zde nevyskytují. Pro vytvořený program je tato kontrola však triviální.

V rámci jednoho řešení může aplikace jen na této chybě ušetřit minutu času. Tato časová úspora dále dramaticky roste v kombinaci s počtem studentů, kteří úkoly odevzdávají, a tím, že možných chyb, jejichž kontrolu můžeme urychlit, se v ERD vyskytuje téměř 30 v závislosti na zadání.

Ušetření času učitele není však jediným přínosem. Při testování bylo odhaleno další využití programu. Slouží jako seznam chyb, na jejichž kontrolu

je snadné při opravě velkého množství řešení zapomenout. Je to tedy jakási garance chyb, které budou u každého studenta opraveny.

Ukázalo se, že ačkoliv aplikace nebude schopná autonomně opravovat odevzdané úkoly v rámci systému BRUTE, slouží jako dobrá pomůcka pro učitele, kteří opravu provádějí. Ušetří jim čas strávený nad opravou odevzdaného řešení a poskytne u každého odevzdaného ERD minimální výstup opravy.

7.2 Testování požadavků

Při testování požadavků stanovených v kapitole 5.1 jsme shledali, že program splňuje všechny požadavky do uspokojivé míry.

Dále jsme porovnali výstup procesu hledání chyb s kritérii hodnocení úlohy (viz kapitola 1.2). Zde jsme si ověřili, že program dokáže zkontrolovat, každé stanovené kritérium. Navíc se v konfiguraci programu dá nastavit kontrola dalších kritérií nad rámec požadavků stávajícího zadání úkolu (např. počet potomků v hierarchii).

Poté jsme si ověřili, že program kontroluje všechny základní modelovací chyby (viz 2.2.2). Zde jsme shledali, že tzv. *syntaktických chyb* se buď v modelovacím nástroji ERDia nelze dopustit (např. spoje, které nikam nevedou), a nebo jsou dostatečně ohlídané v rámci chyb `BasicDefects`.

Nakonec jsme potvrdili, že tzv. *sémantické chyby* (viz 2.2.2) není možné v rámci programu kontrolovat.

Celkový seznam chyb, které program kontroluje, se nachází v příloze B.

7.3 Testování implementace

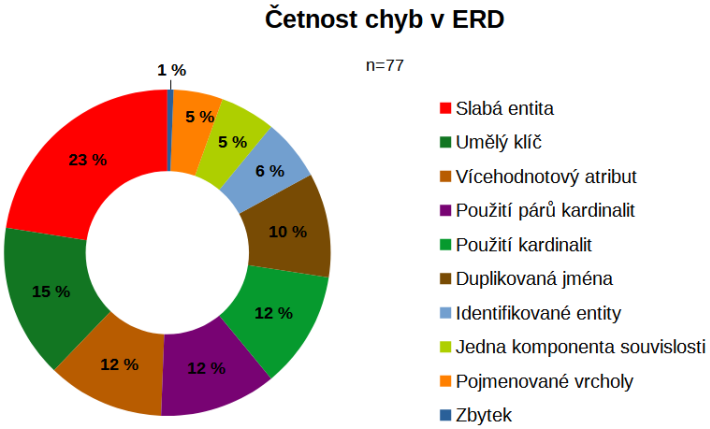
V průběhu manuálního akceptačního testování identifikovali a opravili následující chyby implementace:

- Proporcionální bodování chyby - využití kardinalit
- Přidání chyby - využití dvojic kardinalit
- Pojmenování entit, vztahů a atributů způsobem “new entity” se počítá jako chyba
- Správná identifikace správnosti slabých entit
- Správné řazení přítomných chyb na výstupu
- Zavedení kódování výstupu pro českou lokalizaci
- Zavedení formátu HTML pro výstup s možností stylizace pomocí kaskádových stylů
- Nesprávné cizí klíče v RM
- Nesprávné redukování vztahů v RM

7.4 Statistika

Na základě 77 ERD odevzdaných studenty byla také vytvořena následující statistika četnosti přítomných chyb 7.1. Je vidět, že nejčastější chybou je nesprávné modelování slabých entit, využití umělých identifikátorů a nedostatečné využití druhů a dvojic kardinalit.

Do statistiky nebyly započteny kvantitativní chyby typu "počet entit", neboť záleží na konkrétním zadání a konfiguraci programu. Tabulka četností, z které byl graf vytvořen, se nachází v příloze C.



Obrázek 7.1: Četnost nalezených chyb

Kapitola 8

Závěr

Cílem této práce bylo analyzovat možnosti automatického vyhodnocování úloh v předmětu Databázové systémy se zaměřením na tvorbu entitně vztahových diagramů a vytvořit program, který by úlohy opravil a vyhodnotil.

V analytické části jsme nejdříve shrnuli základní informace o ERD (kapitola 2.1) a relačních modelech (kapitola 4.1). Poté jsme porovnali dostupné nástroje pro tvorbu ERD (viz 3.1). Nakonec jsme na základě těchto informací vytvořili návrh (kapitola 5.1) pro výslednou aplikaci, která bude splňovat požadavky zadání.

Dále jsme pomocí návrhu vytvořili program, který dokáže vyhledat a vyhodnotit chyby v ERD. Navíc jsme i dokázali implementovat převod ERD do relačního modelu (viz 6.4). Aplikaci jsme otestovali v cílovém prostředí systému BRUTE a na základě těchto testů opravili nedostatky aplikace. Nakonec jsme vyvodili přínosy aplikace v kapitole 7.1.

Program byl využit k opravě ERD cca 40 studentů předmětu DBS v letním semestru 2022/23. Zde úspěšně detekoval všechny syntaktické chyby v modelování (viz 2.2.2) a v dodržení kritérií zadání (viz 1.2).

Nad rámec zadání byl také implementován převod entitně vztahového diagramu do odpovídajícího relačního modelu.

Celkově výsledný program splnil požadavky a očekávání zadání a může být využit ke kontrole ERD v následujících letech výuky předmětu Databázové systémy.

Bibliografie

[1] T. A. Halpin a A. J. Morgan. *Information modeling and relational databases*. Elsevier/Morgan Kaufman Publishers, 2008.

[2] E. F. Codd. „Derivability, redundancy and consistency of relations stored in large data banks“. In: *ACM SIGMOD Record* 38.1 (2009), s. 17–36.

[3] M. C. Wijegunasekera a N. G. J. Dias D. Pieris. „ER to Relational Model Mapping: Information Preserved Generalized Approach“. In: International Postgraduate Research Conference 2019. 2019.

[4] M. Kolp a E. Zimanyi. „Enhanced ER to relational mapping and interrelational normalization“. In: *Information and software technology* 42.15 (2000), s. 1057–1073.

[5] P. Pin-shan Chen. „The Entity-Relationship Model: Toward a Unified View of Data“. In: *ACM Transactions on Database Systems* 1 (1976), s. 9–36.

[6] Chengfei Liu a Jianxin Li. „Designing Quality XML Schemas from E-R Diagrams“. In: čvn. 2006, s. 508–519. ISBN: 978-3-540-35225-9. DOI: 10.1007/11775300_43.

[7] N. Opinaldo. *Top 10 Free ER Diagram Tools in 2022*. 2022. URL: https://gitmind.com/er-diagram-tool.html (cit. 20. 04. 2022).

[8] S. Bagui a R. Earp. *Database design using entity-relationship diagrams*. Crc Press, 2011.

[9] V. Řepa. *Vývojové trendy metodik vývoje informačních systémů - výzva BPR*. 1999.

[10] V. Sobotíková. *B0B36DBS: Database Systems - Anotace*. 2022. URL: https://fel.cvut.cz/cz/education/bk/predmety/50/10/p5010606.html (cit. 25. 04. 2022).

[11] P. Stejskal. *Modelovací nástroj pro ER konceptuální návrh databází*. Bakalářská práce. 2020.

[12] M. Svoboda. *B0B36DBS: Database Systems*. 2022. URL: https://www.ksi.mff.cuni.cz/~svoboda/courses/182-B0B36DBS/ (cit. 20. 04. 2022).

[13] T. Halpin a T. Morgan. *Information modeling and relational databases*. Morgan Kaufmann, 2010, s. 525–526.

[14] M. Evans a Eui Kyun Park Il-Yeol Song. „A comparative analysis of entity-relationship diagrams“. In: *Journal of Computer and Software Engineering* 3.4 (1995), s. 427–459.

Příloha A

Seznam zkratek

Zkratka	Význam
ERD	Entity vztahový diagram
DBS	Předmět databázové systémy
RM	Relační model, relační schéma
XML	Extensible Markup Language
HTML	Hypertext Markup Language
DOM	Document Object Model
SAX	Simple API for XML
UML	Unified Modeling Language
JSON	JavaScript Object Notation

Příloha B

Seznam kontrolovaných chyb ERD

B.1 Základní chyby - BasicDefects

- Anotované hierarchie
- Platné kardinality vícehodnotových atributů
- Pojmenované vrcholy
- Duplikátní názvy entit a vztahů
- Duplikátní názvy atributů na stejném vrcholu
- Entity mají identifikátor
- Slabé entity jsou identifikovatelné
- Kardinality na spojích
- Jedna komponenta souvislosti
- Cyklická hierarchie
- Umělý identifikátor
- Vícehodnotový atribut jakožto klíč
- Jednoduchý klíč součástí složeného klíče

B.2 Chyby počtu využití prvků - QuantityDefects

- Počet atributů
- Počet vícehodnotových atributů
- Počet strukturovaných atributů
- Počet entit
- Počet vztahů

- Počet rekurzivních vztahů
- Počet slabých entit
- Počet entit s několika identifikátory
- Počet složených identifikátorů
- Počet hierarchií
- Počet ternárních a vyších vztahů
- Počet potomků v hierarchiích

B.3 Chyby využití prvků - UsageDefects

- Využití druhů kardinalit
- Využití dvojic kardinalit
- Využití kardinalit na vícehodnotovém atributu
- Využití druhů hierarchií

Příloha C

Četnosti chyb v ERD

Chyba	Počet
Slabá entita	37
Umělý klíč	25
Vícehodnotový atribut	19
Použití párů kardinalit	19
Použití kardinalit	19
Duplikovaná jména	17
Identifikované entity	10
Jedna komponenta souvislosti	9
Pojmenované vrcholy	8
Zbytek	1
Celkový počet	**77**

Příloha D

Seznam přiložených souborů

Jméno	Popis
project.zip	Projekt v archivu ZIP
chyby.txt	Seznam všech rozpoznávaných chyb
input.xml	Příklad vstupu programu
example_config.txt	Příklad vstupní konfigurace
output.html	Příklad výstupu programu pro vstup input.xml a example_config.txt
cetnosti.pdf	Tabulka četnosti chyb ve studentských ERD