

Technická univerzita v Liberci
Hospodářská fakulta

Studijní program: 6209T - Systémové inženýrství a informatika

Studijní obor: M6209 - Manažerská informatika

Objektově relační databázové systémy

Object Relational Database systems

Číslo závěrečné práce: DP-MI-KIN-2004-13

MARTIN ŘEZÁČ

Vedoucí práce: Ing. Vladimíra Zádová (Katedra informatiky)

Konzultant: Ing. Jan Šimek (Unicorn a.s.)

Počet stran: 59

Počet příloh: 0

Datum odevzdání: 5.1.2004

UNIVERZITNÍ KNIHOVNA
TECHNICKÉ UNIVERZITY V LIBERCI



3146086384

ZADÁNÍ DIPLOMOVÉ PRÁCE

pro :

Martin Řezáč

Studijní program :

Systémové inženýrství a informatika (6209T)

Studijní obor č. M 6209

Manažerská informatika

Vedoucí katedry Vám ve smyslu zákona č. 1111/1998 Sb. o vysokých školách a navazujících předpisů určuje tuto diplomovou práci :

Název tématu:

Objektově-relační databázové systémy

Zásady pro vypracování :

1. Srovnání objektových a relačních přístupů.
2. Teoretický koncept objektově-relačních databázových systémů.
3. Situace a zhodnocení objektově-relačních databázových systémů v ČR.

Rozsah diplomové práce: 50-60

(do rozsahu nejsou započítány úvodní listy, přehled literatury a přílohy)

Doporučená literatura:

Halaška, I.: Databázové systémy, 2002, ČVUT

Merunka, V.: Objektový přístup v databázových systémech, 2002, ČZU

Stonebraker, M.: Objektově-relační SŘBD, 2000, BEN

Pokorný, J.: Databázová abeceda, 1998

McClure, S.: Object database vs. object-related database, 1997

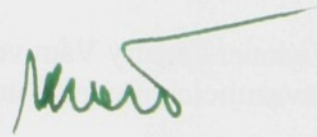
Vedoucí diplomové práce: Ing. Vladimíra Zádová

Odborný konzultant: Ing. Dana Nejedlová

Termín odevzdání diplomové práce: 5.1.2004


Prof. Ing. Jan Ehleman, CSc.
vedoucí katedry




Prof. Ing. Jiří Kraft, CSc.
děkan Hospodářské fakulty

V Liberci dne 31.3.2003

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně s použitím uvedené literatury pod vedením vedoucího a konzultanta. Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména §60 (školní dílo) a §35 (o nevýdělečném užití díla k vnitřní potřebě školy).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé práce a prohlašuji, že souhlasím s případným užitím mé práce (prodej, zapůjčení apod.)

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

Po pěti letech si mohu tuto práci vyžádat v Univerzitní knihovně TU v Liberci, kde je uložena, a tím výše uvedená omezení vůči mé osobě končí.

V Liberci dne: 2. 1. 2004

Podpis: Martin Řezáč

Resumé

V dnešní době jsou databázové systémy nutnou součástí pro dobře fungující podnik či jakoukoliv jinou instituci. Požadavky uživatelů na funkce, objemy dat a výpočetní výkony neustále rostou a každá společnost vyvíjející databázové systémy se snaží nabídnout na trhu ten nejlepší produkt. Tato diplomová práce je věnována databázím, především z pohledu jejich možností a funkčnosti. Aby byl uživatel využívající nějakou databázi spokojen, musí být tato databáze správně definována a navržena. V této práci najdete popis v dnešní době nejvíce žádaného objektově relačního databázového modelu. Tento model kombinuje vlastnosti klasických (relačních) databází s vlastnostmi, které jsou využívány v objektově orientovaných přístupech. Stěžejní část práce je věnována teoretickému konceptu objektově relačního databázového modelu, jehož vlastnosti jsou v drtivé většině demonstrovány na příkladech. Další část textu mapuje současný trh s těmito databázemi s důrazem na novinky a funkce, kterými se produkty odlišují od ostatních.

Abstract

Database systems are at present necessary part for any well-going business or other company. User demands for the functionality, quantity of data and performance of processing data are growing continuously and all companies developing database systems try to offer best product. This thesis focuses on databases primarily from the functionality and applicability point of view. In case the user will be satisfied the database should be defined and designed correctly. Inside this work you will find the description of today most demanded object relational database model. This model aggregates characteristics of classical (relational) database with characteristics used in object oriented programming. Major part of the work focuses on the theoretical concept of the object relational database model, whose characteristics are mostly demonstrated on examples. Next part of the text is mapping current situation on the market with database systems with accentuation on new features and functions differentiating the appropriate product from other products.

Poděkování

Touto cestou bych chtěl poděkovat paní Ing. Vladimíře Zádové nejen za její podnětné připomínky k této práci, ale i za odborné konzultace na téma návrhu databází a především časovou flexibilitu, kterou mi v této záležitosti věnovala.

Dále bych chtěl poděkovat panu Ing. Sergeji Kozlovskemu za jeho konzultace na praktické úrovni konkrétních databázových produktů a za náhled na samotné vytváření databázových systémů.

Obsah

1. ÚVOD	10
1.1. Cíl práce	10
2. OBJEKTOVÝ VS. RELAČNÍ PŘÍSTUP	11
2.1. Úvod do problematiky	11
2.1.1. Historické ohlédnutí	11
2.2. Přístupy k modelování	12
2.2.1. Strukturovaný přístup	13
2.2.1.1. Strukturované metody	13
2.2.1.2. Nástroje pro strukturované metody	14
2.3. Relační databázové systémy	14
2.3.1. Přednosti relačních databází	15
2.3.2. Stinné stránky relačních databází	16
2.4. Objektově orientovaný přístup	17
2.4.1. Objekt vs. relační entita	18
2.4.2. Identita objektu	18
2.4.3. Objektové databázové systémy - OOSŘBD	19
2.5. Srovnání přístupů	21
2.5.1. Silné stránky objektového přístupu	21
2.5.2. Silné stránky relačního přístupu	21
3. OBJEKTOVĚ RELAČNÍ MODEL SŘBD	22
3.1. SQL a OR model	22
3.1.1. Bezpečnost	23
3.1.2. Datové typy v OR modelu	24
3.1.3. Operátory	27
3.2. Plně objektově-relační přístup	28
3.2.1. Dynamické propojení	28
3.2.2. Zpětné volání (callback)	30
3.2.3. Uživatelem definované přístupy k metodám	30
3.2.3.1. Délka datových typů	31
3.3. Komplexní objekty	31
3.3.1. Složené datové typy	31
3.3.2. Sady	32
3.3.3. Odkazy (reference)	33
3.3.4. Manipulace se složenými datovými typy	34
3.3.5. Používání odkazů	37

3.3.6.	Základní typy a komplexní objekty	39
3.4.	Dědičnost	41
3.4.1.	Dědičnost funkcí	43
3.5.	Polymorfismus	46
3.5.1.	Overloading funkcí	47
3.6.	Pravidla	48
3.6.1.1.	Update-update	49
3.6.1.2.	Query-update	50
3.6.1.3.	Update-query	51
3.6.1.4.	Query-query	52
4.	SOUČASNÝ TRH S OBJEKTOVĚ RELAČNÍMI DATABÁZOVÝMI SYSTÉMY	56
4.1.	Přehled databázových platforem	56
4.1.1.	Advantage Ingres Enterprise Relational Database	56
4.1.2.	Oracle 10g	57
4.1.3.	MySQL	58
4.1.4.	Tamino XML Server	59
4.1.5.	MS SQL Server 2000	60
4.1.6.	Interbase	61
4.1.6.1.	Firebird	62
4.1.7.	Informix dynamic server	62
4.1.8.	PostgreSQL	63
4.1.9.	Caché	64
4.1.10.	Sybase Adaptive Server Enterprise	65
4.1.11.	DB2 Universal Database Version 2	66
4.2.	Shrnutí	67
5.	ZÁVĚR	68
6.	POUŽITÁ LITERATURA	69

Vysvětlení zkratk

ACF	Application Context Facility, nástroj pro administrátora databáze
API	Application Program Interface, aplikačně programové rozhraní
ASP	Active Server Page, HTML stránka obsahující skripty
DB	Database, databáze
DBA	Database Administrator, administrátor databáze
DBMS	Database Management system, systém řízení báze dat
ERD	Entity Relationship diagram, nástroj pro modelování
GUI	Graphical User Interface, grafické uživatelské prostředí
HTML	Hypertext Markup Language, jazyk používající odkazy
JDBC	Java Database Connectivity, API specifikace pro spojování programů
JSP	Java Server Page, technologie pro kontrolu nad obsahem HTML stránek
LDAP	Lightweight Directory Access Protocol, zjednodušená verze pro adresářové služby v síti
MPP	Massively Parallel Processors, řízené vykonávání programů více procesory
ODBC	Open Database Connectivity, standard API pro práci s databází
OID	Object Identifier, identifikátor objektu
OLAP	Online Analytical Processing, online analytické zpracování
OLE DB	Object Linking and Embedding Database, standard API pro přístup k odlišným datovým zdrojům
OLTP	Online Transaction Processing, online transakční zpracování
OO	Objektově orientovaný
OODBMS	Object Oriented Database Management system, totéž jako OOSŘBD
OOSŘBD	Objektově orientovaný systém řízení báze dat
OR	Objektově relační
ORDBMS	Object Relational Database Management system, totéž jako ORSŘBD
ORSŘBD	Objektově relační systém řízení báze dat
PC	Personal Computer, osobní počítač
R	Relační
RBDMS	Relational Database Management system, totéž jako RSŘBD
RSŘBD	Relační systém řízení báze dat
SMP	Symmetric Multi Processing, vykonávání programů více procesory v rámci jednoho operačního systému
SQL	Structured Query Language, dotazovací jazyk
SŘBD	Systém řízení báze dat, překlad DBMS
XML	Extensible Markup Language, jazyk používající odkazy

1. Úvod

"Kde je moudrost?

Ztracena ve znalostech.

Kde jsou znalosti?

Ztraceny v informacích."

T. S. Eliot

"Kde jsou informace?

Ztraceny v datech.

Kde jsou data?

Ztracena v databázích."

J. Celko

1.1. Cíl práce

Tuto diplomovou práci věnuji jednomu z nejzajímavějších témat v oblasti práce s daty - modelu objektově relačního přístupu v databázích. Vzhledem k faktu, že dosud neexistuje žádná metodika, která by tuto oblast striktně popisovala, pokusím se nastínit model, který by pro danou dobu mohl být co nejlépe použitelný. Ale nejenom to, aby práce nebyla pojata čistě teoreticky, zmapuji a zhodnotím současný trh výrobců objektově relačních databázových systémů či jejich hybridů.

První část práce je teoretická a je věnována přístupům, ze kterých objektově relační model vychází. Jednotlivé konkrétní funkce a metody těchto přístupů jsou pak popsány a začleněny do návrhu OR modelu.

Následující část je právě stěžejní pro popis takového OR modelu. Jsou v ní definovány jednotlivé funkce, které by ideální ORSŘBD měl podporovat, aby byl schopen vyplňovat požadavky svých současných či budoucích uživatelů. Drtivá většina těchto funkcí nebo vlastností je definována na příkladech, které jsou uvedeny v přílohách této práce. Popis tohoto Objektově relačního SŘBD respektuje jak vlastnosti relačního modelu, tak i zabudování vlastností charakteristických pro objektové modely.

Poslední a zároveň čistě praktická část je věnována popisu současných databázových systémů, které jsou dostupné na našem trhu (a samozřejmě v dnešním globálním světě také na kterémkoliv jiném trhu). Pokusím se je i zhodnotit podle měřítek, které stanovuji v druhé části této práce. Část hodnocení jednotlivých produktů (databázových systémů) bude věnována možnosti realizace v konkrétních podnicích nebo situacích.

2. Objektový vs. relační přístup

2.1. Úvod do problematiky

V této kapitole se nachází stručné historické ohlédnutí za vývojem moderních výpočetních systémů s důrazem na oblast databázových technologií. Je zde také diskutována otázka přístupu k modelování s ohledem na typ přístupu k databázím.

2.1.1. Historické ohlédnutí

Úloha uložení dat (v čase) je jen o málo mladší, ne-li stejně stará, jako výpočetní technika sama. Datovat přesně vznik tohoto oboru jako samostatné lidské činnosti je záležitost značně ošemetná. První komerční aplikace výpočetní techniky se objevují na přelomu padesátých let minulého století a od té doby nabírá jejich vývoj rychlý spád.

Největšími milníky v oblasti vývoje hardwaru jsou nepochybně objev tranzistoru a posléze integrovaného obvodu, který od počátku osmdesátých zajišťuje začlenění výpočetní techniky do běžného života prakticky každého člověka v urbanizovaném světě.

Oblast softwarových technologií prodělala vývoj neméně dramatický. Od vývoje proprietárních "ad-hoc" řešení, která často nerozlišovala operační systém od vlastních aplikací, se postupně přešlo ke strukturované metodologii vývoje softwaru. Řešený problém byl dekomponován na úroveň základních funkcí, které se sdružovaly do funkčních jednotek - modulů. Světlo světa spatřily programovací jazyky třetí generace a nové metody ukládání dat - relační databáze. Nároky uživatelů a komplexnost úloh však neustále narůstaly a některá omezení klasického strukturovaného přístupu se stala překážkou ve vývoji aplikací velkého měřítka (large scale applications). Tyto problémy se snaží řešit nový náhled na celou problematiku návrhu a implementace informačních systémů - objektově orientovaný přístup.

V oblasti uložení dat - databázových systémů - je možno vysledovat vývoj rovněž značně dynamický, nicméně podle mého názoru lehce zaostává za oblastí vývoje aplikačního softwaru. Tato skutečnost však nalézá logické odůvodnění jednak v tom, že požadavky na způsob a logiku uložení dat jsou generovány ze strany aplikací a rovněž v tom, že databázový systém společně se systémem operačním tvoří ty esenciální a nejspolehlivější komponenty celého informačního systému a tudíž žádné dramatické změny nejsou žádoucí, důraz je kladen na stabilitu, spolehlivost a bezpečnost. Síťové a hierarchické databáze ([VER97/2]) výtečným způsobem reprezentovaly hierarchická data, jako například vztah nadřízená - podřízená organizační jednotka či vztah geografické závislosti okres-kraj-země. Jejich hlavní nevýhodou však byla skutečnost, že ukládaly aplikačně specifické optimalizační detaily, jako například ukazatele (pointer), přímo v databázi. Upgrade takovéto databáze se stal velmi obtížným úkolem právě díky "vyladění" pro danou konkrétní aplikaci. Ještě větší problém nastal s udržováním konzistence takto strukturované databáze ([VER97/2]).

V současné době jsou, co do komerčního uplatnění, zdaleka nejrozšířenější databázové systémy relační. Jejich koncept byl v sedmdesátých letech navržen E.F. Coddem, z části právě za účelem odstranit známé problémy databázových systémů té doby. Přes své nesporné kvality (potvrzené uplatněním na trhu) naráží dnes i tato technologie na své limity - nové typy dat (zvukové stopy, obrázky, videa,...) a složité datové struktury působí těmto databázím značné potíže. Tradiční výrobci databázových enginů se tyto problémy snaží odstranit přidáváním dodatečných funkcionalit do svých osvědčených "vlajkových lodí". Na trhu se také objevila početná skupina nových firem, které se, nezátíženy nutností rozvíjet stávající technologii, snaží o implementaci nových způsobů uložení dat. V souladu s duchem doby se jedná obvykle o objektové databáze. O těchto přístupech je pojednáno v dalších částech textu.

2.2. Přístupy k modelování

Nedílnou součástí databázových systémů je i jejich návrh a modelování. Aby se čtenář mohl v této práci lépe orientovat a mohl si jednotlivé modely databází správně začlenit do širokého celku týkajícího se informačních systémů, uvedu v této podkapitole principy modelování a jednotlivé modely, které je možné pro návrhy databází obecně použít.

Podle [REP99] existují tři základní definice modelu. Model znamená:

- 1) Formální vyjádření zkoumaného jevu (systému) sloužící jako vyjádření skutečnosti.
- 2) Zjednodušené zobrazení určitého jevu (systému) pomocí vhodných zobrazovacích prostředků znázorňujících pouze ty rysy, jež jsou podstatné z hlediska cíle, který při konstrukci modelu sledujeme.
- 3) Reprodukce charakteristik určitého objektu na objektu jiném, zvláště vytvořeném pro jejich studium.

Pokud se budeme soustředit na modelování databází, je nutné uvažovat základní charakteristiku databáze, která má být namodelována. Tato charakteristika vychází právě z přístupu realizace, tedy relačního nebo objektového.

Obecně se dají přístupy k modelování dle dnes dostupné literatury rozdělit do dvou hlavních skupin odpovídajících charakteristice databáze.

- 1) Strukturovaný přístup
- 2) Objektově orientovaný přístup

2.2.1. Strukturovaný přístup

Oblast strukturovaného přístupu a relačních technologií spolu značnou měrou souvisí a dá se konstatovat, že pro návrhy relačních databází se aplikují právě strukturované přístupy. Tato souvislost je patrná i historicky, protože oproti objektovému přístupu se rozvoj těchto oblastí datuje zhruba do stejného času.

Strukturovaný přístup se snažil o dekompozici komplexních problémů na elementární funkce a tyto pak sdružoval do funkčních modulů. Jeho hlavním přínosem bylo jednak zjednodušení a zpřehlednění do té doby monolitického aplikačního kódu a zejména pak znovupoužití již jednou definovaných algoritmů skrze opětovné a vzájemné volání funkcí. Funkce pracovaly s relativně jednoduchými datovými strukturami složenými ze základních datových typů, přičemž tyto struktury byly odděleny od vlastní funkcionality (tato skutečnost se později stala jednou z kritizovaných vlastností strukturovaného přístupu; umožňovala totiž vznik chyb v důsledku nekorektního přepisování globálních datových struktur různými funkcemi).

Této filozofii přesně odpovídaly i relační databáze. Umožňovaly ukládat omezenou množinu základních datových typů a aplikacím poskytovaly poměrně pohodlnou a spolehlivou metodu dlouhodobého uložení dat. Zejména spolehlivost a robustnost dnešních relačních databázových serverů jsou faktory, které tyto systémy předurčují pro použití i v situacích, kdy je vlastní aplikační logika navržena a implementována v objektovém prostředí.

2.2.1.1. Strukturované metody

Pokud ke konstrukci uvažujeme použití strukturovaných metod, je nutné rozdělit modelování do dvou fází [REP99]:

- 1) analýza systému - výsledek této fáze odpovídá konceptuálnímu modelu. Na konceptuální úrovni je model reality tvořen třemi modely:
 - a) Datovým modelem, který je statickým popisem reality - jakými prvky a jejich vzájemnými vazbami je realita tvořena,
 - b) funkčním modelem, popisujícím procesy a vztahy mezi nimi,
 - c) modelem řízení, popisujícím časové následnosti jednotlivých proces.

Funkční model a model řízení dohromady jsou popisem "dynamiky" v realitě - jakým způsobem se realita mění (resp. jakými transformacemi procházejí informace o realitě).

- 2) konstrukce (design) systému - výsledkem této fáze je technologický model. Na technologické úrovni je model systému tvořen:

- a) Modelem programové struktury systému (odpovídá procesní části konceptuálního modelu),
- b) modelem logických datových struktur (odpovídá datové části - vyplývá z konceptuálního datového modelu).

2.2.1.2. Nástroje pro strukturované metody

Aby byl přehled týkající se strukturovaného přístupu kompletní, je nutné vyjmenovat nástroje, které se v jednotlivých fázích modelování používají (nebo mohou použít). Tato podkapitola poskytuje pouze základní přehled o nástrojích, jejich konkrétní popis není pro tuto práci stěžejní a je dostupný ve velkém množství dnešní literatury [TOR99], [POK98].

Pro konceptuální popis existuje:

- 1) ERD (Entity Relationship Diagram) - diagram popisující vztahy mezi jednotlivými entitami v rámci modelu,
- 2) DFD (Data Flow Diagram) - diagram pro popsání datových toků mezi procesy, datovými sklady, atp.

Jako podpora těchto nástrojů se využívá strukturových diagramů (Structure chart), datových slovníků (Data Dictionary) a dalších nástrojů v rámci technologického a implemetačního popisu.

2.3. Relační databázové systémy

Základním stavebním kamenem relační databáze je relace (nebo též tabulka). Data jsou v tabulkách ukládána ve formě záznamů, které tvoří jednotlivé řádky tabulky. Každý záznam je v podstatě datová struktura složená ze základních datových typů podporovaných daným konkrétním databázovým serverem. Všechny záznamy jedné tabulky musejí být shodného typu (identické datové struktury). V horizontálním směru v tabulkách rozeznáváme sloupce - jednotlivé prvky datové struktury - atributy.

Speciálním typem sloupce či množiny sloupců je takzvaný primární klíč. Jeho hodnota (u primárních klíčů složených z více sloupců pak kombinace těchto hodnot) musí být v rámci jedné tabulky jedinečná a slouží k identifikaci záznamu v rámci tabulky.

Relace mezi tabulkami se zachycují pomocí tzv. cizích klíčů. Jde, zjednodušeně řečeno, o přítomnost primárního klíče záznamů nadřazené tabulky (master table) v záznamech podřazené tabulky (detail table). Primární klíč nadřazené tabulky je tedy v záznamu podřazené tabulky v pozici cizího klíče a vyjadřuje určitý vztah tohoto záznamu k záznamu tabulky nadřazené. Tímto způsobem lze modelovat vztahy s kardinalitou 1:1 a 1:N, vztahy M:N je nutno zachytit za pomoci tzv. vazební (průsečíkové) tabulky (intersection table).

2.3.1. Přednosti relačních databází

Své dobré jméno si relační databázové servery vydobýly, kromě své robustnosti a stability, i díky následujícím vlastnostem implementovaných v běžně komerčně dostupných produktech předních databázových producentů:

- Realizují pokročilé metody výběru dat z databází. Jedná se především o různé druhy indexace záznamů, způsobů spojování tabulek a implementace sofistikovaných plánovačů úloh. Pokud tyto selhávají, umožňují moderní servery vkládání optimalizačních direktiv přímo do SQL dotazů.
- Umožňují definici pravidel zajišťujících udržení konzistence a integrity dat (např. lze zakázat výmaz záznamu nadřízené tabulky, pokud v tabulkách podřízených existují záznamy k němu se vztahující).
- Jsou optimalizovány pro paralelní zpracování na víceprocesorových systémech. Takovýto systém dovoluje rozložit obsah obzvláště velkých tabulek na několik nezávislých diskových zařízení, přičemž každý procesor pak samostatně zpracovává úlohu nad daným fragmentem tabulky, což v konečném výsledku umožňuje poměrně značné zkrácení doby potřebné pro zpracování dotazu.
- Dovolují (s určitými omezeními) konstrukci transparentních distribuovaných systémů a definici replikací. Tato vlastnost umožňuje vytvářet systémy rozsáhlé co do geografické rozlohy i objemu dat, jedná se však o extrémně komplexní záležitost, která se v praxi neobejde bez další programové podpory. Obecně lze ale konstatovat, že moderní databázové systémy tuto možnost poskytují.
- Ultimátní předností je bezpochyby implementace transakcí a to i nad různě distribuovanou a fragmentovanou datovou základnou (užívá se pak tzv. dvoufázového komitu). Transakci je možno chápat jako množinu operací, které databázi převedou z jednoho konzistentního stavu do druhého. Uplatní se tedy buďto všechny operace množiny, nebo žádná z nich. Pokud uprostřed probíhající transakce dojde k jakýmkoliv problémům, aplikuje se tzv. rollback - dopad všech doposud vykonaných transakcí je odstraněn a databáze se vrací do původního konzistentního stavu.

Kromě výše uvedených předností se mohou moderní relační databázové servery pochlubit ještě celou řadou dalších pozitivních vlastností, jako jsou například kvalitně zvládnutá podpora velkého počtu současně pracujících uživatelů, či dobrá přenositelnost napříč platformami, tzn. že tyto databázové servery jsou portovány do všech důležitých operačních systémů (založených zejména na bázi UNIXu).

Za výhodu je rozhodně možno považovat i použití standardizovaného a efektivního jazyka SQL pro definici a manipulaci s daty, který je ovšem jednotlivými výrobci rozšiřován (oproti standardu) o různá vylepšení. Dopad těchto rozšíření je sporný: na jednu stranu zjednodušují samotnou tvorbu aplikací a mnohdy dávají prostor k optimalizaci, čímž v důsledku podstatným způsobem zvyšují

výkon. Na druhou stranu ruší veškeré výhody ze standardizace plynoucí. Přenos hotových aplikací mezi databázovými servery pak není bez obtíží.

2.3.2. Stinné stránky relačních databází

Moderní čistě relační databázové servery se nemohou chlubit pouze vlastnostmi pozitivními. Jejich hlavními nedostatky jsou podle [VER97/2] následující skutečnosti:

- množina podporovaných datových typů je omezená a dále nerozšiřitelná.
- zachycení komplexních objektů pomocí tabulek je obtížné
- navigace mezi objekty je v SQL obtížná
- analytické operace nad daty jsou s využitím SQL příliš omezené

Požadavek na rozšíření funkcionality vzniká zejména z potřeby ukládat a především zpracovávat nové typy dat jsou například zvukové stopy, obrázky, videa. Bylo by například žádoucí, aby bylo možné úspěšně vykonat následující SQL dotaz:

```
select document.name
  from document
 where contains (document.text, 'budoucí investice');
```

Tento dotaz by měl z tabulky dokument vyzvednout jména všech dokumentů, ve kterých se vyskytuje řetězec "budoucí investice". Předpokladem je, že dokumenty uložené v této tabulce jsou formátu Microsoft Word. Tento dotaz by samozřejmě neprošel syntaktickou kontrolou překladače, ale hlavní překážkou v relaci zůstává skutečnost, že databázový server nerozumí formátu textu, v jakém je uložen. obdobná omezení platí i pro ostatní "bohaté" datové typy (rich data types), jakými jsou například video, obrázky, atd..

Rovněž nutnost rozkládat složité datové struktury (objekty) do množiny několika tabulek popř. jinak modifikovat jejich uložení působí problémy při jejich opětovném vyzvedávání a má dopad zejména na výkonnost a dobu odezvy systému.

2.4. Objektově orientovaný přístup

Objektově orientovaný přístup vznikl jako odpověď na tuto narůstající komplexitu. Jedná se o nový přístup a formu myšlení, která se snaží jak analýzu tak implementaci (psaní kódu) přiblížit více reálnému světu. Alfou a omegou objektového přístupu je samozřejmě objekt.

Objekt je entita, vyznačující se určitými vlastnostmi a definovanou množinou chování. Ve své podstatě se jedná o sloučení datových struktur a nad nimi definované funkcionality do autonomní celistvé jednotky. Korektní práci s těmito objektovými strukturami zajišťují metody objektu (procedury a funkce), přičemž okolní svět má možnost měnit obsah takových struktur pouze skrze publikované metody objektu. Tato vlastnost se v objektové terminologii nazývá zapouzdření a odstraňuje jednu z významných nedokonalostí strukturovaného přístupu. S datovými strukturami manipuluje pouze přesně vymezená, relativně malá skupina funkcí (metod), což má zabránit vzájemnému přepisování obsahu datových struktur jednotlivými funkcemi, například daty v nevhodném formátu.

Objektový přístup však zdaleka není jen o zapouzdření. Dle [NAH91] se jednotlivé aspekty objektového přístupu obvykle popisují následujícími termíny (termíny byly převzaty včetně jejich vysvětlení):

- zapouzdření (encapsulation) - jiný termín pro skrývání informací
- třídy (classes) - seskupení podobných objektů
- dědičnost (inheritance) - sdílení vlastností, které charakterizují třídu všemi jejími podtřídami (subclasses)
- zprávy (messages) - základní mechanismus provádění programu. kdy zprávy zasílané mezi objekty vyvolávají jednotlivé procedury objektů
- perzistence, přetrvávání (persistence) - vztahuje se k myšlence dlouhotrvajících či perzistentních objektů, tzn. že objekty pokračují ve své existenci i poté, kdy aplikace, jež je vytvořila, již ukončila svůj běh
- polymorfismus (polymorfism) - různé objekty různě reagují na shodné zprávy; to implikuje nutnost pozdního svázání funkčních volání (binding of function calls)

Objekt je tedy určitou kvalitnější abstrakcí reality resp. v realitě existujících entit a objektový přístup samotný pak ve svém důsledku umožňuje její přirozenější a snazší modelování. Nazíráno z druhé strany, entity reálného světa lze popsat jejich vlastnostmi a lze u nich vysledovat určité postupy chování, což přesně koresponduje s abstraktním pojetím objektového přístupu.

2.4.1. Objekt vs. relační entita

Relační entita je identifikována množinou definovaných atributů, resp. hodnotami, které tyto atributy nabývají. Jinými slovy se jedná o určitou abstrakci reality a to ať již abstrakci ve smyslu definice entity a jejích atributů na konceptuální úrovni, či abstrakci konkrétní skutečnosti, kdy atributy entity nabývají reálných (z reality převzatých) hodnot.

Objekt, oproti relační entitě, navíc tvoří společně s datovými poli (atributy) i množina operací, definujících jeho chování. Tyto operace se v objektové terminologii nazývají metody a kromě toho, že zajišťují interakci objektu s okolím, mají metody (dle "čisté" objektové specifikace) jednu výlučnou vlastnost: jako jediné mohou manipulovat s obsahem atributů (datových polí) objektu. Tato vlastnost se nazývá zapouzdření a v praxi bývá často povoleno její porušení - jiné objekty přímo přistupují k obsahu atributů daného objektu.

Přidání metod k atributům a zapouzdření však není to jediné v čem se objekty odlišují od relačních entit. Profesor Pokorný [lit6, str6] tvrdí:

"Co dělá objekt objektem je jeho identita, která je nezávislá na hodnotě objektu. Dva objekty mohou být identické (jde o týž objekt), nebo si mohou být rovny (mají stejnou hodnotu)."

2.4.2. Identita objektu

U klasických relačních tabulek je každý záznam (řádka tabulky) identifikována obsahem svých atributů. K jednoznačné identifikaci obvykle slouží jen některé z nich (popř. pouze jeden), které pak dohromady tvoří primární klíč tabulky.

U objektů je pojem identity (a s ní související jednoznačná identifikace) vnímán odlišně. Objekty jsou reprezentovány pomocí tzv. identifikátorů objektů - OIDs (Object Identifiers). Zatímco se stav objektu (obsah jeho atributů) může libovolně měnit, objektový identifikátor OID zůstává neměnný. Naopak v daný okamžik zcela shodné hodnoty mohou být reprezentovány dvěma různými objektovými identifikátory - v praxi to znamená, že dvě entity (reprezentované dvěma objekty) se v daný okamžik nacházejí ve shodném stavu, stále se ale jedná o dvě samostatné části reality v systému reprezentované dvěma odlišnými objekty.

Ve svých důsledcích to znamená, jak již bylo zmíněno v předchozí kapitole, že dva objekty se sobě mohou rovnat tzn. hodnoty jejich atributů jsou vzájemně shodné, přesto se stále jedná o dva různé objekty. Rovněž to znamená, že zdánlivě dva objekty (nacházející se na různých místech systému) mohou být identické - ve skutečnosti se ale jedná o jeden sdílený objekt.

Objektové identifikátory jsou důležité pro vyjádření vazeb mezi objekty, které umožňují přechod od jednoho objektu ke druhému - tzv. navigaci. Jedná se o vlastnost objektového řešení, která má být jedním z jejích důležitých přínosů, umožňuje totiž značný nárůst výkonnosti databáze (resp. databázového stroje).

2.4.3. Objektové databázové systémy - OOSŘBD

Přestože objektové databáze nejsou tématem této diplomové práce, je vhodné se o nich zmínit alespoň v samostatné podkapitole. S objektově relačními databázemi mají některé společné rysy a do určité míry si konkurují v nabízených řešeních.

Dle [POK96] by OOSŘBD měl splňovat tyto kritéria:

- měl by být SŘBD, tj. měl by zajišťovat persistenci dat na vnějších pamětech, souběžný přístup více uživatelů, spolehlivost dat a možnost ad hoc dotazů
- měl by být objektově orientovaný, tj. připouštět existenci identity objektů, typy nebo třídy, zapouzdření (objektů a metod, které s nimi manipulují), polymorfismus, dědění, složené objekty, rozšiřitelnost a výpočetní úplnost.

Objektově relační systémy řízení báze dat splňují první požadavek beze zbytku, přičemž v ideálním případě by měli vyhovět i všem nárokům vzneseným v požadavku druhém. To, co tyto dva světy od sebe odlišuje, je různý startovací bod. Zatímco objektově relační databázové systémy staví na již existujících databázových strojích, ke kterým objektovou funkcionalitu přidávají, pravé objektové SŘBD jsou budovány "od nuly". To umožňuje tvůrcům objektových systémů zabudovat objektový přístup do samotného konceptu vlastního stroje, což by se mělo odrazit v systému uložení dat a samotné jeho práce.

Je nutné konstatovat, že neexistuje žádné objektové paměťové médium. Z toho plyne, že objektové struktury musejí být na médiu (obvykle index-sekvenčního typu) simulovány podobným způsobem, jak je tomu u relačních nebo objektově relačních databází. Výhodou pravých objektových databázových systémů je možnost navrhnout optimalizované datové struktury přímo pro vyjádření objektů, zatímco objektově relační stroje obvykle využívají pro uložení objektů stávající relační mechanismy (struktury), což nemusí být, zejména co se výkonu týká, vždy nejvýhodnější.

Skutečné důvody pro použití termínu objektově relační ze strany jejich producentů jsou dle mého názoru dva:

- 1) Množina implementované objektové funkcionality není prakticky v žádném případě natolik úplná, aby opravňovala výrobce k označení jím produkovaného SŘBD jako objektového SŘBD. Neúplnost přitom může být záměrná (výrobce se nesnaží o vývoj objektového SŘBD; pouze vhodným způsobem rozšiřuje stávající relační technologii o pozitivní rysy objektového

přístupu) nebo nechtěná (vývoj plnohodnotného OOSŘBD je natolik složitý, že výrobce prozatím není schopen poskytnout kompletní OO řešení).

- 2) Termínem objektově relační vyjadřuje výrobce určitou kontinuitu vývoje a návaznost na jeho spolehlivý a osvědčený relační databázový stroj.

Z výše uvedených poznatků a na základě již zmíněných požadavků kladených na objektově orientované databázové systémy je možno dovodit i důvody pro jejich zatím malé uplatnění na komerčním trhu. Vývoj robustního a spolehlivého databázového stroje obsluhujícího desítky až tisíce současně přístupujících uživatelů, implementujícího transakce a distribuované zpracování a zvládajícího objemy dat pohybující se až v řádu terabajtů je záležitost vyžadující roky vývoje. Komerčně dostupné objektové produkty tedy ještě musí tyto vlastnosti prokázat v podmínkách reálného nasazení a jejich producenti musí v této oblasti vyvinout ještě značné úsilí.

Zavádění nových objektových technologií do oblasti ukládání dat má zcela jistě své opodstatnění. V [POK96] jsou citovány následující typy aplikací, u nichž lze s pomocí tohoto přístupu dosáhnout lepších výsledků:

- geografické informační systémy
- návrh logických obvodů pomocí počítače
- monitorování a řízení procesů v reálném čase
- data pro správu sítě

Jedná se tedy o aplikace, které vyžadují uložení složitých datových struktur a jejich opětovné získání z databáze buďto v krátké době nebo ve velkém objemu. Rozložení takových struktur do nutně netriviální struktury relačních tabulek může působit jako výkonnostní brzda, přičemž objektový přístup se zde jeví jako přirozený způsob řešení uvedených potíží.

Čistě objektové databáze však, zdá se, stále trpí počátečními nemocemi a tak se jako přirozená náhrada nabízejí databáze objektově relační. Bylo by ovšem chybou deklasovat objektová rozšíření tradičních relačních databází pouze do role přechodného řešení. Jak ukáže následující text, tyto nové vlastnosti přinášejí mnoho pozitivních změn i do oblasti relační.

2.5. Srovnání přístupů

Každý z přístupů má své klady a zápory. Tyto vlastnosti se většinou reprezentují na základě praktického využití těchto přístupů a vycházejí z jejich konceptu.

2.5.1. Silné stránky objektového přístupu

V následujících bodech jsou reprezentovány silné stránky objektového přístupu tak, jak jsou uvedeny v [VER97/1]:

- schopnost reprezentovat komplexní entity (objekty) reálného světa
- zapouzdření dat s asociovanými obchodními pravidly a metodami
- oddělení toho "co" objekt dělá od toho "jak" to objekt dělá s využitím dědičnosti a polymorfizmu. To umožňuje kvalitní logické modelování a abstrakce
- schopnost zachytit vztahy mezi objekty a jednoduchým způsobem přecházet z jednoho objektu na druhý (navigace)
- schopnost vytvářet nové typy dat

2.5.2. Silné stránky relačního přístupu

Silné stránky relačního přístupu tak, jak v průběhu času vyplynuly zejména z praktického nasazení v relačních databázových systémech dle [VER97/1]:

- podpora vysokého objemu transakcí
- podpora uložení velkých objemů dat
- paralelní zpracování, rozložení zátěže a indexování
- robustní datová integrita a bezpečnostní vlastnosti
- nástroje pro obnovení po havárii - zálohovací (backup) a obnovovací (restore) nástroje
- robustní dotazovací jazyk SQL
- podpora současné práce více uživatelů

V podstatě i všechny tyto vlastnosti již byly diskutovány v předešlém textu. Z uvedených vlastností jsou jasně patrné požadavky, které byly kladeny na relační databázové servery již od dob jejich prvních komerčních aplikací na počátku osmdesátých let. Jsou jimi spolehlivost, integrita, vysoký transakční výkon a stabilita.

3. Objektově relační model SŘBD

Druhá část diplomové práce je zaměřena na návrh objektově-relačního datového modelu. Tento model je popsán na základě vlastností, které přísluší jak relačním datovým modelům, tak i objektovým. Jedná se o kombinaci těchto jednotlivých vlastností a výsledkem je hybridní systém, který by měl "ideálně" vyhovovat současným požadavkům trhu.

Tato stěžejní kapitola je rozdělena na několik podkapitol, které se zabývají jednotlivými vlastnostmi navrhovaného modelu. Princip popisu jednotlivých vlastností je takový, že uvažují relační datový model jako základnu pro obecný SŘBD (vychází i z historie vývoje, viz [POK02]) a na takový koncept nasazují vlastnosti charakteristické pro objektově orientovaný model tak, aby bylo možné s navrženým systémem dosáhnout jak velké spolehlivosti, tak i výkonu.

Pro bližší pochopení objektově relačního modelu jsou v této hlavní kapitoly využity názorné příklady v SQL, které vycházejí z [SKI03]¹.

3.1. SQL a OR model

Pokud se ohlédneme do historie, tak se datové typy s postupem času stávají více komplexními a uživatelé požadují na těchto datových typech možnost používat a vykonávat různé dotazy, vyhledávání a mnoho dalších funkcí. Objektově-relační systémy řízení báze dat jsou vyvíjeny k uspokojování těchto potřeb. Jak je uvedeno v úvodu této kapitoly, objektově-relační SŘBD je hybridem dvou rozdílných typů SŘBD. Má výpočetní výkon relačních SŘBD a stejně jako objektově orientované SŘBD umí pracovat s komplexními datovými typy. ORSŘBD se také někdy nazývají „univerzální servery“.

Objektově-relační SŘBD může manipulovat se stejnými datovými typy jako relační SŘBD – to zahrnuje typy, které mohou být používány v SQL, jako číslo, číslo s plovoucí desetinnou čárkou, datum, znakový řetězec. Oba typy SŘBD mohou ukládat například záznamy zaměstnanců nebo zákaznické účty. Ačkoliv je relační SŘBD výkonný, neumí pracovat s komplexními datovými typy nebo uživatelem definovanými typy. V současné době uživatelé kromě standardních vyžadují datové typy jako audio, video, geometrické body, dokumenty, obrázky, atp.

Objektově orientované SŘBD dokáží dobře a výkonně pracovat s komplexními daty. Podporují také objektové principy jako polymorfismus, dědičnost a zapouzdření. Datové typy OOSŘBD v sobě obsahují hodnoty a funkce, které jsou schopné pracovat na těchto objektových principech. Stejně tak ORSŘBD podporují stejné objektově orientované datové typy, a tak se stávají konkurenty objektově

¹ ze sekce: Databázové technologie - charakteristika ORSŘBD, mechanismy a implementace ORSŘBD
SQL programování - dotazy v databázích (přeloženo z angličtiny)

orientovaných systémů. Podporují také uživatelem definované datové typy, které mohou být konstruovány uživatelem (programátorem) nebo nějakým způsobem dodány třetími stranami.

Objektově relační systémy musí obsahovat podporu systému pravidel pro tvorbu databází [STO00]. Ten definuje, jak má SŘBD nakládat s událostmi typu dotazů a aktualizací (update). Tento soubor pravidel a podpora pro hierarchické strukturování jsou převzaty z objektově orientovaných SŘBD, nicméně aplikace probíhá pomocí SQL jako v relačním modelu. ORSŘBD podporuje SQL-3. Na trhu s databázemi existuje několik variací SQL v závislosti na určitých ORSŘBD ([POK02]).

Objektově relační SŘBD podporuje uživatelské dotazy SQL-3 a to umožňuje uživatelům vytvářet stejné dotazy, jaké mohli používat za použití relačního modelu a SQL-92, a kromě toho může vytvářet i dotazy podle obsahu na komplexní datové typy. To je obvykle umožněno ve spojení se softwarem, který například vyhodnocuje vzorky v obrázku nebo tónování ve vzorku hudby. V OOSŘBD jsou obecně data zpracovávána programy, které jsou napsané v objektově orientovaných jazycích třetí generace jako C++, jež neumožňují vytvářet specifické uživatelské dotazy. Tradiční OOSŘBD se zaměřují na ukládání objektů tak, že komplexní relační dotazy nejsou nutné [MER02][POK98].

Relační SŘBD jsou naopak celkem často využívány v rámci OLTP, tedy pro práci s transakcemi. Transakce jsou obvykle skládány z relativně jednoduchých SQL příkazů s mnoha aktualizacemi (update). Výhody transakčního zpracování jsou rychlost, výkonnost a přesnost. Objektově relační SŘBD jsou také využívány pro práci s dotazy pro podporu rozhodování potřebujících vysoký výkon. Kvůli složitosti dat však tyto dotazy nedosahují takových výkonů v rychlosti jako klasické relační systémy. Částečně je tomu tak proto, že ORSŘBD používá metody, které jsou o mnoho více komplexní nežli ty využívané relačními systémy. Ačkoli jsou tyto metody komplexnější, optimalizace zajišťuje, že jsou vykonávány tak efektivně, jak je to možné. Optimalizace probíhá obvykle tak, že jsou nejdříve zpracovávána jednoduchá data, čímž se redukuje množství zpracovaných komplexních dat v dotazu.

3.1.1. Bezpečnost

Od doby, kdy se v rámci databází začala používat klíčová a důvěrná data, nebyla bezpečnost nikdy snižována ve prospěch výkonu. Není tomu tak ani v ORSŘBD. A od doby, kdy se začaly dotazy tvořit přes SQL, což není jazyk třetí generace, je celkový rozdíl ve výkonu velmi malý ve srovnání se situací, kde by dotazy sdílely stejnou adresu jako SŘBD.

Relační SŘBD zvládají relativně jednoduché datové typy, všechny mohou být reprezentovány alfanumerickými znaky. Proto vytváření reportů a forem pohledů pro vkládání a zobrazení dat je vcelku přímočaré a jednoznačně dané. V současnosti existuje na trhu nepřeberné množství jazyků čtvrté generace (4GL), které je možné použít pro vytvoření uživatelsky příjemného prostředí. Na

druhé straně jsou programovány OOSŘBD aplikace pro řízení a manipulování s uloženými daty. Programátoři tak obecně mají k dispozici více nástrojů nežli koncoví uživatelé.

Přestože klientské nástroje relačních SŘBD mohou manipulovat s datovými typy SQL-3 kompatibilními s SQL-92, jsou příliš omezené, než aby je bylo možné efektivně použít v ORSŘBD. Pro práci s více komplexními daty jsou zapotřebí více sofistikované nástroje. Například jazyky čtvrté generace (4GL) používající výstupy relačních SŘBD, které nedovolí například porovnání obrázků nebo audia. Ačkoliv ORSŘBD vyžadují jiné sady klientských nástrojů než relačních SŘBD, sady pro oba typy musí uživateli dovolit následující:

- vkládat data,
- vykonávat dotazy,
- vytvářet reporty,
- definovat databáze,
- volat služby SŘBD přes jazyky třetí generace.

3.1.2. Datové typy v OR modelu

Z předchozí kapitoly je patrné, že relační SŘBD podporují SQL-92. SQL-92 omezuje sloupce (atributy) v tabulkách relačních SŘBD pouze na tyto typy:

- Číslo,
- datum,
- časový interval,
- číslo s plovoucí desetinnou tečkou,
- znakový řetězec s pevnou délkou,
- znakový řetězec s variabilní délkou.

SQL-92 také definuje aritmetické operace, porovnávací operátory a funkce, které mohou být použity s daným datovým typem. Ačkoliv jsou velmi dobře použitelné, rozhodně nezvládnou vše. Je sice možné kombinovat datové typy, funkce a operátory SQL-92, aby bylo možné simulovat uživatelem definované datové typy, nicméně tento způsob znamená ohromný nárůst náročnosti pro uživatele při zadávání dotazů. A také výkon není tak dobrý jako při použití uživatelem definovaných datových typů, které jsou podporovány ostatními SŘBD ([POK02]).

Jako příklad uvedu databázi, která bude obsahovat záznamy o vybavení skladů výrobce nábytku a jeho prodejních míst. Každý záznam by měl obsahovat umístění budovy (lokaci). SQL nemá definován datový typ, který reprezentuje geografické body. Nicméně je vcelku snadné deklarovat číslo s plovoucí desetinnou tečkou reprezentující zeměpisnou šířku a druhý reprezentující zeměpisnou délku. Problémy vzniknou ve chvíli, kdy se uživatel snaží vykonat dotaz vztahující se k geografickému bodu.

Pro názornost je třeba dotazem zjistit, který je nejbližší sklad ke konkrétnímu prodejnímu místu. V tomto případě musí uživatel napsat příkaz SQL-92 reprezentující daný dotaz. Dotaz musí definovat, co znamená "nejbližší" v rámci algoritmu, který může být použit k určení hodnoty pro každou lokaci. SQL dotaz také musí specifikovat, jak by měli být porovnávány hodnoty lokací, a jak určit hodnotu výsledku. Ve stručnosti je možné konstatovat, že uživatel musí vykonat dotaz, který je komplexní a náročný.

Každý ORSŘBD podporuje prostředí SQL-3. Ale ani SQL-3 nemá nadefinován jako datový typ geografický bod. Nicméně umožňuje uživateli si tento typ nadefinovat, stejně jako jiné datové typy. Také umožňuje definovat funkce a operátory, které mohou být použity jak na uživatelem definovaných typech tak na standardních typech SQL-92. Tento krok vlastního definování funkcí a operátorů je možné eliminovat nakoupením předdefinovaných datových typů, jejich funkcí a s nimi spojených operátorů.

Pokud použijeme příklad výrobce nábytku, v SQL-3 je možné definovat datový typ, který bude reprezentovat lokaci každého skladu a maloobchodu (prodejního místa). Pak je možné definovat funkci, která vezme lokaci maloobchodu a skladu jako své parametry a vrátí vzdálenost těchto dvou objektů. Tato funkce vracející vzdálenost mezi dvěma budovami může být zavolána z jiné funkce, která má jako parametr pouze maloobchod. Funkce vrátí vzdálenost maloobchodu od každého skladu a vrátí sklad, který je k danému maloobchodu nejbližší. Od chvíle, kdy jsou datové typy a funkce již jednou vytvořeny, potřebují dotazy uživatele pouze specifikaci toho, že potřebuje pouze nejbližší sklad k určitému maloobchodu. Narozdíl od dotazů SQL-92 není potřeba aby uživatel specifikoval, jak toho má být dosaženo. Maloobchod je zadán do uživatelem definované funkce, kde je většina kódu dotazu již obsažena. V tomto případě ORSŘBD poskytuje lepší běh při použití SQL-3 nežli relační SŘBD s SQL-92.

SQL-3 rozumí a pracuje přímo s novými datovými typy, takže se neztrácí výkon v překládání datového typu do srozumitelné formy pro SQL-92.

Existují rozdíly mezi jednotlivými koncepty SŘBD podle toho, které datové typy mohou zahrnovat. Koncept domén relačního modelu specifikuje, že datové typy jednoduše znamenají uložení dat určitými konkrétními způsoby. ORSŘBD datové typy zahrnují data a informace o uložení. Mohou také obsahovat funkce, které mění vstup na svůj interní uložený formát a zpět.

```
create type velikost_kontejneru(  
    vnitřnídelka = 6  
    input = velikostNaInt  
    output = velikostNaASCII);
```

Tento příklad ukazuje, jak je možné napsat definici na uživatelem definovaný typ nazvaný "velikost_kontejneru". První řádka za závorkou specifikuje 6 bytů úložného místa, které se bude alokovat pro každou instanci velikosti kontejneru. Následující dva řádky specifikují, jak budou reprezentovány datové typy při uložení (input) a při zobrazení (output). Input volá uživatelem

definovanou funkci "velikostNaInt", která mění hodnotu velikosti kontejneru z formátu, ve kterém jsou data ukládána – ASCII na číslo (integer). A funkce "velikostNaASCII" pak konvertuje hodnotu zpět na ASCII formát. Obecně a při základním nastavení jsou data reprezentovaná uživatelem definovaných datových typů ukládána v ASCII formátu, což je v mnoha případech dostačující. Například text uložený jako alfanumerické znaky není nutno měnit na jiný formát, nicméně pokud se data převedou do numerického formátu, je s nimi pak možno provádět aritmetické operace. Funkce pro převod dat mohou být používány také pro účely jiné, nežli jen pro konverzi hodnot pro interní/externí reprezentaci. Je možné vytvářet datové typy, které zakódují hodnoty ve chvíli, kdy se mají uložit. Následně pak mohou být rozšifrována v případě, že mají být zobrazena. Konverzní funkce mohou být také použity pro umisťování hranic na ukládaná data. V příkladě skladů a maloobchodů je možné omezit geografické body tak, že nebudou akceptovat hodnoty mimo specifickou zemi. Konverzní funkce musí být specifikovány pouze v případě, že data mají být uložena v jiném formátu, než jsou ukládána.

Pro použití v ORSŘBD je možné psát funkce v SQL-3. Je ale také možné psát funkce v jazyku třetí generace (jako například C) a pak použít SQL-3 pro jejich použití v systému. Pokud se v SQL-3 vytváří funkce, je třeba specifikovat její jméno, parametry a jaký typ se vrátí. Je také nutné zapsat kód, který bude vykonán při zavolání funkce.

```
create function spocitej_zisk(float, float)
    returns float as
    select ($1 - $2);
```

Zde je vytvořena jednoduchá funkce, která určuje z prodeje předmětu odečtením nákladů z jeho ceny. Jméno funkce je "spocitej_zisk". Parametry funkce jsou dvě čísla s plovoucí desetinnou čárkou, jedno reprezentující cenu a druhé reprezentující náklady na produkci. Tím, že to jsou rozdílné hodnoty, je třeba specifikovat, v jakém pořadí mají do funkce vstoupit. Funkce obsahuje příkaz SELECT, který bude vykonán při běhu funkce. V tomto případě je cena reprezentována "\$1", což odpovídá prvnímu parametru funkce. Druhý parametr pak znamená náklady a je ve funkci reprezentován "\$2". Funkce jako výsledek vrací číslo s plovoucí desetinnou čárkou. Toto číslo je naplněno hodnotou výsledku příkazu SELECT. Jakmile je funkce jednou definována, je možné ji kdykoliv zavolat v rámci SQL-3 dotazu.

```
select i.polozka
from inventar
where spocitej_zisk(i.cena_polozky, i.cena_nakladu) > 100;
```

Obecně jsou funkce volány specifikací jejich jména a parametrů. Parametry musí být specifikovány ve stejném uspořádání, v jakém je funkce očekává. Při vykonávání dotazu vloží ORSŘBD tělo funkce do tohoto dotazu. Tato vlastnost systému se nazývá rozvíjení (unwinding).

Funkce, která není v SQL musí obsahovat stejné elementy jako SQL funkce (ale definice je odlišná, viz například jméno a jazyk ve kterém je definována).


```
create function(cena_polozky, cena_nakladu)
    returns float
    as external name 'zisk'
    language C;
```

Musí také být registrována před tím, než může být použita. Při registrování funkce, která není napsána v SQL je třeba specifikovat jméno funkce, parametry a jaký typ má být vrácen. Je nutné připojit externí jméno a specifikovat jazyk, ve kterém je funkce napsána. Uživatelem definované funkce napsané v SQL-3 jsou ale lépe vykonávány nežli funkce napsané v jazycích třetí generace jako v C. Je tomu tak proto, že funkce nepsané v SQL nejsou podporovány vlastností rozvinutí (unwind) při vykonávání. Místo toho musí být funkce volána prostředkem pro vykonávání (tzv. stroj).

3.1.3. Operátory

SQL-3 zahrnuje sadu porovnávacích operátorů, které mohou být použity jakýmkoliv předdefinovaným datovým typem. Například operátor ">=" se používá k určení, zda je hodnota větší nebo stejná jako hodnota za tímto operátorem. Je též možné definovat operátory, které mohou být používány s existujícími datovými typy a uživatelem definovanými datovými typy. V tomto ohledu jsou operátory stejné jako funkce se dvěma parametry. Například bude existovat syntaxe "<číslo1> + <číslo2>". Znaménko plus vykoná přidání prvního čísla k druhému. Výsledek je vytvořen tak, že bude připojen na další proměnnou (například <číslo3>). Tato vlastnost je stejná jako schopnost funkce vrátit hodnotu, která může být připojena k proměnné. Definice operátoru se provádí specifikací názvu pro reprezentaci tohoto operátoru.

```
create operator
binding !$
to spocitej_zisk;
```

```
create function spocitej_zisk(float, float)
    returns float as
    select (1$ - 2$);
```

Zde je tento název (reprezentační označení) "\$!".

```
select i.polozka
from inventory
where spocitej_zisk(i.cena_polozky, i.cena_nakladu) > 100
```

Pak je možné vsadit (svázat) toto označení operátoru do funkce, která popisuje úkol vykonaný operátorem. Jakmile je jednou operátor vytvořen, je možné ho zavolat dvěma způsoby. Prvním způsobem je vykonání SQL dotazu, kde se volá daná funkce se svými parametry.

Nebo je možné nahradit funkci označením operátoru tak, že se operátor umístí mezi oba parametry.

```
select i.polozka
from inventory
where i.cena_polozky !$ i.cena_nakladu > 100;
```

3.2. Plně objektově-relační přístup

Aby byl přístup plně objektově-relační, musí SŘBD podporovat:

- dynamické propojení (dynamic linking),
- aktivace klienta/serveru,
- zabezpečení,
- zpětné volání,
- uživatelsky definované přístupy k metodám,
- neomezeně dlouhé datové typy.

Dále musí také tento přístup zahrnovat funkce charakteristické pro objektově orientovaný přístup. Těmito funkcemi jsou především:

- polymorfismus (polymorphism, nutnost pozdního svázání funkčních volání)
- zapouzdření (encapsulation, skrývání informací)
- dědičnost (inheritance, sdílení vlastností, které charakterizují třídu všemi jejími podtřídami)

V této kapitole jsou popsány jednotlivé funkce objektově relačního přístupu, tyto funkce korespondují se základními vlastnostmi uvedenými výše. Více o vlastnostech viz [MER02].

3.2.1. Dynamické propojení

Dynamické propojení povoluje uživatelem definovaným funkcím propojení s ORSŘBD v čase běhu. Takovéto propojení má několik výhod. Uživatelem definované funkce, které jsou propojeny staticky s ORSŘBD, musí být nahrány do paměti kdykoliv se spustí databáze. V takovém případě se zvyšuje počet adres (adresových míst) vyžadovaných pro běh SŘBD, v závislosti na množství těchto statických propojení funkcí. Dynamicky propojené funkce nevyužívají adresové místo až do té doby, kdy jsou opravdu potřeba. Dynamické propojení povoluje uživateli jednoduše vkládat nové datové typy, funkce a operátory. To je důležité hlavně pro funkce, protože ty se optimalizují od svého prvního vývoje (implementace) se svým používáním. Kvůli statickým propojením uživatelem definovaných elementů se musí systém restartovat, to pak negativně ovlivňuje flexibilitu ORSŘBD a vytváří to nepříjemně náročnou implementaci uživatelem definovaných elementů.

ORSŘBD by měl podporovat funkce aktivace jak na straně serveru, tak na straně klienta. Tyto funkce mohou být nápomocny při nutnosti přepnutí z jednoho typu na jiný v závislosti na stávajících specifikacích sítě a zatížení. Funkce aktivovaná na serveru na stejné adrese se nazývá lokální procedurální volání. Jestliže je funkce aktivovaná v jiném adresovém místě nebo na jiném stroji, jako např. na klientu, funkce je pak označována jako vzdálené procedurální volání.

Pro příklad funkce "spocitej_zisk". Funkce přijímá dva parametry, které musí být relevantní (ve správném pořadí, tvaru atp.), když je funkce zavolána. Jestliže je funkce aktivovaná vzdáleným procedurálním zavoláním, hodnota každého parametru musí být zkopírována do jiné adresy. To ale může stát spoustu času a čím více je učiněno volání tohoto typu, tím drastičtěji je ovlivněn výkon. Na druhé straně je relativně málo času stráveného na používání lokálního procedurálního zavolání, protože daný proces se vyskytuje na stejném adresovém místě. Funkce "spocitej_zisk" obsahuje pouze jednu procedurální řádku kódu, takže je to velmi krátká funkce. Jakmile se ale množství kódu ve funkci zvětšuje, zcela očekávaně rostou nároky na výpočty. V objemných funkcích je proces vytváření vzdáleného procedurálního volání bezvýznamný ve srovnání s nároky na vykonání funkce.

Nyní to tedy vypadá, že všechna volání funkcí by měla být realizována přes lokální procedurální volání. Nicméně aktivování funkcí na stejných adresách s sebou přináší nepříjemné důsledky v podobě omezené bezpečnosti. Vzdálené volání funkcí je prováděno procesem s ID (identifikace) odlišným od uživatelského ID v ORSŘBD. Pokud je tedy volání prováděno od klienta, probíhá právě takováto identifikace. Lokální volání funkce naopak běží v procesu s uživatelským ID ORSŘBD. Kdokoliv může záměrně vytvořit chybu a nepovolená instrukce může být náhodně zakódovaná do funkce. Pokud je funkce aktivována z klienta (tedy vzdáleným voláním), může nepatřičný kód způsobit, že se klientovi databáze zhroutí. Pokud ale je tato chyba v lokálním procedurálním volání, může se zhroutit celá databáze, nejen pouze klient, a v extrémním případě i celá síť.

Existují tři techniky, které je možné použít ke zlepšení bezpečnosti při použití aktivace na straně serveru [TOR99].

- 1) První technika zahrnuje běh funkcí z odlišných adres z ORSŘBD. Je nutné si připomenout, že zabezpečení funkce a ORSŘBD mohou mít odlišné ID a adresy i když jsou na stejném serveru. To pak vlastně znamená, že se realizuje vzdálené procedurální volání, a tak je princip stejný, jako by funkce byla aktivována od klienta.
- 2) Druhá technika znamená použití Firewall. Nějaký program ve Firewall může mít za úkol prozkoumávat uživatelem definované kódy funkcí. Program může být schopen modifikovat funkci v případě, že objeví ve funkci kód, který by neměl být povolen k provedení.
- 3) Třetí technika je o psaní funkcí v překládaných jazycích namísto v kompilátorech. Při běhu jsou překládané funkce prováděny důkladněji než funkce přes kompilátory. K nim přivázané kontroly jsou vynuceny na každé generované adrese. Kompromisem pak je, že se použitím překládaných funkcí o mnoho zhorší výkon.

3.2.2. Zpětné volání (callback)

Dobrý ORSŘBD by měl umožnit používat dotazy zevnitř funkce. Tato schopnost se nazývá zpětné volání (callback).

Pro příklad uvažujme kontrolu otisků prstů prováděnou orgány činnými v trestním řízení. Otisk prstů je naskenován do PC a vytvoří se SQL dotaz. Dotaz obsahuje funkci, která prohledává daktyloskopickou databázi, aby našla stejný otisk s naskenovaným. Funkce hledání v databázi bude obsahovat kód, který otevře soubor a porovná každý zaznamenaný otisk, zdali je shodný s naskenovaným. Po ukončení hledání jsou data navraceny a zpracovány dotazem. Pro účinnou funkci callback v prostředí ORSŘBD je nutné, aby uživatelem definovaná funkce byla schopna dovolat se službě SŘBD používající stejné API (aplikační programové rozhraní) jako má klientský aplikační program. Funkce bude pracovat stejně, pokud je provázána na aplikační program nebo povolána SŘBD.

3.2.3. Uživatelem definované přístupy k metodám

Objektově relační SŘBD by měl být schopen provádět hledání za použití jakéhokoliv typu přístupu, který je vyhovující pro uložená data. Použití B-stromu² je vhodné pro dotazy relačních SŘBD, tedy pokud jsou v databázi uloženy pouze jednoduše uložitelné datové typy SQL-92. Složitost datových typů povolených v ORSŘBD znamená snížení použitelnosti vyhledávání jednodimenzního B-stromu.

Pro příklad uvedu výrobce nábytku z předchozích kapitol. Databáze má uloženy lokace každé budovy v tabulce. Při použití SQL-92 je možné simulovat geografický datový typ ukládáním lokace do dvou čísel. V takovém případě je možné využít B-strom k vyhledávání jak podle zeměpisné šířky, tak podle délky. Nicméně samotné hledání nikdy nevrátí přesnou lokaci budovy. Pro vykonávání užitečných hledání geografických bodů je zapotřebí metody, která je schopna pracovat se dvěma dimenzemi. Tyto role mohou sehrát metody prostorového přístupu, jako R-strom (regionální strom). Od doby, co SQL-3 dovoluje vytvářet nové datové typy je možné, že nebude existovat žádná metoda přístupu, která by byla schopná odpovědět na konkrétní dotaz. ORSŘBD by měl uživateli dovolit přidávat nové přístupové metody, které jsou požadovány pro funkčnost. Vytváření takovýchto metod jsou ovšem komplexní programátorské úkoly, které jsou obvykle vykonávány profesionálními programátory.

² B-strom (B-Tree): B-strom je metoda umístování a lokalizování záznamů v databázi (význam písmena B nemá explicitní význam). Algoritmus B-stromu minimalizuje počet přístupů na médium pro lokalizování hledaného záznamu, tím zvyšuje rychlost procesu.

3.2.3.1. Délka datových typů

Dalším problémem týkajícím se datových typů definovaných uživatelem je délka. V ORSŘBD by neměla existovat omezení v délce datových typů, přesto by měl být brán ohled na délku v souvislosti s objemem dat, který je schopen ORSŘBD ovládat. Je možné mít datové typy SQL-3, které jsou krátké a určené délky, jako například geografické body v databázi výrobce nábytku. Je také možné mít dlouhé datové typy stálé délky, jako obrázky (nebo otisky prstů ve vládní databázi). Je možné používat také dlouhé datové typy s proměnnou délkou, jako například komprimované obrázkové soubory. Zde je nutné podotknout, že i relační SŘBD může ukládat libovolně dlouhá data jako BLOB³. Ale BLOB nejsou považovány za pravé datové typy, protože mohou být pouze ukládány a znovu vyvolávány. Informace v nich není možné prohlížet strojem relačních SŘBD.

3.3. Komplexní objekty

Jeden ze základních požadavků objektově relačního SŘBD je podpora komplexních objektů. Komplexní objekty mohou být skládány například z několika uživatelem definovaných datových typů. Mohou reprezentovat komplexní interní struktury, atributy a chování. Aby bylo možné vytvářet komplexní objekty a aby se s nimi mohlo manipulovat, je nutné mít možnost konstruovat nové uživatelem definované datové typy (UDT), a definovat nové uživatelem definované funkce (UDF). ORSŘBD podporuje tři hlavní druhy konstruktorů pro vytváření komplexních typů [TOR99]:

- složené datové typy,
- sady,
- reference (odkazy).

3.3.1. Složené datové typy

Složené datové typy se skládají z kolekce hodnot. Například je třeba vytvořit nový datový typ nazvaný "adresa_t". Vytvoříme ho označením pojmenování typu a názvu a typy atributů, které budou v typu zahrnuty.

³ BLOB (binary large object), je veliký objekt (typicky obrázek nebo hudební vzorek), se kterým je možno manipulovat (například ukládat, stahovat z databáze) za speciálních podmínek, kvůli jeho velikosti.

```
create type adresa_t (  
    cislo    int,  
    ulice    varchar(30),  
    ctvrt    varchar(20),  
    mesto    mesta_t);  
  
create type mesta_t (  
    nazev    varchar(20),  
    populace int,  
    oblast   int);
```

Atribut uvnitř složeného typu může být sám složený typ. Je možné vytvořit tabulku, ve které budou drženy instance složeného typu. V následujícím příkladu je vytvořena tabulka nazvaná “adr”, ve které budou udržovány instance typu “adresa_t”.

```
create table adr of type adresa_t
```

Vytváření typů a jejich připojování k tabulkám připouští flexibilitu v tom, že je možné mít více tabulek, které drží instance stejného typu. Dobrý ORSŘBD připouští definovat samostatné sloupce jako instance složeného typu. Například vytvoření tabulky “zakaznici”, která má v sobě sloupec pro ukládání zákaznických adres, kde právě zákaznická adresa je složený typ “adresa_t”.

```
create type adresa_t (  
    cislo    int,  
    ulice    varchar(30),  
    ctvrt    varchar(20),  
    mesto    mesta_t);  
  
create table zakaznici (  
    jmeno    varchar(30),  
    adresa    adresa_t);
```

3.3.2. Sady

Jako typ konstruktoru v ORSŘBD je také možné použít sady. Existuje možnost vytvářet nové sady typů z jakékoliv existující báze nebo složeného typu. Následující obrázek definuje jako typ atributu adresy složený typ “lokalni_zakaznik_t”, aby byl sadou složeného typu “adresa_t”.

```
create type lokalni_zakaznik_t (  
    lz_jmeno    varchar(30),  
    adresa       setof(adresa_t));
```

To umožňuje sadě adres (například domácích a obchodních) být uloženy pro každého zákazníka. Při setkání se sadami objektů může být počet členů velmi vysoký, a to způsobí odpovídající změnu ve velikosti záznamů. Pro dobré řízení databáze je vyžadováno, aby uživatel nastavil prahy (hranice) pro velikosti sad. Prvky sady rostoucí příliš rychle mohou být ukládány samostatně mimo záznam, do kterého patří. Záznam pak bude obsahovat odkaz na místo, kde se nacházejí konkrétní členové.

3.3.3. Odkazy (reference)

Jako typ konstruktorů je také možné použít odkazy. Stejně jako je u sad vytváření odkazu jako složeného typu efektivní vytvoření dobře použitelný nový typ. Odkaz je použit pro definování vztahů mezi složenými typy. Odkazy mohou být používány pro jednoznačnou identifikaci složených typů nebo řádků v rámci celé databáze. Odkazy je také možné vytvářet pro sady. Následující příkaz definuje uživatelem definovaný typ “spolecnost_t”.

```
create type spolecnost_t (  
    nazev          varchar(30),  
    adresa         adresa_t,  
    telefon        telefon_t,  
    ceo            varchar(30),  
    ceo_ref        ref(snr_pracovnik_t));
```

Atribut “ceo” je pro jméno CEO (chief executive officer, výkonný manažer) společnosti. Definovaný atribut “ceo_ref” je odkazem zajišťující cestu k více detailním datům CEO. Tato detailní data jsou uchována v tabulce “snr_pracovnik_t”. Takovéto používání odkazů zajišťuje optimalizaci ORSŘBD jinými cestami navigací dat v databázi. Odkazy umožňují nahradit složité vkládání jednoduššími cestami.

Existují i jiné konstruktory typů, které mají omezenou použitelnost v ORSŘBD. Tím jsou myšleny především:

- seznamy (list),
- zásobníky (stack),
- fronty (queue),
- pole (array),
- vložitelné pole.

To jsou konstruktory typů mající stejnou interpretaci jako jejich ekvivalenty v programovacích jazycích. Vložitelné pole je speciálním typem pole, kde vložení způsobuje dynamické zvýšení délky pole.

3.3.4. Manipulace se složenými datovými typy

Manipulace se složenými datovými typy vyžaduje některé dodatky (rozšíření) ke standardnímu SQL-3. Prvním z nich je umožnění složeným typům, aby byly použitelné jako parametry v uživatelem definovaných funkcích. Pro příklad vezměme transportní společnost, kde je “povoleni_t” složeným typem definovaným na své databázi.

```
create type povoleni_t (  
    drzitel varchar(30),  
    tridy   varchar(6),  
    dat_plat    date,  
    dolozky int);  
  
create type ridic_t (  
    jmeno      varchar(30),  
    adresa     varchar(40),  
    telefon    telefon_t,  
    rid_prukaz prukaz_t,  
    vek        int);  
  
create table ridici of type ridic_t;
```

Pro jednoduchost již je definována funkce “konec_platnosti”, která indikuje, kdy proběhlo datum vypršení daného povolení vrácením výsledku boolean (tedy pravda/nepravda). Můžeme jí pak použít v SQL příkazech:

```
select jmeno, adresa  
from ridici  
where konec_platnosti (rid_prukaz);
```

Uživatелеm definované funkce by měly být také schopny vracet výsledky, které jsou instancemi složených typů. To vyžaduje dodatek (rozšíření) standardního SQL, který povolí tyto druhy funkcí. Konkrétně do příkazu “from” v rámci SQL příkazu. V následujícím příkazu je definována funkce “pres50”, která vrací složený typ “povoleni_t” pro všechny řidiče starší padesáti let.

```
create function pres50  
returns povoleni_t as  
select rid_prukaz  
from ridici  
where vek >= 50;
```

Takovou funkci je možné použít ve “from” v dotazu jako je tento:

```
select drzitel, dolozky  
from pres50()  
where dat_plat < '2003-01-01'
```

Dotaz vrací jméno a doložku všech padesátiletých řidičů a starších, kterým vypršela platnost řidičského oprávnění před lednem 2003.

V tradičním SQL se na jednotlivé sloupce tabulky odvoláváme následující syntaxí:

- `jmeno_tabulky.jmeno_sloupce`

Pro to je potřeba třetí dodatek SQL, který umožní používat kaskádovanou syntaxi tečky, v situacích odkazování na atributy složeného typu. Syntaxe je následující:

- `jmeno_slozeneho_typu.jmeno_atributu`

K držiteli a datu vypršení platnosti řidičského průkazu pro řidiče starší nebo staré 50 let je možné přistupovat použitím tohoto dotazu.

```
select rid_prukaz.konec_platnosti
from ridici
where vek >= 50;
```

Rámcem tohoto dotazu je tabulka řidičů a vyžadován je sloupec „rid_prukaz“. Tečka zde indikuje, že žádoucí informací pouze držitel a konec_platnosti v rámci tohoto složeného typu. Kaskádovaný způsob tečky by měl být také přístupný pro kódování kvalifikace v podmínkách dotazu "where".

```
select jmeno
from ridici
where rid_prukaz.dolozky > 1;
```

Tento dotaz vrátí jména všech řidičů, kteří mají více nežli jednu doložku na řidičských průkazech.

O složených datových typech lze hovořit také jinak. Raději nežli jako na kolekci atributů by na ně mělo být nahlíženo jako na typy mající kolekci funkcí – jednu pro jeden atribut. Každá funkce má stejné jméno jako jí korespondující atribut. Manipulace s těmito funkcemi je možná použitím standardní funkční syntaxe, například:

- `jmeno_atributu(jmeno_slozeneho_typu)`

Při konstruování dotazů je tedy možné požit tuto interpretaci.

```
select jmeno
from ridici
where dolozky(rid_prukaz) > 1;
```

Zde budou vrácena jména všech řidičů, kteří mají více nežli jeden záznam. Další možná variace je v umístění názvu funkce napravo od jejích parametrů za použití tečky namísto závorek.

Dodatky SQL vyžadované pro práci se složenými datovými typy mohou být využité i pro sady složených datových typů. Existuje složený typ “ridic_t”, který má atribut “telefon” definovaný jako sada složených typů “telefon_t”.

```
create type permit_t (  
    drzitel      varchar(30),  
    tridy        varchar(6),  
    dat_plat     date,  
    dolozky      int);  
  
create type ridic_t (  
    jmeno        varchar(30),  
    adresa       varchar(40),  
    telefon      setof(telefon_t),  
    rid_povoleni povoleni_t,  
    vek          int);  
  
create teble ridici of type ridic_t;
```

To umožňuje mít více telefonních čísel pro každou instanci “ridic_t”. Například každý řidič může mít své služební číslo, číslo domů, mobilní číslo a kontakt při pohotovosti. Pokud bude použito příslušné rozšíření SQL, je možné specifikovat sadu složených typů jako parametr funkce a zároveň mít funkci, která vrací sadu složených typů. Je možné vytvořit funkci nazvanou “domaci_uzel_353”, která vrací boolean, zda je řidičovo domácí číslo z uzlu 353. Je pak možné použít tuto funkci v rámci SQL dotazu, který například hledá jména všech řidičů, jejichž domácí telefonní čísla jsou v uzlu 353. Funkce je zasazena s parametrem “telefon_t”, jako svým parametrem.

```
select jmeno  
from ridici  
where domaci_uzel_353 (telefon);
```

Pro tento úkol je možné použít i variantu s tečkou (telefon.domaci_uzel). Například výběr jmen všech řidičů, kteří žijí v uzlu 353. Zde “ridici.telefon” určuje sadu.

```
select jmeno  
from ridici.telefon  
where '353' in telefon.domaci_uzel;
```

V případě ukládání složených typů a sad složených typů nelze pominout fakt, že v rámci databáze mohou existovat několikanásobné kopie stejného objektu. To znamená vyšší náročnost pro udržení integrity dat v databázi. Z tohoto důvodu by měli být složené typy a sady složených typů používány pro data, která se nemění příliš často, nebo se objevují pouze jednou.

3.3.5. Používání odkazů

Tradiční relační databázové systémy fungují na vztazích primární klíč-cizí klíč. Odkazy znamenají alternativu k těmto vztahům. Například bude existovat databáze pro továrnu mající dvě tabulky. "Sekce", typu "sekce_t", "pracovník" typu "pracovník_t". Tabulka sekce má jako primární klíč nastaven "nazev_sekce". Atribut supervizor je cizí klíč. Obsahuje jméno pracovníka, který funguje v sekci jako supervizor.

```
create type pracovník_t (  
    jmeno_prac    varchar(30),  
    nazev_sekce   varchar(20),  
    poc_dat       date,  
    vydelek       int,  
    vek           int);  
  
create table pracovník of type pracovník_t;  
  
create type sekce_t (  
    nazev_sekce   varchar(20),  
    supervisor     varchar(30),  
    telefon_ext    varchar(3),  
    pocet_prac     int,  
    stroje         int,  
    umisteni       int);  
  
create table sekce of type sekce_t;
```

Nyní bude vytvořen dotaz, který vrátí detailnější informace o určitém supervizorovi. Tradiční přístup primární-cizí klíč bude vypadat takto:

```
select pracovník.vydelek, pracovník.vek  
from pracovník, sekce  
where pracovník.jmeno_prac = sekce.supervisor  
    and sekce.nazev_sekce = 'dokoncovani';
```

Dotaz vyvolá výdělek a věk supervizora v sekci "dokoncování". Existence spojení tabulek v dotazu jej činí pomalým a vlastně drahým. Alternativa k použití spojení v dotazu spočívá právě v použití odkazu. S odkazem může jedna tabulka obsahovat hodnotu, která ukazuje na výskyt složeného typu uloženého v jiné tabulce. Následuje úprava předchozího příkladu a do tabulky sekce je nyní zařazen odkaz na typ "pracovník_t".

```
create type pracovník_t (  
    jmeno_prac    varchar(30),  
    nazev_sekce   varchar(20),  
    poc_dat       date,  
    vydelek       int,  
    vek           int);  
  
create table pracovník of type pracovník_t;  
  
create type sekce_t (  
    nazev_sekce   varchar(20),  
    supervisor     varchar(30),  
    supvr_ref      ref(pracovník_t),  
    telefon_ext    varchar(3),  
    pocet_prac     int,  
    stroje         int,  
    umisteni       int);  
  
create table sekce of type sekce_t;
```

Podpora těchto odkazů vyžaduje metody jednoznačné identifikace jednotlivých řádků v tabulce. Obecně jednoznačný identifikátor objektu (OID) je asociován s každou řadou každé tabulky. Je to hodnota, která je uložena v atributu typu odkaz (reference). Podpora pro odkazy (reference) vyžaduje dodatky (rozšíření) standardního SQL.

- 1) První z nich je zprostředkování speciální dereferenční funkce (DEREF), která přijímá odkaz na záznam a vrací aktuální záznam.
- 2) Druhým je podpora pro funkce, která akceptuje odkazy jako parametry a vrací odkazy jako výsledky. Aplikujme například tento jednoduchý dotaz na databázi továrny:

```
select supvr_ref
from sekce
where nazev_sekce = 'dilna'
```

Dotaz vrátí odkaz na pracovníka supervizora sekci “dilna”, nikoliv aktuální záznam pracovníka. Pro vrácení aktuálního záznamu pracovníka je nutné použít “deref” na atribut reference.

```
select deref(supvr_ref)
from sekce
where nazev_sekce = 'dilna'
```

Funkce “deref” musí fungovat pro všechny typy odkazů. Může být použita i tečková notace ve spojení s “deref”. Například je možné zkonstruovat dotaz, který vrací počáteční datum supervizora na sekci “dokončování”. Implikace funkce “deref” vyžaduje potřebu opětovné inverze funkce “deref” a inverzní “deref” funkce (“ref”) akceptuje aktuální záznam jako parametr a vrací ukazatel (tedy odkaz)

```
select ref(deref(supvr_ref))
from sekce
where nazev_sekce = 'dilna';
```

na tento záznam.

Nyní následuje použití funkce “ref” k zaznamenání změny supervizora v sekci “dilna”.

```
update sekce
set supvr_ref =
    (select ref(pracovnik)
     from pracovnik
     where jmeno_prac = 'Zeleny')
where nazev_sekce = 'dilna'
```

Zde je atribut pro sekci dilna „supvr_ref“ ukotven na ukazatel v tabulce pracovník, konkrétně záznam „Zeleny“. Používání odkazů jako náhrady za vztahy primární-cizí klíč nabízí zvýšenou bezpečnost, protože je zaručeno, že jsou jednoznačné a v čase neměnné. Například příjmení zaměstnankyně by se mohlo změnit ve chvíli, kdy se vdá, atp. Reference samotná tedy neobsahuje pro uživatele žádné užitečné informace. Proto, aby získala smysl, musí být dereferencována.

Pro ORSŘBD je logické, že podporuje koncept sad pro podporu sad referencí. Pro příklad bude do tabulky sekce zahrnut atribut „pracovníci“ na příkladu databáze továrny. Sloupec bude zaznamenávat všechny pracovníky v každé sekci.

```
create type sekce_t (
    nazev_sekce    varchar(20),
    supervisor     varchar(30),
    supvr_ref      ref(pracovnik_t),
    telefon_ext    varchar(3),
    pocet_prac     int,
    pracovníci     setof(ref(pracovnik_t)),
    stroje         int,
    umisteni       int);

create table sekce of type sekce_t;
```

Toto pole bude obsahovat odkaz na záznam každého pracovníka zaměstnaného v nějaké sekci. Celá kolekce referencí pak utváří sadu. Je nutné mít možnost manipulovat se sadami odkazů stejným způsobem jako s jinými sadami. Například níže vytvořený dotaz je použitelný pro získání detailů všech pracovníků zaměstnaných v sekci "dilna".

```
select deref(*)
from (select pracovníci
      from sekce
      where nazev_sekce = 'dilna');
```

Výsledkem deref(*) v příkazu select znamená dereferenci členů vrácené sady z klauzule from. Pokud je třeba znát pouze věk každého pracovníka sekce "dilna", je možné v rámci select příkazu použít tečkovou notaci. Sada odkazů znamená stejné problémy s integritou dat jako problémy spojené s jinými sadami objektů. jakákoliv změna v datech musí být přenesena přes všechny výskyty dat v databázi.

3.3.6. Základní typy a komplexní objekty

Dobré ORSŘBD zajišťují dvě možnosti rozšíření systému [TOR99]:

- 1) Uživatelem definované základní typy a k nim přiřazené funkce,
- 2) uživatelem definované komplexní objekty a funkce.

Objektově relační SŘBD musí podporovat základní typy a komplexní objekty v jednom systému. Fakt, že základní datové typy jako datum nebo další byly začleněny do většiny komerčních SŘBD, znamená, že i dobrý ORSŘBD by měl podporovat (kromě standardních základních typů) i tyto základní typy.

Druhý důvod pro podporu základních typů a komplexních objektů je věnován zapouzdření. Základní typy jsou kompletně zapouzdřeny, složené typy jsou celkově transparentní. Jediná manipulace možná se základním typem je jeho vyvolání(náhled) nebo vykonání funkce, která jej pojímá jako parametr.

Naproti tomu s poli složených objektů může být individuálně manipulováno za použití dotazovacího jazyka. Na zapouzdřené typy je nahlíženo jako na privátní, kdežto na transparentní typy jako na veřejné. Vnitřní reprezentace základního typu může být změněna bez ovlivnění programů, které ho používají. Je třeba pouze předefinovat navázané funkce vstupu a výstupu. Na druhou stranu, transparentní pole ve složeném typu nemohou být upravována bez úpravy všech programů, které je používají. Existuje obecné tvrzení, že základní typy jsou vhodné pro zapouzdřená data a komplexní objekty jsou vhodné pro nezapouzdřená data.

Třetím důvodem pro podporování dvou typů je použití identifikátorů objektu (OID). Ty jsou rozmístěny systémem a jsou používány k jednoznačné identifikaci řad (záznamů) uvnitř tabulek. Obecně – základní typy nemají systémem alokované OID, ty jsou pouze dostupné pro komplexní typy. Konverze a třídění dat znamená další dobrý důvod pro podporu základních typů a komplexních objektů v rámci jednoho systému. Existuje spousta datových typů, které potřebují rozdílné externí a interní formáty (například číslo - integer). Vstupní nebo výstupní funkce pro základní typy přirozeně podporují konverzi formátu a uživatelem definované znovuspořádání. SŘBD, který nepodporuje základní datové typy musí umožňovat konverzi formátu a uživatelem definované znovuspořádání trochu jiným způsobem.

3.4. Dědičnost

Dobrá objektově relační SŘBD umožňuje podporu pro dědičnost atributů existujících datových typů. Ve spojení se základními typy a komplexními objekty dovoluje dědičnost definovat nové datové typy a vytvářet schémata (nebo struktury), které úzce modelují a reprezentují požadavky uživatele. Dědičnost dat se omezuje na složené typy. Nicméně také základní typy mohou existovat v hierarchii dědičnosti, stejně jako mohou dědit funkce. Když se vytváří základní typ, je možné specifikovat, že je dědicem nějakého základního supertypu. Vytvoříme datový typ „pracovník_t“, který má čtyři atributy: příjmení, jméno, adresa, věk. Je možné považovat „pracovník_t“ za supertyp použitelný pro tvorbu dalších subtypů. Například je možné použít SQL k vytvoření podtypů „supervisor_t“ a „hodin_mzdy_t“.

```
create type pracovník_t (  
    příjmení    varchar(20),  
    jméno      varchar(10),  
    adresa     varchar(30),  
    věk        int);  
  
create type supervisor_t (  
    poc_dat     date,  
    měsíční_příjem int,  
    naklady     float)  
under         pracovník_t;  
  
create type hodin_mzdy_t (  
    hodiny      int,  
    hodinová_sazba float,  
    dávky       float)  
under         pracovník_t;
```

Každý nově vytvořený podtyp automaticky zdědí všechny elementy dat supertypu. Například "supervisor_t" dědí atributy příjmení, jméno, adresu a věk od svého supertypu. Tento subtyp také specifikuje tři další atributy. Dobrá objektově relační SŘBD také podporuje dědičnost elementů dat z více databází. Tomu se říká násobná dědičnost a může být rozvržena do několika úrovní. V následující situaci je vytvořen podtyp pod existujícími typy "supervisor_t" a "hodin_mzdy_t". Tento nově vytvořený typ dědí elementy dat ze svých rodičů a k tomu má ještě jeden vlastní, "odd". Příjmení, jméno, adresa a věk jsou již zděděné elementy v rámci "supervisor_t" a "hodin_mzdy_t" a tyto elementy budou obsaženy i v novém subtypu.

```
create type supervisor_hodin_mzdy_t (  
    odd          varchar(10))  
under supervisor_t, hodin_mzdy_t;
```

S dědičností dat je možné seskupovat složené typy do hierarchie dědičnosti. Datové typy vespod hierarchie by měli dědit většinu svých atributů ze supertypů na vyšších úrovních. Tento modulární přístup může být používán ke zlepšení efektivity a udržení konzistence v elementech dat.

V případě, že datový typ zdědí pole z více nežli jednoho supertypu, vzniká problém dvojmyslu. V případě, kde supervisor_t a hodin_mzdy_t budou mít oproti příkladu výše každý ještě atribut "zabezpečení" každý jiného typu, vznikne dvojmysl pro "supervisor_hodin_mzdy_t", který je

dědicem těchto dvou. Jedním z řešení této dvojsmyslnosti je v nepovolení definice typu, který by mohl způsobit výskyt dvojsmyslu. Práci na pravidlech pro rozpoznávání dvojsmyslu by měl zastávat administrátor. Nicméně tato pravidla mohou být komplexní a obtížně pochopitelná pro uživatele.

Dědičnost může být aplikována pouze na pojmenované datové typy. Vytváření tabulky, která není z pojmenovaných datových typů, neumožní použití dědičnosti.

Lepší přístup ve vytváření tabulek je zkonstruovat nejdříve typy a pak je navázat do tabulek. Pro systém to pak znamená velkou flexibilitu.

```
create table worker (  
    prijmeni    varchar(20),  
    jmeno      varchar(10),  
    adresa     varchar(30),  
    vek        int);
```

Existuje možnost vytvářet mnoho tabulek ze stejného typu (např. worker_t). To umožňuje hierarchii tabulek být odlišnou od typu hierarchie používané pro definování tabulek. V kontrastu s tím ale

```
create table pracovnik of type pracovnik_t;
```

hierarchie tabulek a jejich korespondující typ hierarchie bude stejný v systémech, které umožňují pouze jednu tabulku pro určitý typ.

Následující příklad vytváří hierarchii tabulek za použití specifikovaných typů "pracovnik_t", "supervisor_t", "hodin_mzdy_t" a "supervisor_hodin_mzdy_t".

```
create table pracovnik of type pracovnik_t;  
create table supvr of type supervisor_t under pracovnik;  
create table hodiny of type hodin_mzdy_t under pracovnik;  
create table supvr_hodiny of type supervisor_hodin_mzdy_t  
under supvr, hodiny;
```

Obecně by mělo být možno odkazovat na jakoukoliv tabulku a mít rozsah SQL příkazů přednastavených pro tabulku a všechny tabulky pod ní. Například tento příkaz bude prozkoumávat tabulku "pracovnik" a hledat všechny jedince, kterým je 30.

```
select prijmeni  
from pracovnik  
where vek = 30;
```

Tento příkaz bude ale také prohledávat všechny tabulky pod tabulkou "pracovnik" v tabulkové hierarchii. Pokud je nutné omezit se pouze na tabulku "pracovnik", je třeba použít příkaz takto:

```
select prijmeni  
from only (pracovnik)  
where vek = 30;
```


Odlišné implementace dědičnosti jsou nabízené odlišnými výrobci SŘBD a mají též rozdílné výkonnostní charakteristiky. Detaily v implementaci určitého ORSŘBD budou určovat efektivitu designu pro databáze.

3.4.1. Dědičnost funkcí

Opravdová síla objektově relačních SŘBD plyne z dědičnosti funkcí. Chování dědičnosti uživatelem definovaných funkcí bude záviset na parametrech funkcí. Je možné uvažovat o dědičnosti funkcí jako napojení se ke každému uzlu v hierarchii dědičnosti jmen všech funkcí s datovým typem jako parametrem.

```
create function mensi_vydelek (supervisor_t)
returns      boolean as
return      $1.mesicni_prijem < (select mesicni_prijem
                                from supvr
                                where prijmeni = 'Dvorak'
                                and jmeno = 'Jan');
```

Dobrým příkladem bude předchozí případ čtyř typů, kde jsou již vytvořené tabulky, které drží instance. Nyní bude vytvořena funkce nazvaná "mensi_vydelek", která indikuje jestli má jednotlivec nižší plat nežli supervisor Jan Dvorak.

Z definice funkce je patrné, že výskyt typu supervisor_t je jediným parametrem funkce. Funkce vrací hodnotu boolean true (pravda), jestliže je měsíční plat pracovníka nižší nežli supervisor Jan Dvorak. Je zřejmé, že tato funkce je přivázána k uzlu supervisor_t v hierarchii typů. Nyní je možné tuto funkci použít v dotazu. Následuje jednoduchý dotaz.

```
select s.prijmeni
from only (supvr) s
where mensi_vydelek (s);
```

Výraz "only" identifikuje, že rozsah bude pouze tabulka "supvr". Nicméně když je "only" vynecháno, rozsah příkazu bude rozšířen i na podřízené tabulky. Ačkoliv funkce lowearner může být jasně spočítána z tabulky supvr, není jasné, jak bude vypočítána z podřízených tabulek. Pokud není dána definice konkrétní funkce na typu, ORSŘBD by se měl pokusit automaticky zdědit funkci ze supertypu. V tomto použitém případě ORSŘBD použije funkci "mensi_vydelek" definovanou na supertypu "supervisor_t". Ve výsledném efektu pak bude "supervisor_hodin_mzdy_t" dědit funkci ze "supervisor_t".

Ještě jednou případ vytvoření funkce "mensi_vydelek" na typu "supervisor_t". Funkce bude vytvářet hodnotu pro každý výskyt v "supervisor_t". Pak je možné podívat se na virtuální atribut v datovém typu "supervisor_t". Jako takový může být děděn v uzlech shora dolů v hierarchii dědičnosti. To znamená, že na funkci dědičnosti v objektově relačním SŘBD může být nahlíženo ve smyslu datové

dědičnosti. Tato interpretace umožňuje odkazy funkcí jako na normálních attributech složeného typu. Například je možné použít dotaz tímto způsobem:

```
select s.prijmeni  
from supvr s  
where s.mensi_vydelek;
```

Zde tabulka supvr dědí funkci "mensi_vydelek" z typu "supervisor_t". Písmeno "s" je učeno jako alias pro tabulku supvr. Přístup virtuálním atributem je indikován umístěním argumentu nejdříve v dotazu za použití tečkové notace. Tento dotaz vrátí všechny vstupy v supvr tabulce, kde jsou výdělky nižší nežli u Jana Dvoraka.

Při zapisování dotazů je možné vkládat uživatelem definované funkce jako virtuální atribut nebo jako funkci.

3.5. Polymorfismus

V objektově orientovaném programování znamená polymorfismus (slovo původem z řečtiny - "mající mnoho forem") možnost využití objektu jiným způsobem, v jiném kontextu. Konkrétněji to znamená více forem entit jako proměnných, funkcí nebo objektů. Obecně existuje několik druhů polymorfismu [lit27]:

- 1) Funkci je možné měnit v závislosti na parametrech, které dostává. Například jestliže parametr bude číslo, volaná funkce by mohla hledat výsledek prohledáváním čísel zaměstnanců. Pokud by to byl znakový řetězec, funkce by hledala přes jména zaměstnanců. Každopádně tato funkce bude v rámci programu pojmenována stejně. Takový druh polymorfismu se nazývá *overloading*. Vzhledem ke značné použitelnosti tohoto druhu polymorfismu je detailnějšímu popisu funkce *overloading* věnována následující podkapitola.
- 2) Proměnná může mít několik forem a program se rozhoduje, kterou formu použije pro konkrétní případ. Například proměnná pojmenovaná "userid" může být jak číslem, tak řetězcem znaků. Pokud program zná způsob jak rozeznat formu proměnné pro daný případ, je schopný s touto proměnnou pracovat.
- 3) Polymorfismus může v OO programování také znamenat datový typ "nějaký", definovaný pro specifikovaný seznam obsahující nějaké datové typy, které mohou být vykonány funkcí. Například pokud funkce definuje délku seznamu, není důležité jaké datové typy jsou v seznamu obsaženy.

3.5.1. Overloading funkcí

Dobré ORSŘBD by měli podporovat overloading funkcí. Overloading umožňuje definovat tolik verzí konkrétní funkce, kolik jich je potřeba implementovat v určité aplikaci. Být usměrňovačem definice funkce pro jednotlivé podtypy je velmi žádoucí vlastností ORSŘBD. Jako důkaz poslouží typ hierarchie skládající se z kořenového typu "vuz_t" a subtypů "offroad_t", "cizi_t", "motor_t" a "cizi_offroad_t". SQL vypadá následovně:

```
create type vuz_t (  
  nazev  varchar(20),  
  model  varchar(10),  
  cena   int);  
  
create type offroad_t (  
  prevody int,  
  delka   int,  
  sirka   int)  
under vuz_t;  
  
create type cizi_t (  
  zeme    varchar(20))  
under vuz_t;  
  
create type motor_t (  
  objem   int,  
  kw      int,  
  ventily int)  
under vuz_t;  
  
create type cizi_offroad_t (  
  spotreba float)  
under cizi_t, offroad_t;
```

Jakmile jsou typy nadefinované, je možné zkonstruovat korespondující tabulkovou hierarchii. Bude definována nová funkce, "dobra_hodnota", která určí, zda určitý offroad reprezentuje dobrou peněžní hodnotu. Definice bude vypadat následovně:

```
create function dobra_hodnota (offroad_t)  
returns boolean as  
return $1.cena < (select cena  
                  from offroad  
                  where jmeno='Skoda' and model='Superb')  
and $1.prevody > 5;
```

Kladná hodnota boolean bude vrácena v případě, že vozidlo je ohodnoceno (levnější) méně nežli Skoda Superb a má více nežli pět převodových stupňů. Jestliže definice "dobra_hodnota" je jako jediná na typu "cizi_t", pak bude dědičná pro podtyp "cizi_offroad_t". Při určování peněžní hodnoty pro cizí offroady jsou předpokládána odlišná kritéria. To vyžaduje implementaci jiné verze (overload) funkce "dobra_hodnota" pro subtyp "cizi_offroad_t". Tato druhá verze funkce "dobra_hodnota" vrací kladný boolean, když má auto aspoň čtyři stupně a nabízí spotřebu větší nežli 8.

Rozšíření druhé definice funkce nazvané "dobra_hodnota" je výsledkem zlepšené hierarchie dědičnosti. Dvě odlišné funkce nazvané "dobra_hodnota" se připojují ke dvěma uzlům v hierarchii, ve které byly explicitně definovány.


```
create function dobra_hodnota (cizi_offroad_t)
returns boolean as
return $1.gears >= 4 and $1.spotreba > 8;
```

Pokud bude spuštěn dotaz oslovující funkci "dobra_hodnota", ORSŘBD použije jednu funkci "dobra_hodnota" na tabulku "offroad" a druhou na tabulku "cizi_offroad".

```
select o.nazev
from offroad o
where dobra_hodnota (o);
```

Objektivně relační ORSŘBD podporující overloading funkcí může potřebovat násobné funkce pro vykonání jednoduchého dotazu. V praxi je povolení provádění funkce jednoho jména s odlišným smyslem (variací v definici) do více typů nazváno polymorfismus [NAH91].

Při použití overloadingu funkcí může také vzniknout dvojsmysl. Je to přímý důsledek podpory násobné dědičnosti systémem. V předchozím příkladě bude definice funkce "dobra_hodnota" na typu "cizi_offroad_t" smazána a nová definice "dobra_hodnota" přidána na typu "cizi_t". Nyní, když bude vytvořen dotaz na typ "cizi_offroad_t", vznikne dvojsmysl v tom, která funkce "dobra_hodnota" zděděná ze supertypů má být použita.

```
select c_o.nazev
from cizi_offroad f_o
where dobra_hodnota (f_o);
```

Jedním řešením problému dvojsmyslu je ohlášení run-time chyby⁴. Tento problém je možné napravit následujícími způsoby:

- přejmenováním jedné ze dvou verzí funkce,
- overloadem funkce zahrnutím nové definice na podtyp "cizi_offroad_t".

3.6. Pravidla

Plně objektově relační SŘBD potřebuje silná pravidla systému, aby byl schopen udržet integritu dat v databázi. Tato pravidla nemusí být o moc flexibilnější nežli ta, se kterými se můžeme setkat v relačních systémech. Obecná forma pravidel je "na nějakou událost proved' nějakou akci". To vyžaduje, aby ORSŘBD sledoval výskyty aktivujících se událostí. Běžně je vykonána akční část pravidla okamžitě po tom, co proběhne událost. Pružná pravidla systému umožňují akční části pravidla být vykonána i před nebo hned po tom, co je vykonána událost. ORSŘBD by měl dovolit události a akčním částem pravidel být SQL příkazem. To znamená, že ve vytváření pravidel může uživatel

⁴ Tedy chybou objevující se za běhu vykonávaného programu

použít své nadefinované funkce. Systémy podporující dědičnost dat a funkcí by měli také podporovat dědičnost pravidel. Čtyři základní variace na obecný model pravidel jsou:

- update-update,
- query-update,
- update-query,
- query-query.

Podle mého názoru by měl dobrý ORSŘBD umožňovat definici neomezeného množství těchto pravidel. Uživatel by mělo být také dovoleno konstruovat složená pravidla, která kombinují aspekty čtyřech základních typů.

3.6.1.1. Update-update

Prvním ze čtyřech základních pravidel je update-update. Zde je událost aktualizací a akce následující je také aktualizace. Pro příklad bude existovat databáze, která udržuje informace o osobních počítačích. Typická aktualizace v této databázi by mohla vypadat následovně:

```
update pc
set procesor = 'PIII800'
where vyrobce = 'Alphasys'
and model = 'Premium';
```

Nyní bude vytvořen uživatelem definovaný typ `pc_t`, který je instancí v tabulce `pc`.

```
create type pc_t (
    vyrobce varchar(30),
    model   varchar(30),
    procesor varchar(10),
    hdd      float,
    ram      int,
    cache    int);
```

```
create table pc of type pc_t;
```

Následuje příklad pravidla update-update, které by mohlo být definované na této databázi:

```
create rule novy_proc_update as
on update to pc.procesor where vyrobce = 'Alphasys'
and current.model = 'Premium'
do update pc
set procesor = new.procesor
where vyrobce = 'Alphasys'
and model = 'PremiumX';
```

Pravidlo znamená systémovou kontrolu na update typu procesoru instalovaným na model PC Premium. Pokud je detekován tento update, pravidlo způsobuje automatickou systémovou aktualizaci procesoru, který se používá na sesterském modelu Alphasys na procesor PremiumX. Potom je vlastně update jednoho modelu díky tomuto pravidlu i updatem pro model jiný. Akční klauzule je pravidlem

update-update a může být použit nějakým povoleným SQL příkazem ve spojení s klíčovými slovy "new" nebo "current". V tomto případě new.procesor ukazuje na pole v aktualizovaném záznamu v čase, kdy se událost objeví. Nastavuje tak hodnotu použitou v akční části pravidla.

Je možné vytvořit více nežli jedno pravidlo update-update pro spuštění stejné události. Například je uveden případ, kde je instalován nový PIII800 procesor v modelu Alphasys premium. Je žádoucí, aby tato aktualizace také spustila aktualizaci procesoru na Alphasys economy na PIII650.

```
create rule rip_update as
on update to pc.procesor where vyrobce = 'Alphasys'
                                and current.model = 'Premium'
do update pc
    set procesor = current.procesor
    where vyrobce = 'Alphasys'
    and model = 'Economy';
```

Nyní, když Alphasys premium model dostane procesor PIII800, tak model PremiumX dostane stejný nový procesor, a model economy automaticky dostane přiřazen procesor, který měl dříve model Premium.

Je možné také vytvořit více obecných update-update pravidel. Například je třeba vytvořit pravidlo, které automaticky aktualizuje velikost RAM na jakémkoliv Alphasys počítači, kterému je přiřazen nový procesor.

```
create rule update_ram as
on update to pc.procesor where current.ram <=16
do update pc
    set ram = current.ram + (16)
    where vyrobce = 'Alphasys';
```

Pravidla update-update jsou velmi užitečná pro zajištění integrity v systému. Pravidla update-update budou tedy regulovat události, která mohou způsobovat nedodržení integrity.

3.6.1.2. Query-update

Query-update pravidlo obsahuje dotaz jako událost a vykonává update jako akci. Adekvátní může být situace, ve které je třeba uchovávat logy všech uživatelů, kteří se dotazovali databáze na informace o procesoru instalovaným na spektru Alphasys PC. Pokaždé, když systém zaregistruje tento typ dotazu, je nutné, aby zapsal do tabulky alphasys_query. Zde je názorný příklad:

```
create rule novy_query_update as
on select to pc.procesor where current.vyrobce = 'Alphasys'
do insert into Alphasys_query values
    (current.model, uzivatel);
```

Akce vkládá záznam do tabulky Alphasys_query. Každý záznam se skládá z konkrétního modelu PC a identifikace uživatele, který se dotazoval. Tabulka užívá informací k záznamu každého přístupu k informacím o procesoru.

Pravidla query-update jsou velmi výhodnou pomůckou pro vytváření přístupových logů a vysledovatelnosti záznamů auditových cest (audit trails)⁵.

Ne všechny výrobci SRBD nabízejí systémy spouštěčů (trigger systems)⁶, které podporují pravidla query-update. Tento nedostatek má za následek obtížné podchycování auditových cest.

3.6.1.3. Update-query

Dobrý ORSRBD by měl podporovat nějakou z forem systémů výstrahy. Ten vykonává mechanismy, které upozorňují uživatele na aktualizace databáze. Pravidlo update-query může být použito právě pro iniciaci/spuštění upozornění. V tomto pravidlu je událostí update a akcí je pak vygenerování odpovědi uživateli. Příkladem bude vytvoření tabulky bankucet pojmenovaného typu "ucet_t".

```
create type ucet_t (
    prijmeni    varchar(20),
    jmeno       varchar(10),
    credit_rating varchar(10),
    zustatek    int,
    posl_trans  date);

create table bankucet of type ucet_t;
```

Zde je žádoucí implementovat upozorňovač, který dá uživateli vědět, kdy se změnil stav na účtu. Požadované pravidlo je možné implementovat následujícími SQL řádky:

```
create alerter zustatek_alert
(mechanism = 'callback');

create rule zabezpeceni_update_aquery as
on update to bankucet.zustatek where current.prijmeni='Hnedy'
do alert zustatek_alert
```

První příkaz vytváří upozorňovač nazvaný "zustatek_alert". Upozornění tohoto alertu bude zasláno všem zainteresovaným klientům, protože je specifikován mechanismus callback. Druhé pravidlo definuje pravidlo update-query, které sleduje aktualizace na stavech účtů se jménem "Hnedy". Pokud je zaznamenána změna, akcí pravidla je upozornit alerter "zustatek_alert". Pokud uživatel nechce, aby alert zasílal automatické upozornění klientům, je možné užít volební mechanismus (poll mechanism)⁷.

Pokud chce uživatel dostat notifikaci alertů, na svém klientovi (klientském PC), musí zvolit server. Klient volí server užitím příkazu "poll;". Pak dostane pouze upozornění vykázaná od doby, co zvolil server. Obecně je v SQL řádce požadováno "listen[nazev_alerteru];". Může existovat několik

⁵ Audit Trail - Auditová stopa, znamená záznam průběhu, podle kterého je možné zpětně vysledovat položku (kterými procesy prošla, kde byla uložena atp.) většinou pro účely auditu.

⁶ Trigger je kolekce příkazů SQL (obecně procedura), která je spuštěna, nastanou-li nějaké události (více [lit1] str.108).

⁷ Poll mechanism je mechanismem pro výzvu na server a následné získání nějakého výsledku.

uživatelů naslouchajících stejnému alertu. Je také možné mít klienta naslouchajícího několika alertům. Pokud klienti naslouchají alertům, není jim zabráněno ve výkonu běžné práce.

Mechanismus callback je částečně použitelný pro upozorňovače, které jsou například navrhované k upozorňování aplikací určených k manipulaci s abnormálními událostmi. Aplikace navrhované pro monitorování časově-kritických procesů by byly dobrými kandidáty pro mechanismus callback, například systémy monitorování průmyslové továrny. Tyto aplikace stráví většinu svého času v nečinném stavu, naslouchající přiřazeným alertům. Aktivními se stávají pouze ve chvíli, kdy obdrží upozornění.

3.6.1.4. Query-query

Pravidlo query-query sleduje události, které jsou "upravovací" a odpovídají událostem vykonávajícím nějaké jiné úpravy. Pro příklad bude použita tabulka nazvaná tiskarna, pojmenovaného typu "tiskarna_t".

```
create type tiskarna_t (  
    vyrobce          varchar(20),  
    typ              varchar(10),  
    os               varchar(10),  
    dpi              int,  
    cb_rychlost      int,  
    barva_rychlost   int,  
    cena             float);  
  
create table tiskarna of type tiskarna_t;
```

V této tabulce budou drženy laserové tiskárny dvou různých výrobců – shotgun a fastprint, se shodnou cenou.

```
create rule query_query  
on select to tiskarna.cena where current.vyrobce='shotgun'  
and current.typ = 'laser'  
do instead  
    select cena  
    from tiskarna  
    where vyrobce = 'fastprint'  
    and typ = 'laser';
```

Toto query-query pravidlo bude vykonávat následující funkci. Pravidlo sleduje události, které opravují cenu laserových tiskáren shotgun. Když takováto událost nastane, systém vrátí cenu ekvivalentní k ceně fastprint. Použitím slova "instead" je způsobeno, že akce bude vykonána namísto události, která vystřelí pravidlo. Dotaz na cenu laserové tiskárny shotgun nebude nikdy aktuálně vybírat sady elementů dat stranou uložení informace.

```
select cena  
from tiskarna  
where vyrobce = 'shotgun'  
and typ = 'laser';
```

Cena shotgun může být považována za virtuální element dat. Cena laserové tiskárny fastprint je fyzicky uložena a vrácena, když je vyžádána cena shotgun nebo fastprint. Ukládání pouze jedné ceny a vrácení této jako výsledek všech dotazů garantuje, že cena obou tiskáren se jeví stejná pro všechny uživatele dotazujících se databáze. Stále je ale možné aktualizovat hodnotu uloženou v poli "cena" laserové tiskárny "Shotgun". Nicméně pravidlo query-query bude vždy mít přednost při dotazech na databázi a tato hodnota nebude vrácena.

Pravidla query-query mohou být používána pro zajištění větší bezpečnosti databáze, která může být dosažena s tradičními SQL systémy. Tradiční systémy fungují na principu udělování nebo odebrání přístupu k určitým informacím v závislosti na tom, jakou úroveň autorizace má uživatel. To může vést k úniku určitého množství dat. Vezmeme například databázi, která zahrnuje tabulku bankovních účtů.

```
create type ucet_t (  
    prijmeni      varchar(20),  
    firstame     varchar(10),  
    cis_uctu      int,  
    zustatek      float);  
  
create table bankucet of type ucet_t;
```

Řekněme, že účetní dostane přístup pouze pro čtení všech účtů na kterých je zůstatek menší nežli 500000.

```
create view ucetni_zam as  
select prijmeni, jmeno, cis_uctu, zustatek  
from bankucet  
where zustatek < 500000;
```

Toho dosáhneme vytvořením takového pohledu. Pokud účetní rozeběhne dotaz na nějakého známého držitele čísla účtu a získá nulový výsledek, znamená to pro účetního že výše zůstatku přesahuje částku 500000. Tímto způsobem se mohou důvěrné informace nedopatřením dostat k neautorizovaným uživatelům. Bezpečnost může být zvýšena vytvořením query-query pravidla, aby se vypořádalo s dotazy právě na takovéto účty.

```
create rule query-query as  
on select to bankaucet.zustatek  
where current.zustatek >= 500000  
do instead  
return nahodny_zustatek();
```

Z posledního řádku je patrné, že pravidlo vrátí náhodný zůstatek jako odpověď na dotaz pro účty se zůstatkem vyšším 500000. Pak je nemožné, aby neautorizovaný uživatel byl schopen získat informace, na které nemá autorizační právo. Správné informace podle tohoto pravidla dostane pouze autorizovaný uživatel.

Při používání pravidel se může objevit množství významových problémů. Věci, na které je třeba dávat pozor, zahrnují: neočekávané výsledky kvůli násobným pravidlům vykonávaným na stejné události,

nekonečné smyčky vznikající z řetězových pravidel, přerušené akce následované přerušením transakce, načasování aktivace pravidel a jaký bude jejich efekt na konečný stav databáze.

Je možné definovat množství pravidel. Je možné mít více pravidel, která se vykonávají na stejné události. Uvažujme dvě pravidla používaná k aktualizaci atributu ceny u laserové tiskárny výrobce fastprint.

```
create rule update_update_1 as
on update to tiskarna.cena where current.vyrobce = 'shotgun'
                                and typ = 'laser'
do update tiskarna
    set cena = new.cena
    where vyrobce = 'fastprint'
    and typ = 'laser';

create rule update_update_2 as
on update to tiskarna.cena where current.vyrobce = 'shotgun'
                                and typ = 'laser'
do update tiskarna
    set cena = new.cena/2
    where vyrobce = 'fastprint'
    and typ = 'laser';
```

Je zřejmé, že obě pravidla jsou vykonávána ve stejný čas, když je aktualizována cena laserové tiskárny. První pravidlo nastavuje cenu fastprint tiskárny, nastejno s cenou shotgun tiskárny, a druhé pravidlo nastavuje tuto cenu na poloviční. Výsledkem je, že stejné datové elementy jsou nastavovány na dvě rozdílné hodnoty ve stejný čas. Pořadí ve kterém jsou pravidla vykonána, určuje konečnou hodnotu, která se uloží k ceně fastprint tiskárny. Pořadí pravidel vykonávaných ve stejný čas je určováno systémem, nikoliv uživatelem. Pak bychom měli předejít vytváření více pravidel s konfliktními akcemi, které jsou startovány ve stejný čas.

Existuje i možnost vytvářet řetězová pravidla. To znamená, že jedna akce jednoho pravidla bude znamenat událost pro jiné pravidlo, a tak dále. Vezměme tato dvě pravidla jako příklad databáze tiskáren.

```
create rule retezove_1 as
on update to tiskarna.cena where current.vyrobce='shotgun'
                                and typ = 'laser'
do update tiskarna
    set cena = new.cena
    where vyrobce = 'fastprint'
    and typ = 'laser';

create rule update_update_2 as
on update to tiskarna.cena where current.vyrobce='fastprint'
                                and typ = 'laser'
do update tiskarna
    set cena = new.cena/2
    where vyrobce = 'techhouse'
    and typ = 'laser';
```

Když vznikne aktualizace ceny tiskárny shotgun, cena ekvivalentní fastprint tiskárny je aktualizována tak, že budou shodné. Aktualizace ceny tiskárny fastprint odstartuje druhé pravidlo v řetězu, které aktualizuje cenu laserové tiskárny výrobce techhouse. Nebezpečí vytváření řetězových pravidel je v nekonečných smyčkách.

Akční část pravidla se obvykle vykonává ve stejné transakci, která pravidlo startuje. Jestliže je transakce přerušena, akční část pravidla je přerušena také. Vezměme příklad pravidla update-update.

```
create rule update_update as
on update to tiskarna.cena where current.vyrobce = 'shotgun'
                                and typ = 'laser'
do update tiskarna
    set cena = new.cena
    where vyrobce = 'fastprint'
    and typ = 'laser';
```

Pokud se aktualizuje cena laserové tiskárny shotgun a pak se přeruší transakce, akce updatování transakce je přerušena také. To je běžně žádoucí postup události. Bylo by užitečné, kdyby systém pravidel zajišťoval funkcionalitu tak, že bude možné provést akční část pravidla po tom, co je aktivování transakce přerušeno. Například pokud je pravidlo query-update použito pro implementaci auditových stop, bylo by žádoucí vykonat akční část pravidla i když bude transakce přerušena. Výsledkem pak bude kompletní auditová stopa ukazující vstupy všech dotazů vytvořených na datech zahrnující i ty, které byly přerušeny. Pokud tedy chceme vykonat akční část i když je transakce přerušena, je nutné odstartovat akční část v odlišné transakci.

Ve většině SRBD je pravidlo vykonáno okamžitě. Nicméně aplikace se vyskytují kdekoliv, kde může být vyžadováno odstartování pravidla, které má být odloženo dokud není zapsána aktivace transakce. Řekněme, že jsme definovali pravidlo, které maže dodavatele tiskárny z tabulky dodavatelů v případě, že je smazána tiskárna z tabulky tiskáren.

```
on delete tiskarna
then delete dodavatel
where nazev_produkту = current.vyrobce;
```

Uvažujme typickou transakci v databázi. Nejdříve jsou všechny tiskárny vyrobené výrobcem fastprint mazány z tabulky tiskáren. Pak je jméno produktu pro rozsah tiskáren fastprint změněno na "print2000" v záznamech dodavatelů v tabulce dodavatelů. Pokud je pravidlo odstartováno v čase první aktualizace, jsou pak dodavatelé rozsahu fastprint smazáni. Výsledkem je, že dodavatelé tiskáren fastprint nebudou v tabulce dodavatelů v hlášené doložce (predicate clause) vykonání druhého procesu aktualizace. Požadováno je tedy odložení vykonání pravidla. Načasování vykonání pravidla může být kritické v určování konečného stavu databáze na konci transakce. Například, pokud musí být pravidlo query-query vykonáno okamžitě, ostatní mohou být spuštěny na konci transakce. Ideálně bychom měli být schopni specifikovat načasování, jaké chceme.

4. Současný trh s objektově relačními databázovými systémy

Poslední a praktická část této diplomové práce je věnována přehledu a zhodnocení databázových platforem, které se v současnosti na našem trhu vyskytují. Samozřejmě toto zhodnocení není úplné, platforem je více, ale zde by měli být uvedeny ty hlavní, nejvíce využívané. Ne všechny databázové systémy uvedené v podkapitolách jsou plně objektově relační. Některé z nich jsou relační systémy založené na SQL-92, nicméně protože objektově relační databázový model vychází především z relačního modelu a navíc jsou tyto systémy poměrně významné a často používané, proto není možné je v rámci této práce opomenout.

4.1. Přehled databázových platforem

Uvedené produkty je nutné chápat jako stručný přehled toho, co je na dnešním databázovém trhu k dispozici. Zahrnuty jsou jak systémy řízení báze dat dostupné zdarma, tak klasické komerční databázové platformy. V žádném případě se ale nejedná o kompletní přehled všech existujících produktů, zastoupeny jsou platformy, které jsou v oblasti databázových technologií významné s přihlédnutím k zastoupení v České republice. Obdobně neznamená neuvedení některé z vlastností, že ji daný produkt nepodporuje, toto zhodnocení je zaměřeno spíše na v současnosti typické oblasti databázových technologií. Mnohé databázové platformy jsou k dispozici v různých variantách - od jednoduživatelských verzí pro přenosné počítače až po výkonné superpočítače. Bližší informace o jednotlivých platformách a jejich variantách včetně podrobných technických popisů je možné získat například z příložených internetových zdrojů (viz kapitola Literatura).

4.1.1. Advantage Ingres Enterprise Relational Database

Aktuální verze: 2.6

Platforma Advantage Ingres Enterprise Relational Database je ve skutečnosti skupina produktů, kam patří i databázový server dříve známý pod obchodním názvem OpenIngres a později Ingres II. Celkový podíl na databázovém trhu se u této platformy pohybuje v řádu jednotek procent, nicméně i tak se jedná o kvalitního představitele moderních databázových technologií.

Nezávislé studie obvykle tuto platformu upřednostňují pro výhodný poměr cena/ výkon. V současné době velmi často zmiňovanou výhodou je snadná tvorba aplikací, které využívají Ingres ERD jako vestavěné databázové jádro mající dostatečně nízké systémové nároky. Otevřenost produktu Ingres ERD doplňuje možnost plnohodnotného přístupu k datům uloženým v databázích konkurenčních

platformou pomocí rozhraní Advantage Ingres Gateways a Advantage EDBC⁸. Zájemci o moderní technologie mají k dispozici nejen podporu XML, ale také podporu distribuovaného zpracování a replikačních mechanismů, tvorby vícevrstevných databázových aplikací, mobilního přístupu k datům a nechybí ani maximální využívání systémových zdrojů nabízených podporovanými provozními platformami - včetně podpory 64bitových architektur.

Velmi přehledná je administrace celého prostředí, ke které je možné využít nástroj s grafickým uživatelským rozhraním nazvaný Advantage Ingres Visual DBA. Tento nástroj umožňuje nejen základní správu databázových serverů a jednotlivých databází, ale například také administrátory stále žádanější optimalizaci výkonu celého prostředí. Uvedený nástroj lze použít i jako základní klientskou aplikaci pro zadávání příkazů jazyka SQL. Díky aplikačnímu rozhraní architektury Advantage Ingres Management Architecture je dokonce možné vytvářet vlastní administrátorské nástroje přizpůsobené konkrétním požadavkům.

Produkt Ingres ERD je dostupný pro celou řadu operačních systémů, včetně linuxových klonů a vyspělých komerčních unixů, nechybí ani podpora Windows. Více informací je dostupných v [COM03].

4.1.2. Oracle 10g

Aktuální verze: Release 1 (10.0.1.)

Databázové produkty společnosti Oracle patří mezi naprostou světovou špičku - to potvrzují i nezávislé průzkumy přisuzující této firmě na trhu s databázovými technologiemi celkový podíl mezi 40 až 50 procenty. Vlajkovou loď této společnosti představuje v oblasti databází v září uvedená platforma Oracle10g [ORA03].

Oracle10g obsahuje celou řadu zajímavých technologií, největší novinkou je ovšem podpora databázového zpracování ve gridu⁹ založeného na využití výpočetního výkonu libovolného množství geograficky oddělených počítačů. Zpracování ve gridu je již využíváno pro nedatabázové aplikace, známý je například projekt v rámci SETI. V oblasti databázových technologií je ovšem tento přístup revoluční a společnost Oracle se tak databázové konkurenci opět o pořádný kus vzdaluje.

Se zpracováním ve gridu souvisí také podpora tvorby vysoce škálovatelných a nepřetržitě dostupných aplikací pomocí technologie Real Application Cluster. Podstatné při nasazení této technologie je možnost škálování i na levnějších strojích tvářících se jako clustery a fakt, že v případě výpadku

⁸ Advantage EDBC je "middleware", který zajišťuje vykonávání aplikací na platformách Windows, Linux a UNIX s vysoce výkonným přístupem čtení/zápis.

⁹ Použití technologie gridu je definováno jako řízené užívání velkého počtu serverů a úložišť fungujících jako jeden počítač.

(plánovaného i náhodného) jednoho či více clusterů nedochází k ovlivnění funkčnosti provozované aplikace. V nejhorším případě může dojít k poklesu celkového výkonu - tento stav ale nemusí mnozí uživatelé ani zpozorovat.

Mezi další vlastnosti patří mnoho moderních databázových prvků - od podpory objektových vlastností a XML přes bezpečnost zajišťovanou pomocí virtuálních privátních databází a analytické funkce pro datové sklady a dolování dat až po podporu nestrukturovaných dat prostřednictvím Internet File System a vysoce škálovatelných a nepřetržitě dostupných aplikací. Samozřejmostí je bezproblémová podpora národních prostředí včetně správného třídění, dále pak podpora transakčního a distribuovaného zpracování či dotazovacího jazyka SQL.

Databázové platformy společnosti Oracle nabízí velmi mocné administrátorské prostředí s názvem Oracle Enterprise Manager, pomocí kterého lze provádět komplexní správu celého prostředí (tedy několika serverů i databází, jež navíc mohou být provozovány v geograficky odlišných lokacích) včetně plánování jednotlivých administrátorských činností, reakcí na problémové stavy apod [ABB02].

Důležitá je integrace Oracle10g s ostatními produkty portfolia společnosti Oracle - především jde o aplikační server a balík podnikových aplikací. V případě zájmu tak mohou uživatelé a vývojáři používat vyjma operačního systému vše od jedné společnosti. Databázová platforma Oracle10g je dostupná pro drtivou většinu dnes dostupných operačních systémů, Linux a Windows nevyjímaje.

4.1.3. MySQL

Aktuální verze: 4.1

Platforma MySQL je známá především díky úspěšnému tažení v oblasti internetových aplikací, zejména pak v kombinaci se skriptovacím jazykem PHP a webovým serverem Apache. Obliba je dána nejen kvalitami platformy a dostupností zdrojového kódu, ale také tím, že v některých případech není nutné za používání tohoto produktu platit žádné licenční poplatky. V případě zájmu je k dispozici také placená technická podpora.

V současné verzi podporuje MySQL dotazovací jazyk SQL na úrovni mnoha prvků z nejvlivnějších standardů (včetně standardu SQL 92 vstupní úrovně a nově také SQL 99) doplněných o některá vlastní rozšíření. Uživatelé této platformy tak mají k dispozici například agregační funkce, vnější i vnitřní spojování dotazů, dočasné tabulky, podporu dotazů nad tabulkami z různých databází, mohou ovlivnit prioritu některých SQL příkazů apod. Typická je pro MySQL také podpora celé řady datových typů. Jistou nevýhodou je chybějící podpora uložených procedur (chystá se až ve verzi 5.0) [MYS03].

Nechybí podpora replikačních technologií ani fulltextového vyhledávání. Maximální teoretický objem zpracovávaných dat představuje 8 miliónů terabajtů, primárně je tato platforma určena pro zpracování dat středně velkých rozsahů - především stovky megabajtů). Vývojáři databázových aplikací mají k dispozici celou řadu přístupových rozhraní, nechybí podpora jazyků C, C++, Java či Perl. Mezi podporovanými databázovými rozhraními lze pochopitelně nalézt i ODBC¹⁰ (MyODBC). K dispozici je také několik klientských nástrojů, které jsou dostupné jak pro nativní prostředí (obvykle na platformách Windows), tak i pro prostředí webových prohlížečů.

Pravděpodobně nejdiskutovanějším problémem je v případě MySQL podpora transakčního zpracování. Ve standardních konfiguracích není transakční zpracování implementováno. Na druhou stranu je nutné si uvědomit, že tato databázová platforma je určena především pro prohlížení dat. Obejití transakčního zpracování v konečném důsledku umožňuje platformě MySQL dosahovat vynikajícího výkonu.

Produkt MySQL je dostupný pro celou řadu operačních systémů, především pak unixů. Podporovány jsou ale také Windows nebo Mac OS X Server. Lze předpokládat, že tato databázová platforma bude i nadále dominovat především v mnoha internetových aplikacích, zejména pak všude tam, kde je primární záležitostí získávání dat.

4.1.4. Tamino XML Server

Aktuální verze: 4.1

Databázová platforma Tamino XML Server od společnosti Software AG využívá pro ukládání a správu dat formáty vyhovující standardům technologie XML. Základní myšlenka takového přístupu vychází především ze snahy odbourat převáděcí mezikrok mezi daty v jiném a XML formátu, který je v případě aplikací XML nad klasickými databázovými platformami typický [SOF03].

Tato platforma tak nabízí vysoký výkon při minimální nadbytečné administraci a vývoje. K datům uloženým v databázích Tamina je možné přistupovat například prostřednictvím dotazovacího jazyku X-Query, který vychází ze specifikace XPath. Vzhledem k orientaci technologie XML na internetové prostředí, je možné zadávat dotazy jako součást adresy URL přímo ve webovém prohlížeči (například, dotazy lze "spouštět" přímo také z aplikace vytvořené vlastními prostředky).

Rychlou a bezproblémovou tvorbu internetových aplikací založených na XML usnadňuje aplikační prostředí Tamino X-Application, pomocí kterého je možné přistupovat k datům uloženým v

¹⁰ Open Database Connectivity (ODBC) je otevřené standardní aplikačně programové prostředí (API) pro přistupování k databázím. Použitím ODBC příkazů v programu je možné přistupovat k souborům v různých databázích včetně MS Access, dBase, DB2, Excel a Text. Jako příslušenství k ODBC softwaru je nutné mít konkrétní ovladače pro konkrétní databáze. Hlavním dodavatelem je Microsoft.

databázích Tamino bez nutnosti přílišného programování. Vývojáři i nadále mohou využívat své oblíbené vývojové nástroje internetových aplikací, navíc nechybí ani aplikační rozhraní pro Javu či C++. Díky podpoře přístupu k datům uloženým v konkurenčních databázových platformách (pomocí ODBC, OLE DB¹¹ apod.) umožňuje Tamino XML Server přístup k datům i v těchto externích zdrojích - samozřejmě opět s možností jejich převedení do formátu XML. Vývojáři aplikací tak nemusí obtížně řešit problémy vyplývající z heterogenosti současně využívaných datových zdrojů.

Prakticky všechny možnosti nabízené touto databázovou platformou jsou dostupné prostřednictvím webového rozhraní, jinak tomu není ani v případě kompletní administrace samotného serveru a jednotlivých databází. Tamino tedy představuje komplexní platformu umožňující relativně bezproblémový vývoj a provoz aplikací využívajících výhod technologie XML.

Důležité také je, že Tamino XML Server nabízí drtivou většinu z toho, co běžné databázové platformy - od zpracování velkého objemu dat až po bezpečnostní prvky. Mezi podporované operační systémy, pro které je Tamino XML Server k dispozici, patří především přední unixové platformy (například HP-UX, IBM AIX) a MS Windows řady NT.

4.1.5. MS SQL Server 2000

Aktuální verze: 2000

Společnost Microsoft již dávno není dodavatelem pouze operačních systémů či kancelářských aplikací, ale velmi dobře se jí daří také na poli vývojových nástrojů a databázových technologií. Vlajkovou lodí je v poslední uvedené kategorii platforma MS SQL Server 2000 [MIC03].

Na začátku vývoje prvních verzí této databázové platformy byla úzká spolupráce se společností Sybase, MS SQL Server tedy původem vychází z produktů Sybase. Mezi hlavní novinky poslední verze patří například rozšíření podpory technologie XML, zpracování analytických principů používaných v rámci dolování dat či vylepšení nástroje SQL Query Analyzer určeného především k výkonovému ladění, nechybí ani podpora bezpečných připojení SSL.

Díky podpoře internetových technologií je navíc MS SQL Server 2000 považován za člena rodiny .NET. Vzhledem k tomu, že zejména s variantou Enterprise Edition se společnost Microsoft zaměřuje na podnikovou sféru, nijak nepřekvapí ani podpora škálovatelnosti (MS SQL Server dokáže při symetrickém multiprocessingu využít až 32 procesorů a 64 GB operační paměti), ani podpora tvorby nepřetržitě dostupných databázových aplikací. Součástí distribuce MS SQL Serveru pochopitelně není pouze samotný databázový server, ale také celá řada dalších základních administrátorských nástrojů a klientů.

¹¹ OLE DB je aplikačně programové rozhraní od Microsoftu pro přístup k odlišným datovým zdrojům OLE DB nezahrnuje pouze schopnosti SQL díky ODBC, ale zahrnuje také přístup k datům jiným nežli SQL.

Určitým omezením je orientace na operační systémy od stejné firmy, která vadí především možným zákazníkům využívajícím unixové provozní platformy. Na druhou stranu představuje MS SQL Server produkt, jenž je vhodný pro všechny zájemce, kteří chtějí mít maximum produktů od jednoho dodavatele, a snížit si tak riziko plynoucí ze vzájemné nekompatibility. Nechybí ani podpora platformy Windows Server 2003.

4.1.6. Interbase

Aktuální verze: 7.1

Platforma InterBase odpovídá standardu jazyka SQL92, nabízí ovšem také některá vlastní rozšíření. Samozřejmostí je podpora transakčního zpracování, deklarativní integrity včetně domén či národních znakových sad. Nechybí ani možnost zpracování read-only databází a technologie podpora XML. Aplikační logiku lze na straně databázového serveru zajistit prostřednictvím uložených procedur a triggerů. Je zde možnost snadného rozšiřování funkčnosti databázového stroje pomocí tzv. uživatelsky definovaných funkcí. Implementovány jsou také 64bitové vstupně/výstupní operace a dělení databáze do více souborů s teoretickým celkovým limitem 32 TB. Nechybí ani podpora SMP¹² v edici InterBase Server [BOR03].

Standardní součástí produktu je pouze velmi jednoduchá klientská aplikace, pomocí které uživatelé sice zvládnou mnohé ze základních administrativních kroků, ale například na inteligentní podporu jsou nuceni zapomenout. Naštěstí je k dispozici celá řada nástrojů dalších firem, které je možné pro správu InterBase použít, podobně jako je tomu v případě platformy Firebird.

Mnohé nedostatky ovšem plně vyváží široká podpora ze strany vývojových nástrojů značně oblíbené se InterBase těší zejména mezi vývojáři v Delphi. Je to dáno i tím, že k dispozici je celá řada komponent, které nabízí při tvorbě databázových aplikací nad InterBase skutečně maximum - od prostého zpracování dat až po administraci serveru. K podporovaným operačním systémům patří vedle Windows včetně Windows Server 2003 také Linux a Solaris.

Mezi odbornou veřejností lze stále ještě zaznamenat probíhající diskuse o budoucnosti této platformy, zejména pak v souvislosti s otevřeným projektem Firebird. Je sice pravdou, že na přelomu tisíciletí nebyla budoucnost InterBase příliš dobrá, proto také došlo jedno desetiletí po získání této platformy Borlandem k uvolnění zdrojového kódu InterBase, nicméně na období nejbližších let je podpora dalšího technologického rozvoje vedením společnosti přislíbena.

¹² SMP (symmetric multiprocessing) je vakonávání programů za použití několika procesorů sdílejících operační systém a paměť. V symetrickém multiprocesingu sdílejí procesory pakety a I/O sběrnici nebo datovou cestu.

4.1.6.1. Firebird

Aktuální verze: 1.5

V roce 2000 došlo ze strany společnosti Borland k uvolnění zdrojového kódu databázové platformy InterBase včetně zpřístupnění verze 6.0 bez nutnosti hradit licenční poplatky. Způsob licencování se sice po nějaké době opět navrátil ke komerčním metodám, nicméně tou dobou již existoval otevřený projekt s názvem Firebird. Skupina složená z předních osobností okolo InterBase využila uvolnění zdrojového kódu a zahájila vývoj nové databázové platformy.

V současné době je k dispozici Firebird verze 1.5, která je oproti verzi 1.0 a InterBase 6.0 založená na pročištěném kódu a jazyce C++. Nová verze tak odstraňuje celou řadu známých chyb a problémů původní InterBase. Mezi velké přednosti této relační databázové platformy patří vedle dostupnosti zdarma včetně zdrojového kódu také podpora standardů SQL, bezpečnostních mechanismů včetně rolí, snadná správa, podpora 64bitových architektur a vysoký výkon při relativně nízkých systémových nárocích. Samozřejmostí je podpora transakčního zpracování a velmi rozsáhlých databází (největší známá instalace zpracovává data o objemu cca 600 gigabajtů). Základem platformy je multigenerační architektura, která přináší například snadný a vysoce výkonný konkurenční přístup k datům. Značné mezery má sice Firebird v základních klientských nástrojích, nicméně k dispozici je celá řada produktů od třetích stran [FIR03].

Na druhou stranu se jedná o nedostatky, které v případě této platformy klady rozhodně nepřevyšují. Obdobně je možné rozšířit pomocí produktů od nezávislých dodavatelů i funkčnost databázového serveru - především pak díky uživatelsky definovaným funkcím. Velké oblibě se tato databázová platforma těší u vývojářů aplikací, především v prostředí Delphi. Je to dáno nejen snadnou dostupností a mnoha možnostmi Firebirdu, ale také existencí celé řady komponent zajišťujících nativní připojení k databázím Firebird.

Platforma Firebird je dostupná jak pro unixové operační systémy, tak i pro Windows. K dispozici je také speciální varianta provozuschopná bez instalace databázového serveru, a tím pádem snadno použitelná v jednouživatelských databázových aplikacích.

4.1.7. Informix dynamic server

Aktuální verze: 9.4

Informix Dynamic Server spolu s několika dalšími databázovými produkty převzala roku 2001 pod svá křídla společnost IBM. Některé hlasy hovořily o tom, že to bude znamenat konec vývojové linie Informixu, případně produktové řady DB2 [IBM03]. Ale nestalo se tak a společnost IBM zatím

dostává svému slibu podpory obou hlavních směrů. Obdobně jako vyspělé databázové platformy konkurence nabízí i Informix Dynamic Server podporu tvorby vysoce škálovatelných a výkonných a nepřetržitě dostupných aplikací, patří sem například asynchronní vstupně-výstupní operace a dynamická správa sdílené paměti. Samozřejmostí je také podpora replikačních technologií, objektově relačního modelu, jazyka SQL či celé řady přístupových rozhraní včetně JDBC¹³ a OLE DB.

Vedle klasických databázových aplikací je Informix Dynamic Server díky podpoře prostorových dat ideální platformou také pro provoz geografických aplikací. Maximální teoretická velikost databáze v nové verzi dosahuje hranice 128 PB. Jednotlivá rozšíření jsou v případě Informix Dynamic Server realizována pomocí modulů nazývaných DataBlade, například uvedenou podporu prostorových dat má na starosti modul IBM Informix Spatial DataBlade.

Samozřejmostí je také snadná administrace celého databázového prostředí, mnozí administrátoři ocení bezproblémové zálohování realizovatelné až na úrovni konkrétní tabulky. Administrace je dostupná nejen z klasických klientů včetně řádkových utilit, ale také pomocí klientů spustitelných v prohlížeči webových stránek. Celé prostředí je tak administrátor schopen spravovat z jednoho místa.

Databázová platforma Informix Dynamic Server je k dispozici pro celou řadu provozních platforem, obdobně jako v případě kvalitativně srovnatelné konkurence se jedná jak o unixové systémy včetně Linuxu, tak i o systémy společnosti Microsoft.

4.1.8. PostgreSQL

Aktuální verze: 7.3

PostgreSQL patří mezi přední představitele kategorie volně dostupných produktů formou volných zdrojových kódů a před osmi lety vyšel ze svého předchůdce, kterým byl projekt POSTGRES. Tato platforma je k dispozici zcela zdarma, bez ohledu na způsob použití - tedy jak pro komerční, tak i nekomerční projekty. Zájemci mají k dispozici také komerční technickou podporu od nezávislých společností.

Mezi charakteristické prvky této platformy patří podpora objektově relačního modelu či transakčního zpracování. Oproti jejímu předchůdci byl nepříliš vydařený dotazovací jazyk PostQuel nahrazen standardními konstrukty SQL. Samozřejmostí je podpora konkurenčního přístupu k datům, referenční integrity, triggerů, uživatelsky definovaných datových typů a celé řady objektových vlastností, včetně dědičnosti. PostgreSQL je vhodnou databázovou platformou také pro tvorbu internetových aplikací, například ve spojení se skriptovacím jazykem PHP.

¹³ Java Database Connectivity (JDBC) is je API specifikace pro spojování programů psaných v Java s daty v běžně používaných databázích. Toto rozhraní dokáže dekodovat žádosti o přístup přes SQL, které jsou poskytovány programům řídícím databázi

Vývojáři aplikací využívajících PostgreSQL mají k dispozici podporu aplikačního rozhraní pro celou řadu vývojových nástrojů - od C++ přes Perl až po Javu. Podporována jsou také přední databázová rozhraní, ODBC a JDBC nevyjímaje [POS03].

Serverová část PostgreSQL je dostupná především pro unixové operační systémy. O něco složitější mají situaci příznivci Windows, kteří mohou server portovat na operační systémy firmy Microsoft jen za určitých technologických podmínek. Příznivější stav je u klientských aplikací, které jsou pro Windows k dispozici bez problémů. PostgreSQL je všeobecně považován za jednu z nejkvalitnějších volně dostupných databázových platform. Díky podpoře ze strany mnoha vývojářů a celé řady společností včetně firem zabývajících se linuxovými distribucemi je tak možné PostgreSQL považovat za perspektivní produkt, se kterým se budeme v praxi setkávat stále častěji.

4.1.9. Caché

Aktuální verze: 5

Srdcem platformy Caché je multidimenzionální architektura umožňující různé postrelační pohledy na spravovaná data [INT03].

K datům lze díky této architektuře přistupovat nejen klasickým relačním způsobem, ale například také objektově - prakticky se všemi vlastnostmi známými z objektově orientovaných technologií. Fyzicky jsou data ukládána jako řídká vícerozměrná pole. Vedle objektového či relačního přístupu lze použít libovolnou další projekci zpřístupňující multidimenzionálně uložená data, například projekci do XML.

K dispozici jsou také nativní i univerzální databázová rozhraní a brány pro zajištění z hlediska vývojáře i uživatele transparentního přístupu k datům v konkurenčních platformách. Platforma Caché nabízí také skriptovací technologii CSP (Cache Server Pages), která je svými možnostmi srovnatelná například s ASP¹⁴ či JSP¹⁵. Velkou výhodou oproti mnohé konkurenci je možnost využívat bitmapové indexy v rámci aplikací s klasickým transakčním zpracováním - tedy nejen pro OLAP, ale také pro OLTP.

Součástí standardní distribuce je celá řada vyspělých nástrojů, základem je Caché Studio, které lze použít především pro vývoj aplikací. Databázovým administrátorům je určen nástroj Caché Configuration Manager, pomocí kterého lze ovlivnit celou řadu vlastností konkrétních instalací Caché. Databáze lze spravovat prostřednictvím nástroje Caché Control Panel. Pro práci s jazykem SQL je

¹⁴ Active Server Page (ASP) je HTML stránka zahrnující jeden nebo více skriptů, které jsou vykonávány Microsoft Web serverem předtím, než je stránka odeslána uživateli.

¹⁵ Java Server Page (JSP) je technologie pro kontrolu obsahu nebo zobrazení webových stránek za použití malých programů specifikovaných ve webové stránce a spouštěných na web serveru. Účelem těchto programů je modifikace stránky předtím, než je odeslána uživateli. Technologie JSP je srovnatelná s ASP, je vytvořená firmou Sun Microsystems.

určen nástroj Caché SQL Manager, k dispozici je také terminálový nástroj Caché Terminal, pomocí kterého lze řešit například velmi komplikované situace vyžadující přístup k databázím na co nejnížší úrovni. Platforma Caché je k dispozici pro celou řadu operačních systémů, především pak pro přední unixy a Windows.

4.1.10. Sybase Adaptive Server Enterprise

Aktuální verze: 12.0

Společnost Sybase nabízí v rámci svého produktového portfolia celou řadu databázových produktů - nosným článkem je robustní databázová platforma známá pod názvem Sybase Adaptive Server Enterprise. Sybase ASE (Adaptive Server Enterprise) podporuje vedle běžně dostupných prvků (jazyk SQL, transakční zpracování, snadná administrace) také některé ne příliš časté vlastnosti.

Mezi rozšiřující schopnosti lze zařadit například podporu zpracování transakcí v heterogenním databázovém prostředí či ladění příkazů v jazyce SQL, o které se stará SQL Překladač. Jak již bývá pro databázové platformy této kategorie typické, nabízí i Sybase ASE podporu automatického přizpůsobování databázového serveru aktuálním požadavkům, mimo jiné Sybase ASE využívá tzv. dynamickou optimalizaci výkonu. Podpora spočívá také v možnosti měnit výkonnostní nastavení databázového serveru (velikost vyrovnávací paměti, priority apod.) bez nutnosti zastavení serveru - tedy za plného provozu [SYB03].

Velký důraz klade tato databázová platforma na problematiku zabezpečení. V Sybase ASE nechybí například omezení přístupu na úrovni konkrétního řádku, podpora zabezpečeného přenosu pomocí protokolu SSL, digitálních certifikátů podle standardu X.509 v3 či spolupráce se serverem LDAP¹⁶. Pro zajištění zabezpečení na úrovni řádků využívá Sybase ASE technologii ACF (Application Context Facility) nabízející kontextově závislé klíče - při přístupu k datům jsou navraceny pouze ty záznamy, které z bezpečnostního hlediska odpovídají klíči vygenerovanému při přihlášení uživatele [SYB03].

Stranou zájmu Sybase nezůstává ani podpora moderních databázových aplikací. Přínos Sybase ASE spočívá především v možnosti využít technologie XML, komponent Enterprise JavaBean a Javy přímo na úrovni databázového stroje. Poslední z uvedených předností umožňuje vývojářům tvorbu uložených procedur přímo v Javě, případně v kombinaci s jazykem SQL. Mezi podporované operační systémy patří především kvalitní unixy (HP-UX, IBM AIX apod.), stranou ale nezůstávají ani operační systémy společnosti Microsoft (serverová část ovšem vyžaduje řadu NT). Samozřejmostí je přitom podpora 64bitových variant.

¹⁶ LDAP (Lightweight Directory Access Protocol) je softwarový protokol pro možnost lokalizovat organizace, jednotlivce a další zdroje jako soubory a součásti v síti, a to jak v Internetu, tak v intranetu. LDAP je zjednodušená verze DAP (Directory Access Protocol), která je částí X.500 (standardu pro adresářové služby v síti).

4.1.11. DB2 Universal Database Version 2

Aktuální verze: 8.1

Vlajkovou lodí v oblasti databázových technologií je pro IBM platforma DB2 Universal Database Version 8. Jedná se o produkt, který má za sebou dvě desetiletí vývoje. Ten se obdobně jako v případě mnohých programových produktů projevil především na kvalitě platformy a významně se zasadil také o přední pozici společnosti IBM na trhu s databázovými technologiemi.

Produkt DB2 představuje relační databázovou platformu s kvalitní podporou standardů SQL, transakčního zpracování a bezpečnostních mechanismů, která ve variantě Enterprise-Extended Edition plně vyhovuje požadavkům na vývoj a provoz vysoce výkonných, škálovatelných a nepřetržitě dostupných databázových aplikací. Vysoký výkon je přitom zajišťován především paralelním zpracováním (SMP, MPP¹⁷), nechybí ani možnost využití databázových clusterů, podpora dělení (partitioningu) či automatického procesu výkonostního ladění pomocí technologie SMART - Self Managing and Resource Tuning [IBM03].

Součástí této platformy jsou také rozšíření vhodná pro zpracování multidimensionálních či prostorových dat a umožňující nasazení databázové platformy v rámci analytických aplikací, například při využití technologie dolování dat. Obdobně jako v případě konkurenčních produktů i zde mají zákazníci na výběr z celé řady edicí plně vyhovujících jejich konkrétním požadavkům, novinkou z první poloviny letošního roku je DB2 Universal Database Express Edition určená především pro malé a střední firmy.

Aktuální verze DB2 umožňuje nejen zpracování klasických strukturovaných dat, ale také dat nestrukturovaných - od textu přes obrázky až po multimediální data. Vývojáři otevřených a rozlehlých databázových aplikací mají k dispozici pokročilé replikační technologie, navíc mohou při tvorbě webových aplikací využít podporu XML. Naprosto pochopitelná je také úzká spolupráce s moderními vývojovými nástroji, VisualAge for Java nevyjímaje. Popisovaná databázová platforma je známá vedle kvalitativních vlastností také podporou celé řady provozních systémů.

V případě operačních systémů jsou zastoupeny takřka všechny významné produkty - od špičkových unixových systémů včetně HP-UX či IBM AIX přes Windows včetně Windows Server 2003 či Windows CE. Změnou oproti předchozí verzi je ukončení podpory operačního systému OS/2. Nejbližší budoucnost databázových platform od IBM bude patrně ve znamení pokračujícího boje s největším konkurentem - Oracle10g.

¹⁷ MPP (massively parallel processing) je řízené vykonávání programů více procesory, které pracují na odlišných částech programu a každý procesor používá vlastní operační systém a paměť.

4.2. Shrnutí

Z předchozích kapitol věnovaných popisu funkčností a modulů výrobců databázových systémů je patrné, že databáze se od svých počátků vyvíjejí prakticky stejně rychle, jako technologie, které využívají. S vyššími možnostmi v oblasti výkonů procesorů a výkonů pamětí se objevují databázové stroje plně využívající těchto rychlostí a funkcí, aby byly schopné zvládnout větší množství dat potřebných k chodu dnešní moderní společnosti. Také záběr dnešních databázových systémů je v porovnání s minulostí o mnoho širší. Většina výrobců poskytuje v rámci databázových systémů nástroje na datové sklady, dolování dat, možnosti prezentace dat na internetu atp.

Přestože se v popisu jednotlivých produktů neobjevili veškeré nabízené funkce, může si každý alespoň trochu zasvěcený čtenář rámcově představit, k čemu je možné databáze využít a také jaké jsou jejich dnešní možnosti.

Bohužel, až na výjimky poskytované zdarma, není možné zjistit přesné ceny produktů zde popsanych. Také není možné postavit tyto produkty jednoduše do tabulky a přes standardní funkce je porovnat. Každý z těchto produktů je natolik specifický, že z vzniklého porovnání by nebylo možné označit ten či onen za lepší než jiný a podobně. Proto jsem se takového tabulkového srovnání vyvaroval a subjektivní hodnocení nechám na samotném čtenáři.

5. Závěr

Databázové technologie se již před mnoha desetiletími staly neodmyslitelnou součástí informačních technologií. Stejně jako každé další programové vybavení i databázové produkty prošly bouřlivým vývojem, který stále ještě není u konce.

Za databázové lze v dnešní době považovat drtivou většinu existujících aplikací – od jednoduchých evidencí přes účetní programy či internetové portály až po podnikové informační systémy. Se službami využívajícími databázi se dnes setkáváme na každém kroku, na poště při odesílání doporučeného dopisu, na letišti při odbavování k letu, kdekoli při zavolání z mobilu se do databáze operátora uloží záznam o uskutečněném hovoru atd.

Na téma databází a systémů pro řízení databází již bylo napsáno mnoho, ale informací, přístupů a metod věnujících se této oblasti je tolik, že není možné takto globální a otevřené téma několika publikacemi úplně vyčerpat. To byl také jeden z hlavních důvodů, proč jsem se rozhodl zpracovat na toto téma diplomovou práci.

Cílem této práce bylo zaměřit se na objektově relační modely databází současnosti jak z teoretického, tak praktického hlediska. První část byla věnována teorii, konkrétně srovnání přístupů, ze kterých objektově relační model vychází a následně vytvoření modelu, který by měl v současnosti vyhovovat pro podnikové účely v oblasti řízení dat. Druhá část pak byla praktická, věnovaná hlavním tvůrcům databázových systémů v současnosti, s důrazem na Českou Republiku.

Hlavní přínosy práce jsou v nastínění a popsání funkcí dnešních objektově orientovaných systémů řízení báze dat, které se mohou vyskytnout v objektově relačních modelech. V neposlední řadě je přínosem zmapování současného trhu s databázovými systémy z pohledu míry implementace objektově relačních prvků, jehož součástí bylo i zhodnocení a popsání provázanosti s ostatními moderními přístupy v současnosti, jako například datových skladů, dolování dat, internetových technologií JAVA, XML apod.

6. Použitá literatura

- [ABB02]: M. Abbey, M.Corey, I. Abramson; Oracle 9i, Softpress, Praha 2002
- [BOR03]: Borland Software Corporation, www.borland.com, prosinec 2003
- [COM03]: Computer associates, www.ca.com, prosinec 2003
- [CON03]: Connect!, časopis, <http://connect.cpress.cz>, listopad 2003
- [FIR03]: Firebird, www.firebirdsql.org, prosinec 2003
- [IBM03]: IBM, www.ibm.com , říjen 2003
- [INT03]: InterSystems, www.intersystems.com, listopad 2003
- [MER02]: V. Merunka: Objektový přístup v databázových systémech; ČZU v Praze, 2002
- [MIC03]: Microsoft, www.microsoft.cz, říjen 2003
- [MYS03]: MySQL AB, www.mysql.com, říjen 2003
- [NAH91]: E Nahouraii, F. Petry: Object Oriented databases; Computer Society Press, Los Alamitos, 1991
- [ORA03]: Oracle, www.oracle.com, říjen 2003
- [POK98]: J. Pokorný: Databázová abeceda; Science s.r.o., 1998
- [POK02]: J. Pokorný: Dotazovací jazyky; Karolinum, Praha 2002, první vydání
- [POK96]: J. Pokorný: Objektově Orientované SŘBD - motivy, principy, současný stav; Sborník semináře Moderní databáze, Ústí nad Labem, 1996
- [POS03]: PostgreSQL Global Development Group, www.postgresql.org, listopad 2003
- [REP99]: V. Řepa: Analýza a návrh informačních systémů, Ekopress, 1999
- [RIO00]: R. M. Riordan: Relační databázové aplikace; Computer press, Praha 2000, první vydání
- [SKI03]: Skillsoft corporate e-learning solutions, www.smartforce.com, 2003
- [SOF03]: Software AG, www.softwareag.com, říjen 2003
- [STO00]: M. Stonebraker, P. Brown: Objektově-relační SŘBD, analýza příští velké vlny; BEN – technická literatura, 2000
- [SYB03]: Sybase, www.sybase.com , říjen 2003
- [TOR99]: T. J. Torey: Database Modeling & Design; Morgan Kaufmann Publishers, San Francisco 1999, třetí vydání
- [VER97/1]: R. Vermeulen: Object-relational databases; Object magazine, August 1997, strana 31
- [VER97/2]: R. Vermeulen: Pushing the Limits; Object magazine, August 1997, strana 32