TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika

Studijní obor: 1802T007 – Informační technologie

Způsoby vytváření databázových informačních systémů založených na relačních a objektově orientovaných databází

Database information systems design based on relation and object oriented databases

Diplomová práce

Autor: **Bc. Lukáš Nový**

Vedoucí práce: Ing. Roman Špánek, Ph.D

Konzultant: Ing. Pavel Tyl

V Liberci 20. 5. 2011

Originál zadání

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Tímto bych chtěl poděkovat Ing. Romanu Špánkovi Ph.D. za vedení této diplomové práce, cenné rady a připomínky při vypracování.

Abstrakt

Diplomová práce pojednává o procesu návrhu databázových systémů založených na různých databázových modelech. Konkrétně práce popisuje relační databázový model, objektově orientovaný databázový model a model objektově relační.

Práce rovněž obsahuje rešeršní část, ve které jsou popsány dostupné objektově orientované databázové systémy, jejich přednosti a nedostatky.

Součástí práce je dále studium konkrétních rozdílů v analýze a návrhu databázových informačních systémů založených na relačním a objektovém modelu. Jsou popsány jednotlivé vývojové fáze a jejich úloha v životním cyklu návrhu databázového systému. Podrobně jsou popsány ERD a DFD diagramy a dále jsou pak uvedeny metody objektově orientované analýzy pomocí jazyka UML.

Závěrem je uveden a zdokumentován postup pro řešení konkrétního databázového problému, na kterém je demonstrován rozdíl v relačním a objektovém přístupu.

Klíčová slova: Objektové databáze, relační databáze, návrh informačního systému, ERD, UML

Abstract

The thesis discusses the process of database systems design based on different database models. Specifically describes relational database model, object-oriented database model and object-relational model.

The thesis also contains a search section, which describes the available object-oriented database systems, their strengths and weaknesses.

The thesis is to further study specific differences in the analysis and design of database information systems based on relational and object model. There are described various developmental stages and their role in the life cycle of database design. Are described in detail the ERD and DFD diagrams and then the methods are object-oriented analysis using UML.

The conclusion is given and documented process for a specific database problem, which is demonstrated by the difference in relational and object-oriented approach.

Keywords: Object-oriented databases, relational databases, information system design, ERD, UML

Obsah

Seznam zkratek a symbolů

SQL - Structured query langure

DDL - Data definition langure

DML – Data manipulation langure

OOP – Objektově orientované programování

OID – Object identifier

ODMG - Object database management group

ODL - Object data definition language

OQL - Object query language

CSP - Caché server pages

ERD – Entity relationship diagram

DFD – Data flow diagram

UML – Unified modeling langure

Úvod

Tato diplomová práce by měla poskytnout přehled používaných metodik návrhu a analýzy databázových informačních systémů.

V současné době se stále více prosazují objektově orientované databázové systémy. Ty jsou však od tradičních relačních databázových systémů značně odlišné. To sebou přináší i nutnost využití odlišných technik návrhu a analýzy, které jsou pro objektově orientované systémy vhodné. Teoretická část práce poskytne podrobný rozbor metod užívaných jak pro návrh relační tak i objektově orientované databáze. V praktické části budou tyto poznatky demonstrovány na řešení konkrétního databázového problému.

1 Úvod do databázových modelů

1.1 Relační databázový model

Na úvod této kapitoly si nejdříve definujme některé důležité pojmy a termíny z databázového světa.

- **Databáze**

Pod pojmem databáze se rozumí určitá množina spolu navzájem souvisejících a logicky uspořádaných dat. Tato data se týkají určitého výseku skutečnosti reálného světa.

- **Systém řízení báze dat**

Je obecně softwarový systém, který spravuje a zajišťuje mechanizmy potřebné pro chod a práci s databází.

- **Databázový systém**

Databázový systém se skládá z databáze a systému řízení báze dat.

Relační databázový model je založen na matematickém aparátu z teorie relačních množin [3].

Definice 1:

Mějme množinu D1, D2, …, Dm. Potom R je relací nad touto množinou, jestliže $R \subseteq D1 \times D2 \times \ldots \times Dm$. *Prvky relace R jsou uspořádané n-tice (D1, D2, …, Dm). Di je i-tou doménou relace R.*

Definice 2:

Relaci v relačním modelu dat budou představovat všechny uspořádané trojice <A, D, T>, kde: A je konečná množina hodnot jmen atributů, D je zobrazením přiřazující všem jménům atributů a $\in A$ *doménu D(a), T je zobrazením konečné podmnožina kartézského součinu všech jmen atributů domén D(a).*

Základem relačního modelu je *relace*. Relace je možné zobrazovat v alternativní formě své podoby jako relační tabulky. Každá relační tabulka má své *relační schéma*, které definuje vlastní strukturu tabulky. Relační schéma tabulky se skládá z jejího názvu, seznamu atributů a domén jednotlivých atributů.

Definice 3:

Schéma relace R=(A1:D1, A2:D2, …, Am:Dm), kde R je název relace, Ai jsou jména atributů a Di domény těchto atributů.

Atributy relačních tabulek jsou jejich sloupce a doménou atributu je množina přípustných hodnot. Řádek tabulky (n-tice) pak představuje konkrétní záznam, který je v ní uložen.

Relační tabulka má následující vlastnosti:

- Nezáleží na pořadí atributů.
- Nezáleží na pořadí řádků.
- Žádné 2 řádky relační tabulky nejsou shodné. Toto pravidlo plyne z teorie relačních množin, relační tabulka představuje množinu a prvek množiny je uváděn pouze jednou.
- Každá hodnota daného atributu je ze stejné množiny hodnot (domény).
- Každý atribut má jednoznačný identifikátor.
- Název tabulky je jednoznačně identifikovatelný.
- Data v tabulce musí vyhovovat integritním omezením.

Pro úplnost si ještě řekněme, že pod pojmem *databázové schéma* rozumíme množinu všech jednotlivých relačních schémat a *integritních omezení*. Integritní omezení jsou množinou tvrzení omezujících hodnoty v n-ticích jednotlivých relací. Zajišťují tedy, že do databáze budou ukládána pouze data, která vyhovují předem stanoveným pravidlům.

Dále si definujme pojmy jako nad-klíč, klíč, primární klíč a cizí klíč [4].

- **Nad-klíč**, je množina atributů pomocí, kterých můžeme identifikovat řádek tabulky.
- **Klíč**, je nejmenším možným nad-klíčem.
- **Primární klíč** je vybraným klíčem, který jednoznačně identifikuje řádek tabulky.
- **Cizí klíč** se zavádí při vztahu mezi dvěma tabulkami. Jedná se o speciální typ integritního omezení, které zajišťuje referenční integritu. Cizí klíč je atributem nebo skupinou atributů, které jsou v druhé referenční tabulce primárním klíčem.

Při návrhu relační databáze se můžeme setkat s několika podstatnými problémy, které jsou nežádoucí a je nutné je eliminovat. Prvotní cílem je zamezit vzniku redundantních dat. Znamená to tedy, že se snažíme co nejvíce zabránit opakování dat v různých tabulkách, bez toho aby došlo k jejich ztrátě.

Dalším nežádoucím jevem vznikajícím při špatném návrhu relační databáze jsou anomálie. Anomálie mohou být následujících typů:

- **Aktualizační** anomálie vznikne, pokud je ve špatně navrženém schématu změněn některý atribut. Změnu je poté nutné provést u všech záznamů s daným atributem. Pokud by byl nějaký záznam opomenut, došlo by k anomálii aktualizace.
- **Vložení** je typem anomálie, která nastane při vložení nového záznamu do tabulky. Není možné vložit pouze konkrétní atribut, je nutné znát i všechny ostatní atributy, které tabulka obsahuje.
- **Odstranění** je anomálií, která vznikne odebráním některého záznamu z tabulky. Při odstranění záznamu dojde ke ztrátě informace.

Všechny tyto zmíněné problémy při návrhu relační databáze, je možné při dodržení jistých pravidel eliminovat. Soubor těchto pravidel pro optimální návrh relační databáze se nazývá *normalizace*. Při návrhu databáze pomocí dekompozice je relační schéma postupně rozdělováno do menších tabulek. Jakým způsobem tabulky strukturovat, udávají jednotlivé normální formy.

Ještě než si jednotlivé normální formy představíme, definujme si pojem *funkční závislost*. Funkční závislostí rozumíme takový vztah mezi obecně dvěma množinami atributů, že první atributy určují hodnoty atributů v druhé množině.

Definice 4:

Nechť A, B jsou atributy relace R. B je funkčně závislý na A, pokud každé hodnotě A odpovídá jediná hodnota atributu B, tj. existuje funkce f: A→B.

- **1. Normální forma** (1. NF) – relace je v 1. NF, pokud všechny hodnoty atributů jsou atomické. Hodnoty v atributech tedy nesmí být vícehodnotové a musí být dále nedělitelné.
- **2. Normální forma** (2. NF) – relace je v 2. NF, pokud je v 1. NF a zároveň jsou všechny neklíčové atributy plně funkčně závislé na primárním klíči. Plně závislé musí být na všech částech primárního klíče. Z této definice vyplývá, že 2. NF má svůj význam v případě, že existuje více-atributový primární klíč.
- **3. Normální forma** (3. NF) – relace je v 3. NF, pokud je v 2. NF a všechny atributy, které nejsou součástí primárního klíče, jsou funkčně závislé pouze na primárním klíči. Z uvedené definice plyne, že nesmí existovat žádné tranzitivní závislosti na neklíčových atributech nebo jen části primárního klíče.
- **Boyce – Coddova normální forma** (BCNF) – relace je v BCNF pokud je v 3. NF a navíc atributy, které jsou součástí primárního klíče, tranzitivně nezávisí na jiné části primárního klíče.

Dalšími normálními formami vyšších řádů se tato práce zabývat nebude, uvedeny budou jen pro doplnění. Pokud se totiž relační databáze nachází ve 3. nebo BCNF, jedná se o dostačující podmínku pro vyhnutí se strukturálním vadám. Dalšími formami vyšších řádů jsou 4. a 5. normální forma.

Pro úplnost uveďme další metody, používané pro vytvoření optimálního návrhu relační databáze. Jedná se o *dekompozici* a *syntézu*.

- **Dekompozice**, je metodou pro rozklad jednoho relačního schématu na více schémat. Jedná se o přístup návrhu „shora dolů“. Pro dekompozici musí platit, že výsledné relace musí zachovat stejný informační obsah, jaký měla původní relace. Dále by dílčí relace neměli porušit zachování sémantiky. Znamená to tedy, že funkční závislosti v relacích po provedené dekompozici musejí odpovídat původním.
- **Syntéza**, je metodou, při které jsou ze zadané množiny funkčních závislostí a atributů postupně skládány relace. Postupujeme tedy „zdola nahoru“. Z původní množiny funkčních závislostí vytvoříme minimální pokrytí tak, že nejprve funkční závislosti dekomponujeme na elementární vztahy. Dále z levé strany funkčních závislostí odstraníme redundantní atributy a nakonec odstraníme redundantní funkční závislosti. Takto vzniklá množina se rozdělí do skupin se stejnou levou stranou. Každá takto rozdělená skupina atributů představuje jedno relační schéma, kde primární klíč představuje levá strana závislosti.

V závěru této části bude zmíněn prostředek pro práci s relační databází. Tímto prostředkem je standardizovaný jazyk SQL (Structured query language). Jazyk SQL se skládá ze dvou hlavních částí:

- **DDL** (Data definition language) je částí pro definici dat. Využívá se pro jejich vytváření a modifikaci.
- **DML** (Data manipulation language) je určen k realizaci dotazů, vkládání, mazání a aktualizace dat.

1.2 Objektový databázový model

Objektový databázový model je úzce spjatý s principy, na kterých je založeno objektově orientované programování. Objektové technologie umožňují lépe zachytit a popsat principy a fungování reálného světa oproti strukturálnímu přístupu. V úvodu této

kapitoly budou vysvětleny stěžejní pojmy, týkající se nejen objektově orientovaného programování ale objektových technologií jako celku.

- **Objekt** je základním prvkem objektového přístupu. Objekt zde vystupuje jako abstrakce entit reálného světa, má své chování a vlastnosti. Vlastnostmi objektu jsou jeho atributy, chování definují konkrétní metody objektu. Objekt má tedy svůj datový obsah a také svou funkcionalitu. Objekt není skalárem, může obsahovat další objekty. Objekty mají dále tu vlastnost, že jsou znovupoužitelné. Lze je opětovně využívat v aplikacích.

V programu objekt vystupuje jako uzavřená struktura s vnitřní pamětí, která je z vnějšku objektu nepřístupná. Implementace objektu je při použití *zapouzdření* skryta a navenek je přístupné pouze jeho rozhraní. Objekt je schopen přijmout a zpracovat zaslanou zprávu, zasílání zpráv je jedinou možností jak s objektem komunikovat. Data jsou tedy uvnitř objektů zapouzdřená. Zapouzdření zvyšuje mimo jiné bezpečnost dat a zamezuje jejich duplicitě.

- **Třída** je „šablonou“ pro vytváření objektů. Je to nějaký obecný popis toho jaké atributy a funkčnost budou objekty vytvořené ze třídy mít. Objekt vytvořený ze třídy je její instancí. Všechny objekty vytvořené z jedné třídy mají stejné atributy a stejnou funkcionalitu.
- **Dědičnost** zajišťuje, že třídy mohou dědit atributy a metody od nadřazených tříd. K těmto odvozeným vlastnostem a funkcionalitě je možné přidat vlastní rozšíření. Třída, od které se dědí, se nazývá rodičovská, ostatní třídy jsou jejími potomky. S procesem dědičnosti souvisí pojmy *generalizace* a *specializace*. Můžeme říci, že určitá třída je generalizací (zobecněním) jedné nebo více speciálních tříd. Dědičnost odstraňuje duplicitu při definici tříd, pokud mají některé společné prvky.
- **Polymorfismus** si můžeme definovat jako odlišné chování různých objektů při reakci na stejnou zprávu. Každý objekt na zaslanou zprávu reaguje vykonáním jiné metody. Toto různé chování objektů souvisí s dědičností. Třídy, které jsou potomky generální třídy, implementují její metody. Při zaslání zprávy se pak ve speciálních třídách vykonají metody stejného názvu ale s různou implementací. Polymorfismus pomáhá zajistit věrohodnější analogii s modelováním reálného světa.

Objektový databázový model umožňuje využití všech těchto výše zmíněných dovedností objektových technologií. Objektové databáze dále rozšiřují objektově orientované programování (dále jen OOP) o další funkčnost. Tímto rozšířením je především obohacení OOP *o perzistenci* dat.

Perzistence dat je obecnou vlastností databázových systémů. Perzistencí rozumíme schopnost systému uchovávat data i mimo svůj běh. Objektově orientované a další skupiny programovacích jazyků přináší podporu pro správu dat po dobu svého běhu. Vyznačují se ale slabou podporou pro data, která je nutné uchovávat perzistentně. Mezeru, která vzniká tímto nedostatkem, v OOP vyplňují právě objektové databázové systémy. Ty umožňují perzistentně uchovávat objekty vzniklé z různorodých tříd. Takto vzniklé objekty mohou mít, na rozdíl od dat v relačních databázích, heterogenní strukturu.

Objektový databázový model přináší sjednocení vývoje databázového modelu a objektově orientovaných programových aplikací. Návrh takovýchto aplikací je daleko přirozenější a efektivnější. Díky podpoře datových struktur, které jsou běžně používané v OOP, není nutné data při ukládání do objektové databáze složitě transformovat.

Objektově orientované databázové systémy tedy podporují množiny objektů, různých typů. Tyto množiny objektů jsou známy pod označením *kolekce*. V relačním databázovém modelu je jediným takovým „typem“ relační tabulka.

V objektově orientovaných databázích dále odpadá nutnost definice primárních klíčů. Každý objekt má svůj *OID* (*Object identifier*). OID je jednoznačným identifikátorem každého objektu. OID je objektu přidělen po celou dobu jeho existence, bez ohledu na změnu jeho datové struktury.

Třída a *množina* jsou v objektových databázových systémech odlišné struktury. Třída, jak vyplývá z úvodu této kapitoly, představuje datový typ, zatímco množina je úložištěm objektů. Objektové databáze dovolují pracovat i s množinami obsahující objekty vytvořené z jiných tříd. V relačním databázovém modelu jsou, třída a množina, realizovány v jednom celku relační tabulkou.

Vlastnosti a charakteristiku objektově orientovaných databází jsme si již představili. Nyní se ještě zmíníme o standardu objektově orientovaných databázových systémů. Standardizace by měla sjednotit metodiku vývoje tak, aby byly produkovány univerzální a snadno přenositelné systémy.

Nejprve je ale nutné zmínit, že žádný standard, který by byl uznávaný všemi výrobci a vývojáři objektových databázových systémů, doposud neexistuje.

Existuje skupina sdružující komerční firmy a vývojáře, která se o sestavení takového standardu pokusila. Tímto standardem ve své nejaktuálnější verzi je *ODMG 3.0*. ODMG (Object database management group) [26] je skupinou zabývající se objektově orientovanými databázovými systémy. Standard ODMG 3.0 vychází z obecnějšího objektového standardu skupiny *OMG* (Object management group) [9]. Standard ODMG 3.0 definuje:

- **Architekturu** objektově orientovaných databázových systémů.
- **Datový model** zahrnující definici objektů, atributů, vztahů mezi objekty a objektové typy.
- **ODL** (Object data definition language) jazyk pro definici dat.
- **OQL** (Object query language) objektový dotazovací jazyk.

1.2.1 Objektově-relační databázový model

Objektově-relační databázový model přináší některé objektové prvky do relačních databází. Rozšiřuje relační model o další funkcionalitu. Jedná se především o podporu některých složitějších datových struktur známých z OOP [6].

Základem jsou stále relační tabulky používané v relačním modelu. Je možné definovat objektové třídy a jejich metody. Podporuje práci s vnořenými tabulkami. Záznam v tabulce pak může odpovídat několika záznamům v tabulce vnořené. Objektově-relační model však stále zůstává ve své podstatě modelem relačním.

1.3 Shrnutí kapitoly

Závěr této části bude věnován shrnutí celého úvodu práce, pojednávajícího o databázových modelech.

Relační databáze jsou dnes nejpoužívanější databázovou technologií. Pro relační databáze hovoří jejich vysoký stupeň rozšíření a implementace. Vyznačují se svou relativní jednoduchostí a vysokým výkonem. Jsou standardizovány a mají jasně definovanou metodiku návrhu. Omezením je obtížná práce se složitějšími datovými strukturami a podpora pouze atomických množin dat. Nejsou přímo použitelné v OOP.

Objektové databáze si začínají získávat své místo v databázovém světě. Přináší perzistenci do OOP a podporu pro práci se složitějšími datovými strukturami. Jsou přímo použitelné v dnes velmi rozšířeném OOP. Chybí všeobecně uznávaný standard. Jsou méně známé a z toho plyne jejich menší nasazení, oproti již zavedené relační technologii.

Relační databázový model	**Objektový databázový model**
Záznam	Objekt
Relace	Třída, množina
Atribut	Atribut, metoda
Primární klíč	OID - Object identifier

Obrázek 1 Srovnání relačního a objektového databázového modelu

	Relační databáze	**Objektové databáze**
Míra rozšíření	Vysoká	Nízká
Popis reality	Obtížný	Snadný
Složitější datové struktury	Ne	Ano
Jednotný standard	Ano	Ne

Obrázek 2 Některé vlastnosti relačních a objektových databází

2 Vybrané objektově orientované databáze

V této kapitole budou představeny některé z konkrétních objektově orientovaných databází, které jsou v současné chvíli na trhu dostupné. Bude popsána jejich základní charakteristika a závěr této kapitoly nabídne jejich vzájemné srovnání.

2.1 Db4o

Db4o je objektovou databází, kterou od roku 2008 získala a na jejím dalším vývoji se podílí firma *Versant*. Db4o je k dispozici primárně pod otevřenou licencí a jako open source. Pro komerční užití je připravena zpoplatněná verze db4o, která v sobě zahrnuje i další podporu od společnosti Versant [11].

Db4o je takzvanou embedded databází, která pro svůj běh využívá adresní prostor aplikace. Embedded databáze je knihovnou, která se k aplikaci pouze přilinkuje. Aplikace má tak prostřednictvím knihovny přímý přístup k datovému souboru. Provozovat je možné i velmi jednoduchý klient – server režim.

Db4o je navržena pro práci na platformách NET. a Java. Db4o je možné použít na jakémkoliv operačním systému, který tyto platformy podporuje. V době tvorby této práce je nejaktuálnější db4o verze 8.0.

Podporované jazyky	
Java	JDK 5+, Scala
.NET	C#, Visual Basic

Obrázek 3 Podporované jazyky db4o

Vybrané vlastnosti	
Režimy	Lokální, klient-server
Dotazovací jazyky	Nativní dotazy, SODA QBE(Query By Example) LINQ (Lang. int. query)
Max. velikost souboru	254 GB
Transakce	Násobné, paralelní, ACID

Obrázek 4 Vybrané vlastnosti db4o

2.2 Caché

Databáze Caché přináší možnost pracovat s daty jak čistě objektově tak i relačně. Na trh byla uvedena v roce 1997 společností *InterSystems*. Jedná se o vysoce výkonnou objektovou databázi, umožňující především rychlý vývoj webových aplikací pomocí technologie *CSP* (Caché server pages) a možnost využití jazyka SQL. Databáze je založena na objektovém standardu skupiny ODMG. V současné chvíli je posledním vydáním databáze Caché verze 2010.2 [12].

Data jsou v Caché ukládána ve vícerozměrných strukturách. Caché umožňuje více druhů přístupu k těmto datům. Přístupy lze realizovat zároveň, i při současném přístupu k jedné položce. Přístupy lze realizovat:

- **Pomocí jazyka SQL**.
- **Objektový databázový přístup**.
- **Vícerozměrný přístup k datovým polím**.
- **Pomocí jazyka HTML**.

Rozdílné formy přístupu k datům poskytují možnost různého pohledu na data. Na data tak může být pohlíženo jako na objekty ale stejně tak i jako na tabulky a vícerozměrná pole. Data jsou definována pouze jednou a dále je možné s nimi pracovat jako s objekty nebo tabulkami. Navržena je k tomuto účelu unifikovaná datová architektura, kterou databáze disponuje. Tato funkcionalita umožňuje mimo jiné snadný převod dat z relační databáze na objekty [13].

Caché může být provozována pod většinou známých operačních systémů, jako jsou Windows, Linux, Unix, Mac Os a další. Caché je komerčním placeným produktem. Pro nekomerční využití a akademické účely jsou k dispozici bezplatné verze s některými omezeními.

Činnost systému Caché zajišťují tyto části:

- **Datový server**, zajišťující přístup k vícerozměrným datům, objektům a přístup pomocí jazyka SQL.
- **Aplikační server**, zajišťující objektově relační mapování a podporu pro skriptovací jazyky.
- **Webové funkce**, například technologie CSP a další.

Vybrané vlastnosti	
Rozhraní	ODBC, JDBC, NET.
Podporované jazyky	Caché ObjectScript (vnořené SQL, HTML, Javascript), Caché Basic, Java, C#, C++, Delphi, .NET, XML
Transakce	Vysoký výkon a rychlost

Obrázek 5 Vybrané vlastnosti Caché

2.3 Objectivity

Objectivity je komerční objektovou databází společnosti *Objectivity, Inc*. Implementuje standard ODMG 93. Poslední vydáním databáze je v současnosti verze Objectivity 10. Systém podporuje spolupráci s programovacími jazyky Java, C++, C#, Smalltalk nebo Python. Objectivity je možné provozovat pod operačními systémy Unix, Windows, Mac OS a některých distribucích Linuxu [14].

Objectivity je distribuovanou databází (data mohou být replikována na různých serverech) s podporou ACID (Atomicity, consistency, isolation and durability) transakcí. Objectivity disponuje mechanismem pro integraci relačních tabulek a možnost přístupu do jiných strukturovaných souborů. Databáze se vyznačuje poměrně vysokým výkonem, spolehlivostí a nízkými nároky na údržbu. Objectivity podporuje import a export do XML. Objectivity je vázána zpoplatněnou licencí, lze získat i omezenou verzi pro nekomerční užití a testování.

Pro realizaci dotazů nad daty v databázi Objectivity lze použít:

- **SQL++**.
- **EJBQL** (Enterprise JavaBeans Query Language).
- **LINQ** (Language Integrated Query).
- **PQE** (Parallel Query Engine).

2.4 Versant Object Database

Jedná se o vysoce výkonnou objektovou databázi vyvinutou společností Versant. V současnosti je aktuální verze 8.0.1. Versant je komerční objektovou databázi s rozvinutou podporou a bohatou dokumentací. Je možné získat i omezenou bezplatnou verzi. Versant podporuje objektově orientované jazyky Java, C#, C++. Tyto jazyky jsou přímo použitelné pro definici dat. Doplněna byla podpora i pro jazyky Smalltalk a Python [15].

Versant je možné provozovat v režimu klient-server nebo jako přilinkovaný soubor k aplikaci. Dotazovacím jazykem je VQL (Versant query language) vycházející ze standardu SQL 92, který je specifický dle použitého programovací jazyka. Specifika pro jednotlivé objektové programovací jazyky jsou:

- **Java** (JDOQL, EJBQL, OQL).
- **C++** (VQL, OQL).
- **C#** (VQL, OQL, LINQ).

Versant umožňuje integraci dat z relačních tabulek. Systém Versant je kompatibilní s operačními systémy Windows, Solaris a některými distribucemi Linuxu.

2.5 Jiné objektově orientované databáze

Další objektové databázové systémy nebudou dále charakterizovány ani detailněji popisovány. Pro přehlednost bude uveden pouze výčet některých dalších [16].

- **ObjectStore.**
- **Gemstone.**
- **ObjectDB.**
- **Eloquera.**
- **Voss.**
- **Matisse.**
- **Mercury.**
- **Siaqodb.**
- **EyeDB**.
- **Perst**.

2.6 Závěrečné srovnání

	Db4o	Caché	Objectivity	Versant
Open source	Ano	Ne	Ne	Ne
Podpora integrace relačních dat	Ne	Přirozená	Ano	Ano
Víceuživatelský přístup	Ano	Ano	Ano	Ano
Plná podpora SQL	Ne	Ano	Ne	Ne
Aktuální verze	8.0	2010.2	10	8.0.1
Výrobce	Versant	InterSystems	Objectivity,Inc	Versant

Obrázek 6 Závěrečné srovnání

3 Analýza a návrh databázového informačního systému

3.1 Vývojové fáze

Návrh informačního systému je strukturován do několika na sebe navazujících fází. Každá část návrhu má v životním cyklu vývoje informačního systému svůj význam. Úvod této kapitoly bude věnován jejich charakteristice a významu. Konkrétní postupy a metody k jejich realizaci budou popsány dále.

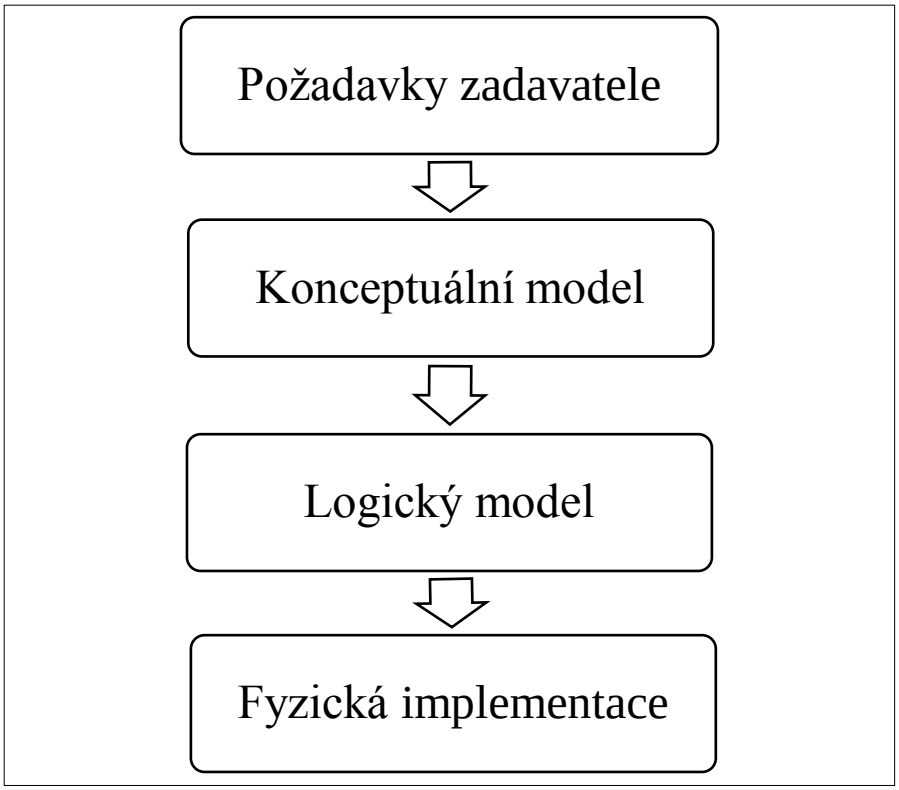
V počátku vývoje je nejprve nutné seznámit se s požadavky na funkcionalitu. Výsledkem by měla být množina uživatelských požadavků na datovou a funkční stránku informačního systému. Tato fáze vývoje se skládá z detailní komunikace s budoucími uživateli a zadavateli. V této fázi vývoje je vhodné využít například *případ užití* (známý také jako use-case).

Po úvodním shromáždění uživatelských požadavků je možné přistoupit k tvorbě *konceptuálního modelu*. Konceptuální model představuje nejvyšší úroveň abstrakce popisu databáze nebo informačního systému jako celku. Popisuje sémantiku budoucího systému. Je obecně aplikovatelný v jakémkoliv technologickém prostředí. Není ovlivněn budoucími prostředky řešení, určuje pouze to co je obsahem nikoliv formu. Konceptuální model je blízký lidskému vnímání reality. Jeho výsledkem je zachycení popisované části reality. Měli by ho poměrně snadno pochopit i lidé, kteří nejsou blíže seznámeni s technickou stránkou řešené problematiky.

Konceptuální model v sobě zahrnuje popis atributů a omezení kladená na data (vztahy, integritní omezení, atributy). Slouží pro potřeby dokumentace a snadnou komunikaci se zadavateli. Konceptuální model zajišťuje, že pohled na data nebude řešen na příliš „nízké“ úrovni. Konceptuální modely jsou řešeny v podobě grafické notace *ER diagramů* nebo *diagramů tříd*.

Na základě konceptuálního modelu je vytvořen logický (známý také jako technologický nebo databázový) model. Logický model zohledňuje technologické pojetí zvolené architektury a její organizaci dat (relační, objektová). Logický model se vztahuje již ke konkrétnímu databázovému modelu. Na této úrovni nejsou zatím řešena implementační specifika.

V poslední fázi vývoje informačního systému je specifikováno, jak budou data fyzicky uloženy a probíhá také vývoj aplikačních programů. Tato koncová fáze vývoje se nazývá *fyzická*.



Obrázek 7 Fáze vývoje databázového informačního systému

Na předchozím ilustračním obrázku je patrné, že po identifikaci požadavků následuje analýza zvoleného problému a na jejím základě je vytvořen návrh databázového modelu, který je v závěrečné fázi implementován.

3.2 Strukturální přístup

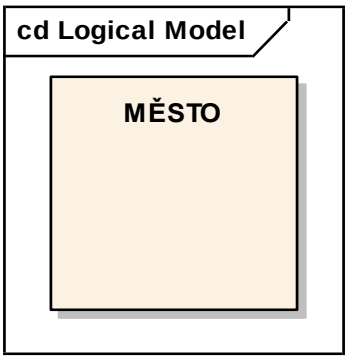
3.2.1 ER diagram

ER (entity relationship) diagram je prostředek pro grafické vyjádření popisu *entit* a *vztahů* mezi nimi. *ERA* (entity relationship attributes) je rozšířením ER diagramu o popis atributů. ER model je nejčastěji používanou technikou pro vytvoření strukturálního konceptuálního modelu (viz. 3.1).

Autorem ER diagramu ve své původní podobě je P. P. Chen. ER diagram je možné vyjádřit v několika mírně odlišných podobách. Jsou jimi například původní Chenova notace, Binární (Oracle) notace, Crowsfeet, IDEF1X, UML a celá řada dalších [17]. Výběr dané notace je čistě na volbě analytika. Pro přehlednost a srozumitelnost návrhu je vhodné držet se zvolené notace po celou dobu analytických prací. V této práci se budeme držet notace vycházející z jazyka UML.

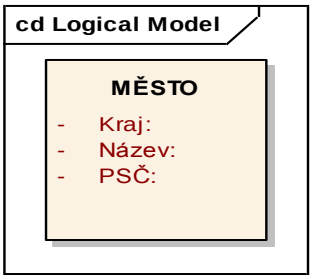
ER diagram je tvořen několika základními konstrukty. Tyto primitiva ER diagramů budou nyní popsány.

- **Entita** představuje určitý objekt z reálného světa o, kterém chceme zaznamenat údaje do databáze. Entity mají tu vlastnost, že jsou od sebe navzájem jednoznačně rozlišitelné pomocí svého identifikátoru (viz. 1.1). Na základě své podobnosti jsou entity sdružovány do množin nazývaných *entitní typy*. Entity stejného entitního typu disponují stejnými vlastnostmi (atributy). Entity Liberec a Jablonec mohou tedy například představovat výskyt entitního typu město.



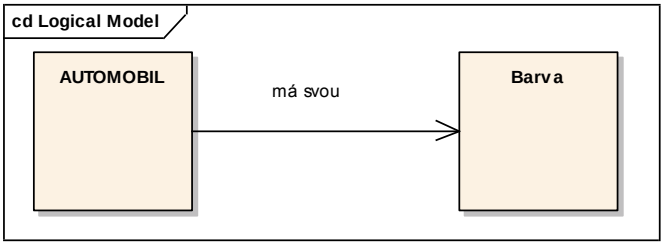
Obrázek 8 Grafické vyjádření entitního typu v ER modelu

- **Atributy entity** jsou hodnoty určující její vlastnosti. Jinými slovy se jedná o datové položky, které entitu charakterizují. Atributy mohou být následujících typů:
  - **Jednoduché** atributy nejsou dále dělitelné. Takovými atributy jsou například jméno nebo příjmení.
  - **Složené** atributy jsou takové, které se dají dále dekomponovat na menší části. Typickým příkladem složeného atributu je adresa, která se skládá z ulice, popisného čísla a města.
  - **Jednohodnotové** atributy.
  - **Vícehodnotové** atributy, které mohou obsahovat více hodnot. Příkladem může být atribut telefon, který se může skládat z více telefonních čísel.



Obrázek 9 Entitní typ se 3 atributy

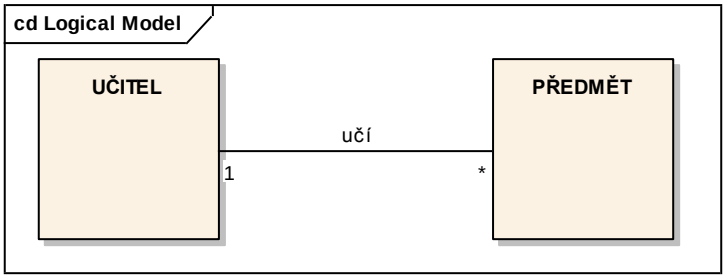
- **Vztahy** vyjadřují vazby, které mezi sebou entity vzájemně mají. Vztahy mohou být mezi 2 (binární) nebo více entitami (n-ární). Vztah je vyjádřen svým jménem (slovesným popisem), vystihujícím jeho podstatu. Vztah mezi entitami může být dále jednosměrný (slovesný popis směřující od entity1 k entitě2), nebo obousměrný (navíc ještě od entity2 k entitě1). Vztah může mít své vlastní atributy nazývané atributy vztahu.



Obrázek 10 Jednosměrný binární vztah mezi entitami

- **Kardinalita vztahu** je typem integritního omezení, které udává, jaký může být minimální a maximální počet výskytů entit v daném vztahu. Pro kardinalitu vztahu mohou nastat následující obecné případy:
  - **1:1** je vztahem, při kterém jedné entitě odpovídá právě 1 entita druhá.
  - **1:N** je vztahem, kdy jedné entitě odpovídá žádná nebo více entit druhého typu. Entitě druhého typu pak odpovídá právě 1 jedna entita typu prvního.
  - **M:N** kardinalita vyjadřuje, že entitě typu 1 může odpovídat více entit typu 2. Stejně tak entitě typu 2 může odpovídat více entit typu 1.

V prostředí modelovacích nástrojů bývá často vztah na straně mnoho vyjádřen symbolem * (obr. 11) nebo také symbolem „vidličky“.

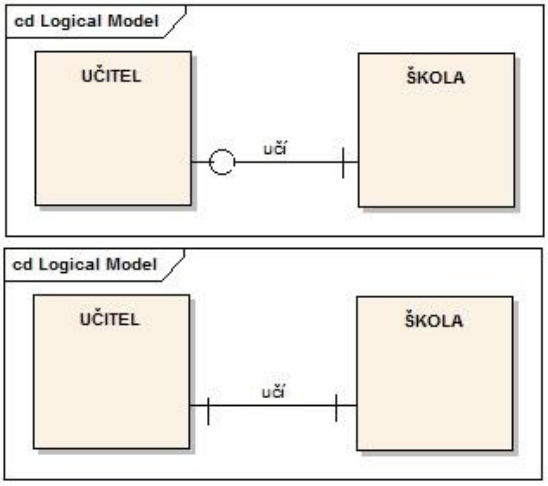


Obrázek 11 Kardinalita 1:N

- **Členství ve vztahu** je dalším z integritních omezení realizovaných v ER diagramu. Členství ve vztahu (známé také jako „parcialita vztahu“) udává, jestli má entita povinnost být do vztahu zapojena. Členství v binárním vztahu může být povinné (totální) nebo nepovinné (parciální). Z uvedeného vyplývá, že členství ve vztahu je buď totální, dále při nepovinném členství jedné z entit parciální zleva nebo zprava, případně parciální oboustranně.

Na (obr. 12) je zachyceno vyjádření parciality vztahu v ER diagramu. V prvním případě entita učitel je v totálním členství ve vztahu. Entita škola má naopak členství ve vztahu nepovinné (parciální). Entita učitel tedy musí učit ve škole, ale škola nemusí mít žádného učitele.

V druhém případě jsou obě entity v povinném členství. Z čehož plyne, že entita učitel musí učit ve škole a entita škola musí mít učitele.

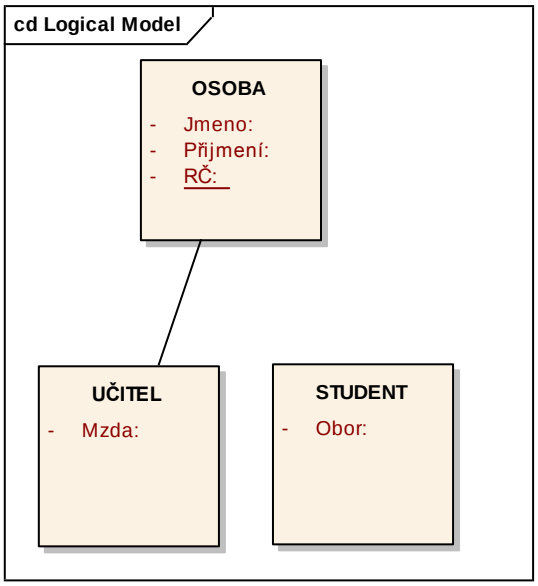


Obrázek 12 Členství ve vztahu v ER diagramu

3.2.2 Rozšíření ER diagramu

ER diagram má některé doplňující konstrukty, které jsou rozšířením jeho základní podoby. Jedná se o rozlišení entitních typů na *silné* a *slabé* a *ISA* hierarchii.

- **Silný** entitní typ je takový, který je jednoznačně identifikovatelný pomocí svých vlastních atributů.
- **Slabý** entitní typ není možné jednoznačně identifikovat podle jeho vlastních atributů. Pro jednoznačnou identifikaci je nutné použít atribut jiného entitního typu. Slabý entitní typ je tedy existenčně závislý na jiném entitním typu (identifikační vlastník). S tímto entitním typem je slabá entita v takzvaném identifikačním vztahu.
- **ISA** hierarchie je rozšířením ER diagramu o možnost vyjádření dědičnosti (viz. 1.2) využívané v OOP. ISA hierarchie přináší do ERD diagramu možnost definice rodičovských entitních množin a jejich potomků. Entitní typ, který je potomkem, bude typem entity rodičovské. Potomek bude mít navíc své speciální atributy. Podoba ISA hierarchie v ER diagramu je na následujícím schématu.



Obrázek 13 ISA hierarchie

3.2.3 DFD

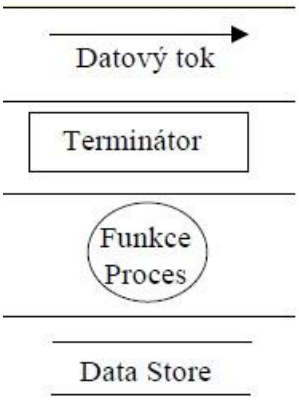
DFD (Data flow diagram) je nástrojem pro provedení funkční analýzy databázového informačního systému. Slouží k identifikaci systémových funkcí a událostí. Stejně jako ERD má i DFD několik podob svých mírně odlišných notací. Konstrukty pro vyjádření DFD jsou:

- **Proces** je funkce, která vstupní data transformuje na výstupní.
- **Terminátor** představuje externí prvky z vně systému. Tyto prvky komunikují se systémem a jsou zdrojem nebo cílem datových toků.
- **Datový tok** označuje směr přenosu dat. Má svůj název a směr.
- **Data store** představuje „datový sklad“. Je pasivním prvkem, který uchovává informace.

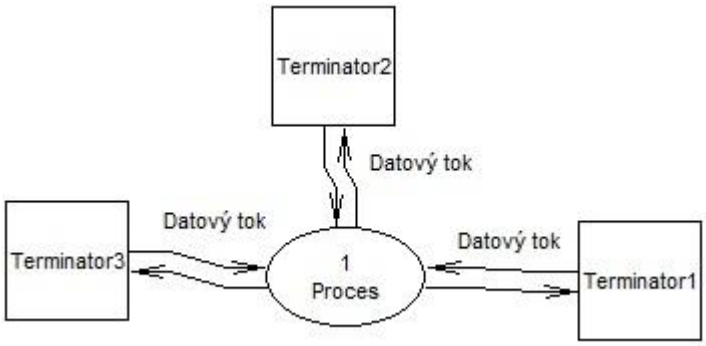
DFD má několik svých úrovní. První úrovní DFD je *kontextový diagram*. Kontextový diagram definuje, jak bude systém komunikovat s okolím světem a jak je do něho zařazen. V kontextovém diagramu jsou zachyceny všechny terminátory komunikující se systémem. Vznikne přehledný obecný popis vstupů a výstupů systému.

Definováním seznamu systémových událostí vznikne upřesněný kontextový diagram. Datové toky dostanou svou konkrétní podobu a diagram již není realizován v tak obecné rovině.

Diagram je dále možné rozkládat do dalších úrovní rozkladu procesu. Pro přehlednost je doporučováno rozkládat maximálně do 3. úrovně.



Obrázek 14 Stavební prvky DFD



Obrázek 15 Kontextový diagram

Při tvorbě DFD je nutné dbát zvýšenou pozornost zejména na následující konstrukty:

- Data store musí mít vstup i výstup. Chybou je pokud se z něj pouze čte nebo se do něj jen zapisuje.
- Vyvarovat se dále musíme funkcí, které mají pouze výstupy nebo naopak pouze vstupy.

Obrázek 16 Upřesněný kontextový diagram

3.3 UML

UML (Unified modeling language) je univerzálním modelovacím jazykem používaným především pro objektovou analýzu a návrh. Jazyk je standardizovaný a průmyslově uznávaný [18].

UML není prostředkem omezeným pouze pro návrh databázových systémů. Zahrnuje v sobě celou kolekci různorodých metod aplikovatelných na popis vývoje informačních systémů jako celku. Pomocí diagramů umožňuje popsat statickou i dynamickou vizualizaci systému. Každý diagram tak zachycuje specifický pohled na vytvářený systém. Jeho aktuální verzí je UML 2.0.

V následující části budou, pospány techniky UML, které jsou podstatné pro analýzu databázových systémů [19].

3.3.1 Případy užití (Use case)

Případy užití jsou vyjádřeny pomocí diagramů, které popisují funkčnost budoucího systému. Definují jednoznačně, co bude budoucí systém obsahovat a jaké bude mít chování. Popisují možné interakce mezi uživateli a systémem.

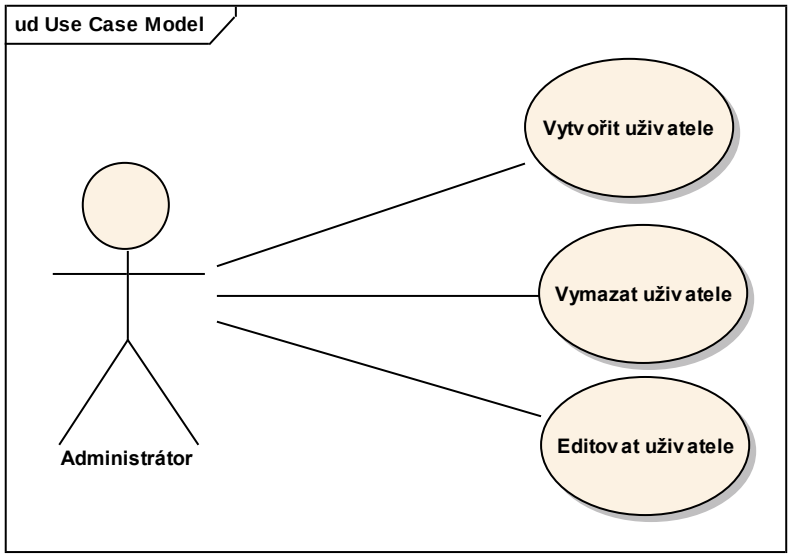
Pro tvorbu případů užití jsou k dispozici následující konstrukty:

- **Aktér** představuje externí objekt, který je v interakci se systémem. Typickým příkladem aktéra je uživatel komunikující se systémem. Aktérem může být i jiný systém. Role popisuje, jak aktér vůči systému vystupuje. Aktér může vystupovat například v roli vedoucího nebo řadového zaměstnance. Všechny systémové akce jsou spouštěny aktéry.
- **Případ užití** je sadou scénářů, které vedou ke společnému cíli.

Na (obr. 17) je znázorněn příklad aktéra v roli systémového administrátora. Administrátor může provádět editaci, mazání a vytváření uživatelů. Tyto úkony představují jednotlivé případy užití. Každý z těchto případů užití má svůj *scénář*, popisující interakci administrátora a systému. Například scénář pro vytvoření uživatele by mohl být následující:

- **Administrátor** spustí volbu vytvořit uživatele v menu systému.
- **Systém** zobrazí příslušný formulář.
- **Administrátor** vyplní formulář a potvrdí volbu vytvořit.
- **Systém** vytvoří uživatele a vrátí se do menu systému.

Takto by mohl vypadat základní scénář. K základnímu scénáři by pak mohl existovat jeden nebo více alternativních scénářů. Tyto alternativní scénáře by ale stejně jako scénář základní vedli ke společnému cíli vytvoření uživatele.



Obrázek 17 Grafická podoba případu užití

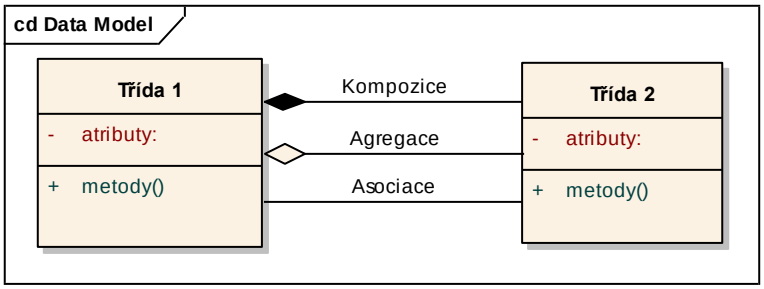
3.3.2 Diagram tříd

Diagram tříd je nástrojem pro statický popis objektově orientovaného systému. Entity jsou vyjádřeny jako třídy s atributy a metodami. Diagram tříd dále zachycuje vazby, které mezi sebou třídy mají.

Základním stavebním prvkem diagramu tříd je samotná třída (1.2). Třída má své atributy a metody. Mezi třídami mohou nastat následující typy vazeb:

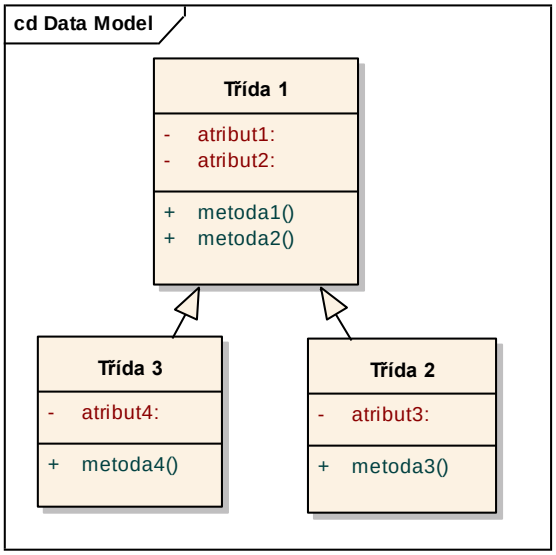
- **Agregace,** vyjadřuje vztah, při kterém je jedna třída částí třídy druhé. Objekt vzniklý z nadřazené třídy využívá schopnosti objektu podřízeného. Nadřazený objekt by měl být zodpovědný za vznik a zánik objektu podřízeného.
- **Kompozice** je speciálním typem agregace. Kompozice vyjadřuje, že existence podřízeného objektu není samostatně možná bez objektu nadřazeného.
- **Asociace** představuje rovnoprávný vztah mezi třídami. Mohou mít určený směr, jinak sou chápány jako obousměrné. Asociace mají stejně jako vztahy mezi entitami v ER modelu svou násobnost. Speciálním

typem asociace je *reflexivní asociace*. Vyjadřuje, že třída je asociována sama na sebe. Objekt vzniklý z této třídy tedy bude mít v sobě odkaz na další objekty z té samé třídy.



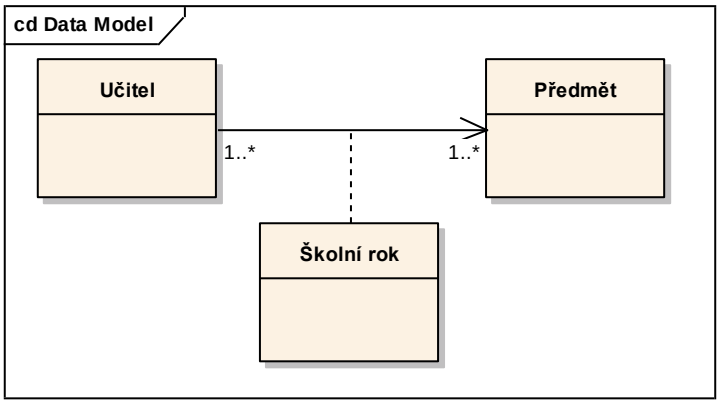
Obrázek 18 Třídy a vazby v diagramu tříd

Dědičnost (1.2) v diagramu tříd je zachycena na následujícím obrázku. Třída 1 představuje rodičovskou třídu. Třídy 2 a 3 jsou jejími potomky. Třída 2 má svůj atribut3 a metodu3. Od rodičovské třídy ještě navíc „zdědí“ atributy1 a 2 a metody 1 a 2. To samé platí i pro třídu 3.



Obrázek 19 Dědičnost v diagramu tříd

Asociační třída se používá při modelování vztahu M:N. Asociační třída má stejné vlastnosti jako ostatní třídy. Na (obr. 20) je znázorněna asociační třída Školní rok. Asociační třída vznikla na základě identifikace vztahu, že učitel může učit více předmětů a předmět může být vyučován více učiteli.



Obrázek 20 Asociační třída

3.3.3 Některé další UML techniky

Jak již bylo řečeno UML je komplexní jazyk pro popis celého informačního systému. Cílem této práce je popis analýzy a návrhu především databázové části informačního systému. Pro tento popis nejsou všechny UML techniky vhodné. Pro ucelený přehled bude uveden výčet dalších technik jazyka UML. Tyto techniky pak najdou své uplatnění především při popisu aplikační části informačního systému.

- **Model objektové spolupráce.**
  - **Sekvenční diagram**, popisuje stavy jednotlivých objektů v čase.
  - **Diagram objektové spolupráce**, popisuje kooperaci mezi objekty.
- **Stavový diagram** popisuje všechny stavy, kterých může objekt v systému nabývat**.**
- **Diagram aktivit** je obdobou stavového diagramu**.**

3.4 Převod konceptuálního modelu na logický

3.4.1 Mapování ER diagramu do relační databáze

Při mapování konceptuálního modelu je nutné zachovat integritní omezení modelu. Integritní omezení jsou na relační úrovni reprezentovány především primárními

a cizími klíči. Atributy entity vyžadující hodnotu jsou na relační straně realizovány nastavením hodnoty not null při definici atributů.

Entitní typ mapujeme na relační schéma tabulky. Název tabulky, stejně jako její atributy, bude odpovídat názvu a atributům entitního typu. V konceptuálním modelu zpravidla nebývají definovány domény atributů, proto je v tuto chvíli vhodné je definovat [22].

Pokud je tabulka mapována ze silného entitního typu, bude jejím primárním klíčem jeho identifikátor. Při mapování slabého entitního typu je do nově vzniklé tabulky nutné přidat ještě atribut, který je primárním klíčem identifikačního vlastníka. Z tohoto atributu se rovněž stane primární klíč.

Následuje přehled způsobů mapování možných vztahů.

- **1:1 (obě entity povinné členství)** tento vztah vede ke sloučení entit do jediné tabulky.
  - Primárním klíčem tabulky se stane libovolný identifikátor jedné z entit nebo budou použity oba.
- **1:1 (jedna entita nepovinné členství)** je vztahem, kdy máme entitu, která existuje nezávisle na entitě druhé.
  - Z nezávislé entity bude vytvořena jí odpovídající tabulka.
  - Dále z entity, která je v povinném členství, vznikne tabulka obsahující navíc cizí klíč odkazující na nezávislou tabulku.
- **1:1 (obě entity nepovinné členství)** vztah vede ke vzniku dvou entitních tabulek a navíc další tabulky vazební.
  - Vazební tabulka bude obsahovat atributy, které budou primárními a zároveň cizími klíči odkazujícími do tabulek entitních.
- **1:N** (**povinná účast na straně více**) je opět vztahem, kdy máme entitu, která existuje nezávisle na entitě druhé.
  - Vztah je realizován přidáním cizího klíče do tabulky vzniklé z entity na straně více.
- **1:N (nepovinná účast na straně více)** vzniknou opět dvě tabulky entitní a jedna vazební.

- Vazební tabulka bude obsahovat atributy, které budou cizími klíči odkazujícími na tabulky entitní.
- Jeden z atributů bude zároveň klíčem primárním.

➢ **M:N** je vztahem mezi entitami, který vždy vede ke vzniku další vazební tabulky.

- Vazební tabulka bude mít svůj primární klíč složen z cizích klíčů odkazujících na entitní tabulky.
- Může obsahovat další atributy.

Mapování ISA hierarchie, pokud se vyskytne v ER diagramu, je možné realizovat několika odlišnými způsoby. Není k dispozici žádné obecné řešení. Mapování ISA hierarchie je tak především závislé na rozhodnutí analytika a na vlastnostech entit. Možné způsoby mapování ISA hierarchie do relačního modelu jsou následující:

➢ **Mapování 1:1** je způsobem, kdy z každého entitního typu vznikne samostatná tabulka. Primární klíč z rodičovské entity je cizím a zároveň primárním klíčem v entitách, které jsou potomky.

➢ **Zahrnutí do rodičovské entity**, atributy potomků migrují do rodičovské entity a vzniká tak jediná relační tabulka. Atributy potomků budou ve výsledné tabulce volitelné. Tento způsob je vhodný, pokud je v ISA hierarchii malý počet entit potomků s omezeným počtem vlastních atributů.

➢ **Rozpuštění rodičovské entity**, při tomto způsobu mapování jsou atributy rodičovské entity zahrnuty do jejích potomků. Tuto variantu lze doporučit v případě, že existuje mnoho speciálních entit s mnoha vlastními atributy.

3.4.2 Objektově relační mapování

V objektově relačním mapování pro nás bude výchozí bod představovat diagram tříd (3.3.2). Diagram tříd se mapuje do relačního modelu velmi podobně, nikoliv však ekvivalentně, jako ER diagram. Metody tříd při mapování na relační tabulky neuvažujeme.

Třídu mapujeme na relační schéma tabulky. Název tabulky, stejně jako její atributy, bude odpovídat názvu a atributům třídy. Instance třídy budou odpovídat záznamům tabulky. Z atributů musí být dále zvolen primární klíč tabulky (1.1).

Asociace jsou v podstatě mapovány stejným způsobem jako vazby v ER diagramu. Kritériem pro mapování je násobnost asociací. Na agregaci a kompozici se při mapování do relačního modelu pohlíží jako na běžnou asociaci.

Mapování dědičnosti odpovídá mapování ISA hierarchie u ERD diagramu.

3.4.3 Převod ER diagramu na diagram tříd

Pokud bychom transformovali ER diagram do diagramu tříd, jednalo by se v podstatě o převod 1:1.

- **Entity** ⟶ **Třídy.**
- **Vazby** ⟶ **Asociace.**
- **ISA** ⟶ **Dědičnost.**

3.5 Relační vs. objektová analýza a návrh

Doposud byly v této práci a především v této kapitole popsány používané metody analýzy a návrhu. Nebylo zatím ale jasně definováno, jaký mají konkrétní význam pro analýzu a návrh studovaných databázových modelů. Tato podkapitola by měla seznámit čtenáře s tím, jaké metody analýzy a návrhu je pro konkrétní databázový model vhodné využít.

3.5.1 Cílová databáze relační

Objasněme si nyní, jak je možné přistoupit k analýze, pokud je od samého počátku jasné, že cílovým technologickým prostředím bude relační databáze. Cílové databázové prostředí hraje v rozhodování jaký způsob analýzy zvolit klíčovou roli. Nicméně existují i další okolnosti, které mají na zvolený způsob analýzy svůj nezanedbatelný vliv.

Nejprve si shrňme obecný postup při návrhu relačního databázového informačního systému.

1. **Vytvoří se konceptuální model** (3.1)**.**
2. **Konceptuální model je transformován do relačního** (3.4)**.**
3. **Provede se datová normalizace do požadované normální formy** (1.1)**.**
4. **Provede se fyzická implementace do konkrétního relačního SŘBD.**

Konceptuální model můžeme realizovat, jak již bylo řečeno, pomocí ER diagramu nebo diagramu tříd. V původním strukturálním přístupu nebylo nutné tento výběr řešit. Pokud bychom tedy prováděli analýzu relačního databázového informačního systému, založeném na klasickém strukturálním přístupu k programování, modelujeme pomocí ER diagramu. Využití třídního diagramu v tomto případě nemá žádný přínos oproti ER diagramu.

Pokud by aplikační část, jako dnes ve většině aplikací, byla řešena pomocí OOP, není rozhodování již tak jednoznačné jako při strukturálním přístupu. ER diagram je navržen původně přímo pro tvorbu konceptuálního modelu následně transformovaného do relační databáze. Poskytuje tedy veškeré potřebně konstrukty. Mnoho návrhářů databází má ER diagram zažitý a nemá důvod uchýlit se při návrhu databázové části k modelu tříd. Je tedy možné zvolit variantu, že konceptuální model databázové části bude řešen pomocí ER diagramu a aplikační část informačního systému pomocí diagramu tříd.

Druhou variantou je realizovat jak databázovou tak aplikační část pomocí UML diagramu tříd.

Není jednoznačně možné vyzdvihnout a doporučit jeden z přístupů. Rozhodnutí stojí na konkrétním analytikovi, který je obeznámen s cílovým prostředím.

Aplikační část	Konceptuální model relační databáze	Doporučení
Strukturální přístup	ER diagram	Vhodné
Strukturální přístup	Diagram tříd	Nevhodné
OOP	Diagram tříd	Vhodné
OOP	ER diagram	Vhodné

Obrázek 21 Vhodnost použití konceptuálního modelu RDB ve vztahu k aplikační části systému

3.5.2 Cílová databáze objektová

Objasněme si nyní, jak je možné přistoupit k analýze, pokud je od samého počátku jasné, že cílovým technologickým prostředím bude objektová databáze.

K návrhu modelu objektově orientované databáze lze jednoznačně doporučit diagram tříd. Diagram tříd je ideálním prostředkem k vytvoření konceptuálního modelu vedoucím na objektovou databázi. Velkým přínosem je, že konceptuální model databáze splývá s modelem logickým. Není tedy nutná žádná složitá transformace.

Návrh celého objektově orientovaného informačního systému se použitím diagramu tříd pro datovou i aplikační část stává přehledným a srozumitelným. Alternativně lze databázovou část modelovat i pomocí ER diagramu, ale diagram tříd je daleko přirozenějším řešením.

Objektově orientované databáze nemají doposud jasně daná pravidla a metodiku pro svůj návrh. Existují již první studie a odborné publikace zabývající se problematikou normalizace v objektových databázích. Všeobecně uznávaná pravidla jak strukturovat třídy a z nich vzniklé objekty pro optimální návrh objektové databáze doposud nejsou ustanoveny.

Přínosem může být využití návrhových vzorů. Další možností je prosté převzetí tříd a vztahů tak jak jsou navrženy pro aplikaci spolupracující s databází. Stále však platí, že návrh modelu objektové databáze je odkázán především na zkušenost a přístup analytika.

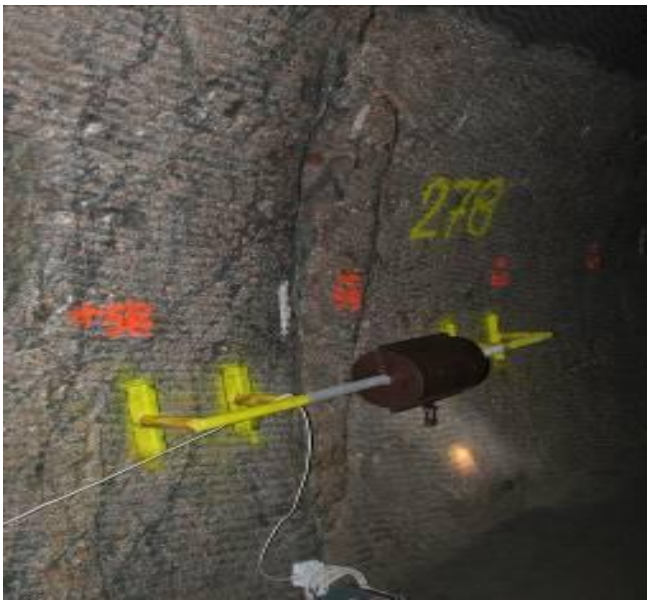
4 Praktická část

Tato část práce bude věnována zdokumentování postupu při řešení konkrétního databázového problému

Bedřichovský tunel (nacházející se v Severních Čechách v Jizerských horách) přivádí pitnou vodu z přehrady Josefův Důl do úpravny vody v Bedřichově. Je vyražen v granitu. Jeho nejhlubší místo se nachází až 140 metrů pod povrchem a jeho délka je více než 2 500 m. Vnitřní profil tunelu je kruhový o průměru 3,3 m. Je v něm umístěno vodovodní potrubí s průměrem cca 80 cm.

Obrázek 22 Bedřichovský tunel

Technická univerzita v Liberci, především pak členové výzkumného centra ARTEC, se podílí na geologickém výzkumu v Bedřichovském tunelu. Jedná se především o sledování dynamiky chování hornin, proudění podzemních vod a následné modelování těchto procesů. K tomuto účelu je nutné provádět celou řadu měření a sběr dat. V tunelu se nachází měřící zařízení a senzory. K optimalizaci práce bylo nezbytné navrhnout přenosový řetězec, který by umožnil automatické měření zvolených veličin a jejich následný přenos. Tento systém pro automatický sběr a přenos dat je vyvíjen kolektivem řešitelů Technické univerzity.



Obrázek 23 Měřící zařízení v Bedřichovském tunelu

Zadáním praktické části je namodelovat chování tohoto systému a to především ve vztahu k databázové části. Dále navrhnout databázové modely pro uložení dat v relační a objektově orientované databázi.

4.1 Požadavky

Pro potřeby ukládání perzistentních dat do databáze byly vzneseny následující požadavky na databázovou část.

V databázi by se měli uchovávat data o měřících zařízeních. Údaj o měřícím zařízení by měl obsahovat název zařízení a jeho základní charakteristiku.

Každé měřící zařízení má několik senzorů. U senzorů je třeba evidovat měřená data, jejich velikost a čas kdy byla naměřena. Senzory mohou být od měřících zařízení v různých vzdálenostech. Senzory tedy budou mít svou konkrétní polohu. Senzor může být umístěn přímo v tunelu nebo na povrchu. V tunelu se bude evidovat jeho vzdálenost od paty tunelu. Na povrchu bude lokalizován dle svých GPS souřadnic zeměpisné šířky a zeměpisné délky.

Dále je nutné uchovávat údaje o typu měření, měřených veličinách (např. průtok, teplota, pH) a jednotkách měřených veličin.

Z požadavků vyplývá, že je třeba uchovávat informace o:

- **Měřícím zařízení.**
- **Měřené veličině.**
- **Typu měření.**
- **Poloze měření.**
- **Senzorech a naměřených datech.**

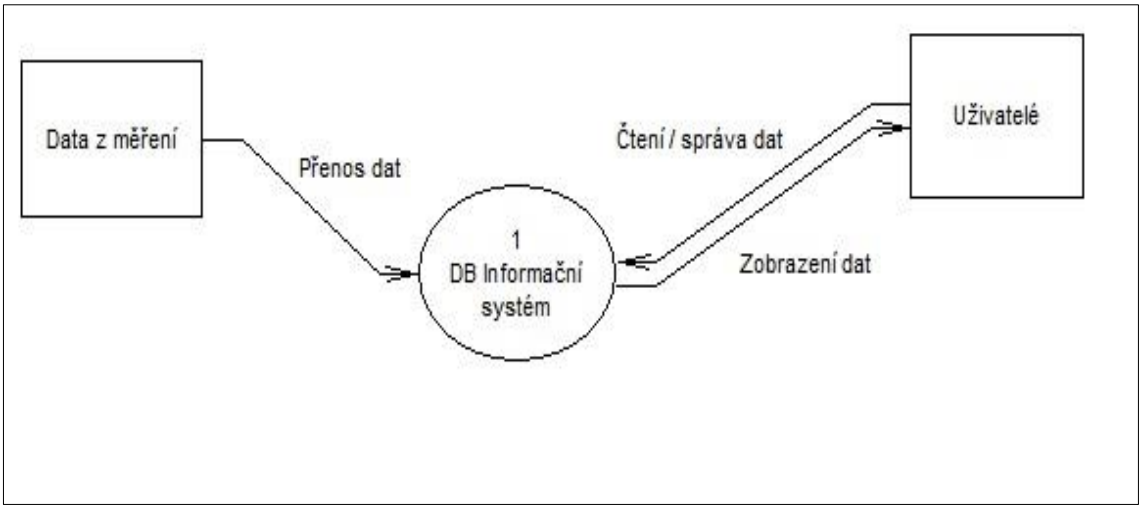
Požadavky na chování systému vůči databázové části a uživatelům jsou následující:

- Systém umožní rozlišit dvě skupiny uživatelů. První skupinou jsou členové řešitelského týmu, druhou část tvoří externí uživatelé (studenti, zástupci firem a jiní). Systém při přístupu ověří identitu uživatele a přidělí mu uživatelská práva.
- Externí uživatelé budou mít možnost prohlížet a zobrazovat přístupná data z měření. Členové řešitelského týmu budou mít jednak možnost zobrazovat data, dále pak budou moci provádět úpravu a manipulaci s daty.
- Přenosový řetězec bude moci po přístupu do databázové části automaticky ukládat data.
- Data před uložením do databáze musí projít kontrolním mechanismem.
- Před zobrazením dat uživatelům je nutné provést jejich transformaci do podoby vhodné pro prohlížení.

4.2 Návrh relačního databázového modelu

Na základě vstupních požadavků na chování systému vůči databázové části a uživatelům byly vytvořeny diagramy datových toků. Tyto DFD diagramy tak přehlednou formou umožní popsat systémové funkce a události.

Vstupy a výstupy databázové části systému jsou zachyceny na nejvyšší úrovni v tzv. kontextovém diagramu (obr. 24). Do systému přistupují uživatelé, kterým systém poskytuje přístup k datům. Data z měření jsou do systému automaticky přenášena pomocí přenosového řetězce.



Obrázek 24 Kontextový diagram databázového informačního systému

Detailněji je chování databázové části informačního systému zdokumentováno v rozkladu kontextového diagramu do další úrovně (obr. 25).

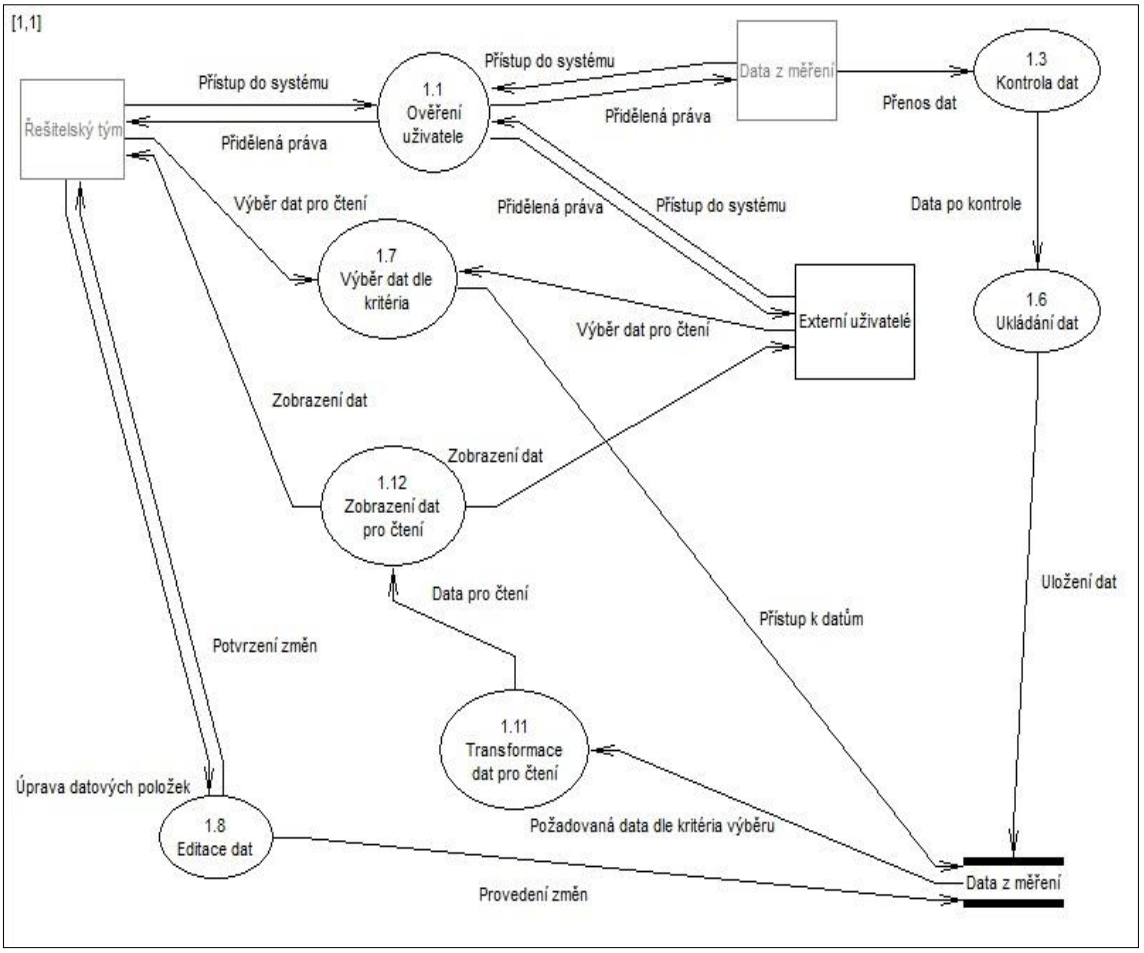
Z diagramu je patrné dělení uživatelů na členy řešitelského kolektivu a uživatele externí. Při přístupu do systému musí uživatelé projít procesem ověření. Požadavek na přístup od uživatele zpracuje a vyhodnotí systémový proces Ověření uživatele. Proces Ověření uživatele přidělí uživateli konkrétní uživatelská práva. Procesem ověření musí projít i přenosový řetězec, zajišťující přenos dat z měření.

Externí uživatelé mají právo data pouze zobrazovat. Externí uživatelé vyberou, dle parametrů jaká data chtějí zobrazit. Přístup k těmto datům pro zobrazení zajišťuje systémový proces Výběr dat dle kritéria.

Vybraná data přetransformuje do podoby vhodné pro čtení systémový proces Transformace dat pro čtení. Zobrazení transformovaných dat uživatelům zajistí systémový proces Zobrazení dat pro čtení.

Členům řešitelského kolektivu, je navíc zpřístupněna možnost úpravy dat. Zpracování požadavku na úpravu dat zajišťuje systémový proces Editace dat, který zároveň informuje uživatele o výsledku provedené operace.

Systémový proces Kontrola dat zkontroluje validitu dat a předá korektní data systémovému procesu Ukládání dat.

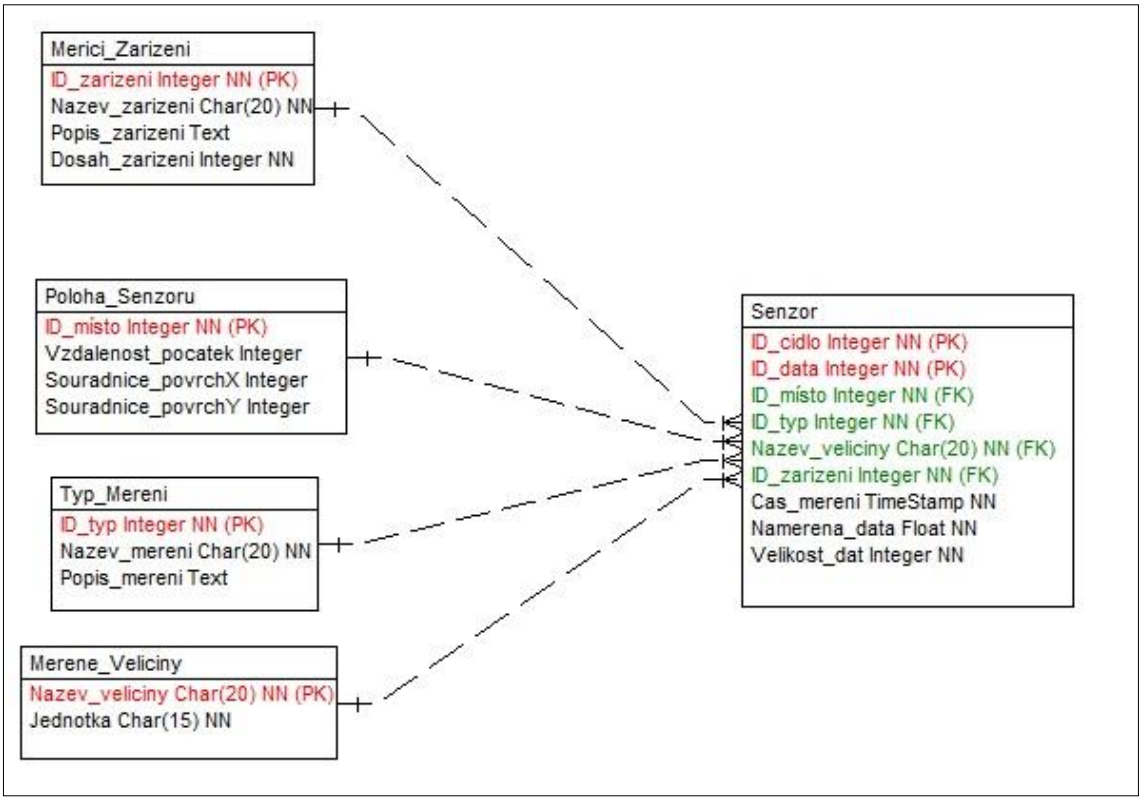


Obrázek 25 DFD diagram další úroveň rozkladu systémových procesů

Model relační databáze (obr. 26) byl navržen tak aby se dalo efektivně dotazovat na data, která budou v systému uchovávána dle definovaných požadavků (4.2).

Vznikly 4 relace, které uchovávají informace o měřících zařízení, poloze senzoru, typu měření a měřených tabulkách. Relační tabulka Senzor, obsahuje cizí klíče, které jsou klíči primárními v těchto 4 tabulkách. Tím je zajištěna referenční integrita dat a lze snadno přistupovat k jednotlivým datům.

Do tabulky Senzor budou zaznamenávána měřená data u konkrétního senzoru a čas měření.



Obrázek 26 Relační model dat

Relační model dat byl naimplementován a otestován v prostředí MS SQL Server 2005. Klient byl realizován v programovacím jazyce Java. Pro spojení byl použit ovladač s knihovnou pro komunikaci sqljdbc4.jar [23].

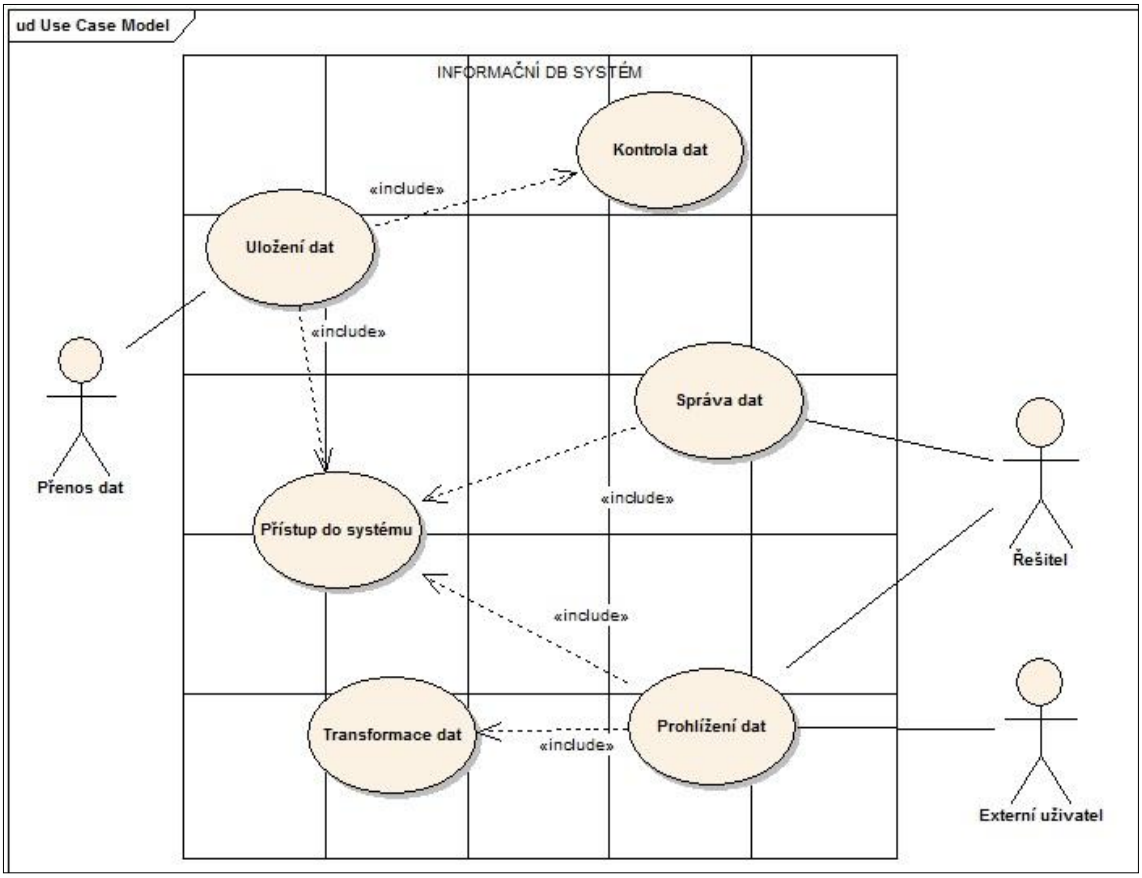
4.3 Návrh objektového databázového modelu

Na základě formulace požadavků na chování systému vůči databázové části a uživatelům byl nejprve při návrhu objektově orientovaného modelu vytvořen diagram případu užití. Případ užití poskytuje ucelený pohled na chování systému a je vhodné ho použít při objektové analýze.

V modelu případu užití (obr. 27) jsou znázorněni 3 aktéři. Aktéři vystupují v rolích přenosového řetězce (přenos dat), řešitele a externího uživatele. Aktéři jsou v interakci se systémem a provádí své případy užití.

Z modelu je patrné, že pro uložení, prohlížení a správu dat je nutné přihlášení do systému. Toto společné chování pro zmíněné případy užití je vyčleněno do samostatného případu užití. Tyto případy užití tedy pro svou realizaci potřebují provést

případ užití Přihlášení do systému. Stejný vztah lze pozorovat i mezi případy užití Prohlížení dat a Transformace dat, resp. Uložení dat a Kontrola dat.



Obrázek 27 Model případu užití

Následuje přehled scénářů jednotlivých případů užití včetně počátečních a koncových stavů systému, před a po jejich provedení.

Název případu užití: **Přístup do systému**

Počáteční podmínka: **Uživatel není přihlášen v systému**

Pořadí	Role	Provedená akce	Typ
1.	Uživatel	Zadá přihlašovací údaje	Základní
1.a	Přenos dat	Provede automatické přihlášení	Alternativní
2.	Systém	Ověří uživatele	Základní
3.	Systém	Přidělí uživatelská práva	Základní

Výstupní stav: **Uživatel (přenos dat) je přihlášen v systému**

Obrázek 28 Scénář případu užití Přístup do systému

Název případu užití: **Prohlížení dat**

Počáteční podmínka: **Uživatel chce prohlížet data**

Pořadí	Role	Provedená akce	Typ
1.	Uživatel	Přihlásí se do systému (viz. Přístup do systému)	Základní
2.	Systém	Zobrazí menu pro prohlížení	Základní
3.	Uživatel	Zadá kritéria pro vyhledávání	Základní
4.	Systém	Provede výběr a transformaci dat (viz. Transformace dat)	Základní
5.	Systém	Zobrazí výpis požadovaných dat	Základní

Výstupní stav: **Systém poskytl uživateli požadovaná data**

Obrázek 29 Scénář případu užití Prohlížení dat

Název případu užití: **Správa dat** Počáteční podmínka: **Uživatel chce upravovat data**			
Pořadí	**Role**	**Provedená akce**	**Typ**
1.	Uživatel	Přihlásí se do systému (viz. Přístup do systému)	Základní
2.	Systém	Pokud je uživatel řešitel – zobrazí menu pro editaci	Základní
3.	Uživatel	Provede úpravy dat	Základní
4.	Systém	Provede změny	Základní
5.	Systém	Potvrdí změny výpisem uživateli a vrátí se do menu pro editaci	Základní
Výstupní stav: **Potvrzení změn, návrat do menu editace**			

Obrázek 30 Scénář případu užití Správa dat

Název případu užití: **Transformace dat** Počáteční podmínka: **Data jsou ve formátu nevhodném pro čtení**			
Pořadí	**Role**	**Provedená akce**	**Typ**
1.	Systém	Přijme vybraná data z databáze	Základní
2.	Systém	Provede transformaci dat do předepsaného tvaru a formátu	Základní
Výstupní stav: **Data jsou v podobě vhodném pro čtení**			

Obrázek 31 Scénář případu užití Transformace dat

Název případu užití: **Uložení dat**

Počáteční podmínka: **Data z měření nejsou uložena v databázi**

Pořadí	Role	Provedená akce	Typ
1.	Přenos dat	Provede automatické přihlášení (viz. Přístup do systému)	Základní
2.	Systém	Provede kontrolu dat (viz. Kontrola dat)	Základní
3.	Systém	Přijme validní data	Základní
4.	Systém	Uloží data do databáze	Základní

Výstupní stav: **Data z měření jsou uložena**

Obrázek 32 Scénář případu užití Uložení dat

Název případu užití: **Kontrola dat**

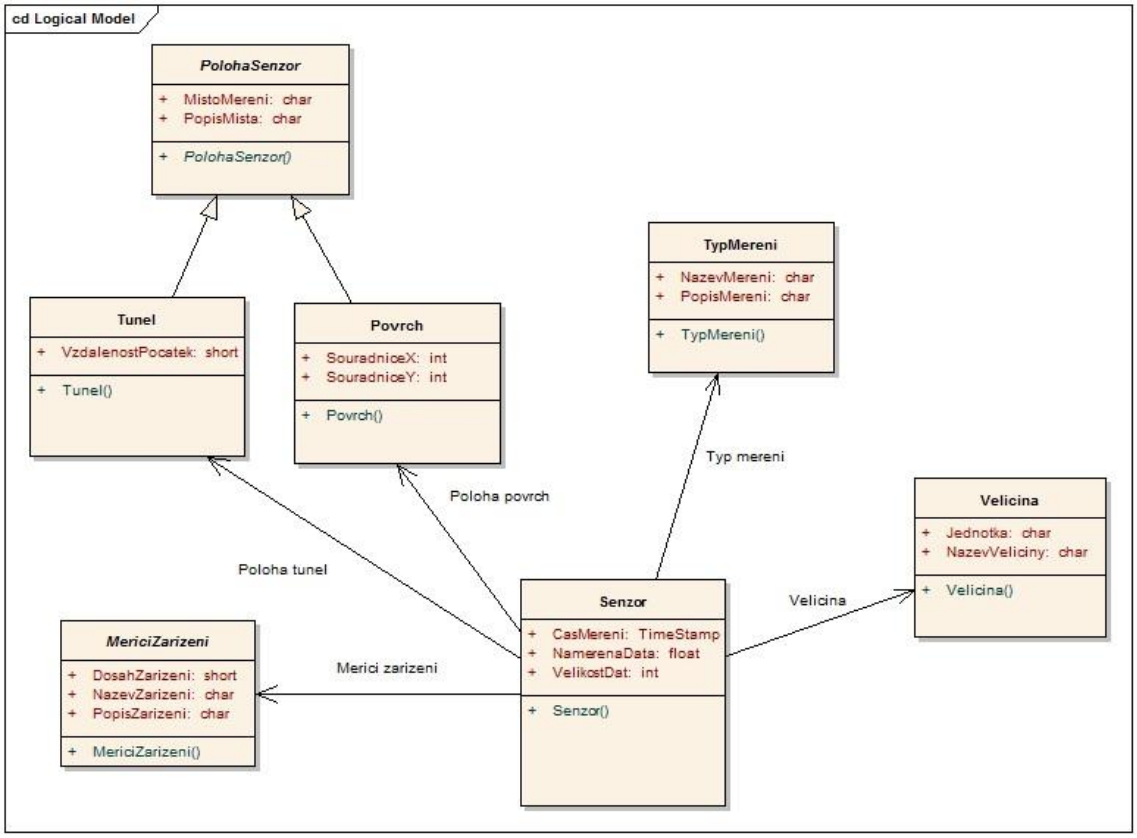
Počáteční podmínka: **Přenos dat přenesl data z měření do systému**

Pořadí	Role	Provedená akce	Typ
1.	Systém	Přijme přenesená data	Základní
2.	Systém	Prochází data a provádí kontrolu	Základní
3.	Systém	Potvrdí správnost dat	Základní
3. a	Systém	Provede opravu dat	Alternativní
3. b	Systém	Potvrdí správnost dat	Alternativní
4.	Systém	Předá data k uložení	Základní

Výstupní stav: **Data neobsahují chyby a jsou předána pro uložení do databáze**

Obrázek 33 Scénář případu užití Kontrola dat

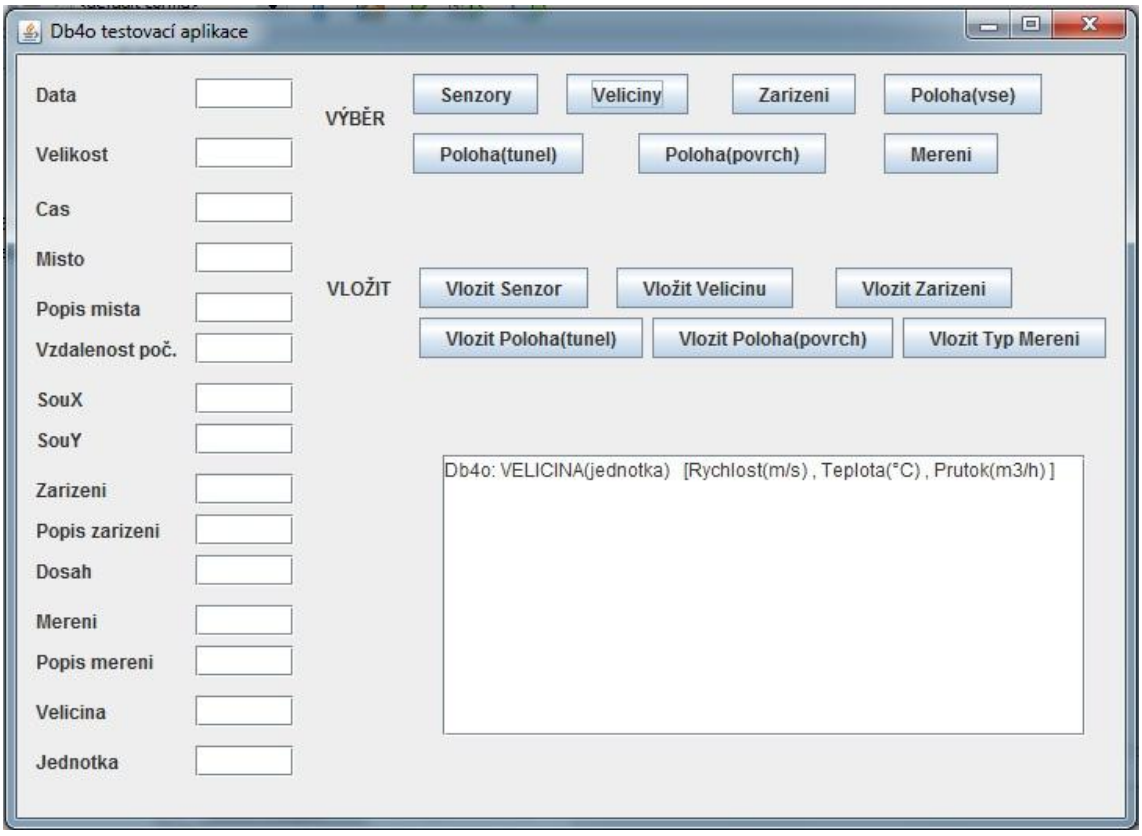
Navržený diagram tříd (obr. 34) pro objektově orientovanou databázi obsahuje abstraktní třídu Poloha Senzor. Od této abstraktní třídy dědí společné atributy specializované třídy Tunel a Povrch. Dalšími třídami v databázovém modelu jsou třídy Typ měření, Měřící zařízení a Veličina. Třída Senzor si pak uchovává reference na objekty vytvořené z těchto tříd.



Obrázek 34 Diagram tříd objektově orientované databáze

Objektově orientovaný model databáze byl naimplementován a otestován na databázi Db4o v režimu embedded. Klient byl realizován v programovacím jazyce Java. Pro spojení byl použit ovladač s knihovnou pro komunikaci db4o-7.12.156.14667–all-java5.jar [24].

Testováno bylo ukládání objektů jak ze třídy senzor tak i objektů vytvářených z ostatních tříd. Dále bylo otestováno získávání dat z objektové databáze. K dotazování na objekty v databázi bylo využito nativního jazyka. Podoba jednoduché testovací aplikace pro objektově orientovanou databázi Db4o je na (obr. 35).



Obrázek 35 Podoba testovací aplikace pro práci s databází Db4o

4.4 Srovnání objektové databáze a relační databáze mapované na objekty v aplikační vrstvě

Pro potřeby základního srovnání dostupné funkcionality objektové databáze a relační databáze mapované na objekty v aplikační vrstvě, bylo provedeno mapování relační databáze pomocí technologie *Hibernate* [25].

Hibernate provádí tzv. objektově relační mapování objektů v Javě do podoby entit relační databáze. Způsob transformace objektů do relační databáze byl definován pomocí externích mapovacích souborů formátu XML. Kromě těchto mapovacích souborů, je možné pomocí technologie Hibernate definovat způsob transformace objektů do relační databáze použitím JavaDoc komentářů nebo anotací.

Základní srovnání obou přístupů:

- Aplikace využívající objektově relační mapování při práci s databází je proti aplikaci pracující s databází objektovou pomalejší. Je to způsobeno nutností transformace dat.

- Pokud využíváme OOP je daleko přirozenější cestou ukládat objekty přímo do objektové databáze k tomuto navržené.
- Objektově relační mapování můžeme využít i bez znalosti jazyka SQL, který je nezbytný pro práci s relační databází.
- Ukládání dat do objektové databáze nevyžaduje téměř žádná nastavení. Objektově relační mapování je nutné nakonfigurovat.

5 Závěr

Diplomová práce se zabývá studiem metodiky analýzy a návrhu objektově orientovaných a relačních databázových systémů. Oba zmíněné modely vyžadují specifický přístup ke svému návrhu. Správně a vhodným způsobem provedená analýza pak může ušetřit spoustu času a v neposlední řadě také finančních prostředků. Práce by měla být tím správným vodítkem jaký způsob analýzy a návrhu zvolit.

Podařilo se splnit všechny dílčí cíle a body zadání. Byla zdokumentována metodika návrhu relačních a objektově orientovaných databázových informačních systémů. Byly představeny nejvýznamnější dostupné objektově orientované databázové systémy. Dále byla provedena analýza a návrhy řešení konkrétního databázového problému. Vytvořené návrhy byly naimplementovány a otestovány ve zvolených databázových prostředích.

6 Literatura

[1] **WELLING, L., THOMSON, L.: MySQL Průvodce základy databázového systému, Brno 2005, ISBN 80-251-0671-3.**

[2] **ŠEDA, M.: Databázové systémy [online], Brno 2002, [cit. 20.-2.-2011]. Dostupné z: http://www.uai.fme.vutbr.cz/~mseda/DBS02_BS.pdf**

[3] **ŠPÁNEK, R.: Databázové systémy [online], Liberec 2010, [cit. 20.-2.-2011]. Dostupné z: http://sirius.mti.tul.cz/~roman.spanek/files/dbs/2011/pr1.pdf**

[4] **ŠIMONOVÁ, S., PANUŠ, J.: Databázové systémy 1 pro kombinovanou formu studia, Pardubice 2007, ISBN 978-80-7194-988-6.**

[5] **FARANA, R.: Tvorba relačních databázových systémů, Ostrava 1999, ISBN80-7078-706-6.**

[6] **MERUNKA, V.: Objektový přístup v databázové technologii [online], 2004, [cit. 23-2-2011]. Dostupné z: http://formular-ekf.vsb.cz/formulare/F01/tsw/getfile.php?prispevekid=797**

[7] **MERUNKA, V: Objektový přístup v databázových systémech, Praha 2002, ISBN 80-213-0882-6.**

[8] **ŠVEC, M.: Objektové databáze [online], 2003, [cit. 25-2-2011]. Dostupné z: http://www.fit.vutbr.cz/study/courses/VPD/public/0203VPD-Svec.pdf**

[9] **GREHAN, R.: ODBMS for RDBMS Users [online], 2008, [cit. 25-2-2011]. Dostupné z: http://www.odbms.org/download/006.03%20Grehan%20ODBMS%20for%20RDBMS%20Users%202005.PDF**

[10] **GREEN, R.: ODBMS architectures an examination of implementations [online], 2008, [cit. 25-2-2011]. Dostupné z: http://www.odbms.org/download/028.01%20Greene%20OODBMS%20Architectures%20September%202006.PDF**

[11] **GREHAN, R.: The database behind the brains [online], 2006, [cit. 23-4-2011]. Dostupné z: http://www.db4o.com/about/productinformation/whitepapers/db4o%20Whitepaper%20-%20The%20Database%20Behind%20the%20Brains.pdf**

[12] **INTERSYSTEMS CORPORATION: Intersystems Caché technology guide [online], 2007, [cit. 23-4-2011]. Dostupné z: http://www.odbms.org/download/CacheTechGuide.pdf**

[13] **KIRSTEN, W., a další: Caché databáze postrelačního typu a tvorba aplikací , Brno 2005, ISBN 80-251-0491-5.**

[14] **ZICARI, R. V.: TechView Product Report: Objectivity/DB [online], 2008, [cit. 23-4-2011]. Dostupné z: http://www.odbms.org/download%5CTechViewObjectivity2010.pdf**

[15] **ZICARI, R. V.: Techview product report: versant object database [online], 2008, [cit. 23-4-2011]. Dostupné z: http://www.odbms.org/download/048.04%20TechView%20Versant.pdf**

[16] **ODMG: Object database vendors [online], 2007, [cit. 23-4-2011]. Dostupné z: http://www.odbms.org/vendors.aspx**

[17] **SONG, Y., MARY, E., PARK, E. K.: A comparative analysis of entity-relationship diagrams, Journal of computer and software engineering, Vol. 3, No. 4, [online], 1995, [cit. 23-4-2011]. Dostupné z: http://www.ischool.drexel.edu/faculty/song/publications/p_Jcse-erd.PDF**

[18] **ARLOW, J., NEUSTADT, I.: UML 2 a unifikovaný proces vývoje aplikací, Brno 2007, ISBN: 978-80-251-1503-9.**

[19] **KANISOVÁ, H., MüLLER, M.: UML srozumitelně, Brno 2004, ISBN 80-251-0231-9.**

[20] **MLÝNKOVÁ, I., NEČASKÝ M.: Databázové systémy [online], 2010, [cit. 23-2-2011]. Dostupné z: http://www.ksi.mff.cuni.cz/~mlynkova/A7B36DBS/slajdy/A7B36DBS_02.pdf**

[21] **KRÁTKÝ, M.: Databázové a informační systémy, verze 20070403, [online], 2006, [cit. 23-2-2011]. Dostupné z: http://www.cs.vsb.cz/kratky/courses/2006-07/tis/download/dbis.pdf**

[22] **VALENTA, M.: DBS – Transformace konceptuálního modelu na relační, [online], 2010, [cit. 23-2-2011]. Dostupné z: https://users.fit.cvut.cz/valenta/doku/lib/exe/fetch.php/bivs/dbs_06_transformace_konceptualniho_shcematu_na_relacni.pdf**

[23] **MISCROSOFT: Using the JDBC Driver [online], 2011, [cit. 15-5-2011]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms378526.aspx**

[24] **DB4O: Db4o-7.4-tutorial-java [online], 2007, [cit. 23-4-2011]. Dostupné z: http://www.db4o.com/about/productinformation/resources/db4o-7.4-tutorial-java.pdf**

[25] **HIBERNATE: Hibernate documentation [online], 2011, [cit. 10-5-2011]. Dostupné z: http://docs.jboss.org/hibernate/core/3.6/quickstart/en-US/html_single/**

[26] **ODMG: Object data management group [online], 2011, [cit. 10-5-2011]. Dostupné z: http://www.odbms.org/odmg/**