Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Student	: **Ondřej Novák**
Vedoucí bakalářské práce	: **RNDr. Helena Palovská, PhD.**
Recenzent bakalářské práce	: **Ing. Libor Gála**

TÉMA BAKALÁŘSKÉ PRÁCE

Relační databáze a objektové principy

ROK : 2007

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 20.08.2007 ....

podpis

Děkuji RNDr. Heleně Palovské, PhD. za cenné rady a odborné vedení bakalářské práce.

Abstrakt

Klasické relační databáze nejsou schopny plnit stále náročnější požadavky na ně kladené. Dochází k rozšíření relačních databází o objektové principy. Cílem práce je prozkoumání možností realizace objektových principů v relačních databázích a stav realizace ve vybraných komerčních databázových systémech. Mezi základní požadavky na objektově relační databáze patří podpora objektové identity a ukazatelů, rozšířených datových typů, včetně uživatelem definovaných typů, a tvorba uživatelem definovaných metod. Úvodní kapitola se zabývá teoretickými základy databázových systémů nutných k pochopení zbytku práce a orientaci na poli databázových systémů. Konec úvodní kapitoly se věnuje obecné charakteristice objektově relačních databázových systémů. V dalších kapitolách je podrobněji prozkoumán stav realizace objektových principů v komerčních databázích Oracle a IBM DB2 včetně jednoduchých ukázkových příkladů práce s objekty.

Abstract

Relational databases are unable to comply more and more challenging requirements. And as the level of requirements rises, relational databases are being extended by object principles. The goal of this thesis is to explore the possibilities of realization of object principles in relational databases and level of implementation in chosen commercial database systems. Basic requirements laid on object-relational databases are object identity and references, expansion of data types including user defined data types and user defined methods. Opening chapter inquires into theoretical basics of database systems essential to comprehend rest of the thesis and to orientate on the field of database systems. Ending of first chapter is aimed at general characteristics of object-relational database systems. Following chapters describe level of realization of object principles in commercial databases Oracle and IBM DB2 including sample examples of work with objects.

Obsah

Úvod

Naprostá většina aplikací musí nějakým způsobem uchovávat data. V dnešní době si lze jen těžko představit efektivní uložení dat bez použití databázových systémů. Přestože jednotlivých modelů dat pro databázové systémy existuje více, od 80. let roste popularita databází založených nad relačním modelem dat. Současně se ale ukazuje, že požadavky kladené na databáze rostou a klasické relační databázové systémy nejsou schopny tyto požadavky uspokojit. Vznik a masové rozšíření čistě objektových databází by znamenal zahození několika desítek let vývoje, proto dochází zejména k rozšíření relačních databází o objektové principy.

Cílem práce je prozkoumat možnosti realizace objektových principů v relačních databázích a stav realizace ve vybraných databázových systémech Oracle a DB2.

Úvodní kapitola je zaměřena na teoretické základy nutné pro pochopení a orientaci na poli databázových systémů. Jsou v ní vysvětleny základní principy databázových systémů, včetně relačního modelu dat, stručně popsaný jazyk SQL (základní znalost SQL je od čtenáře očekávána, proto je jazyk SQL spíše zmíněný než popsaný) a dále objektové principy tak, jak jsou obecně chápány, a také, jak jsou popsány ve standardu ODMG objektově orientovaných databází. Závěr první kapitoly tvoří popis relačních databází rozšířených o objektové principy a nároky na ně.

Další dvě kapitoly se zabývají prozkoumáním realizace objektových principů ve vybraných komerčních databázích. Druhá kapitola se zaměřuje na databázi Oracle, třetí kapitola na databázi DB2 od IBM. Syntaxe a někdy i logika práce s jednotlivými databázovými objekty se v obou systémech liší. Z tohoto důvodu, a také z důvodu větší názornosti, jsou obě kapitoly doplněny o konkrétní příklady, které nemají za cíl dopodrobna ukázat všechny možnosti využití (ostatně možností využití je neomezeně mnoho), ale jednoduše a pochopitelně ukázat logiku práce s daným objektem.

1. Teoretické základy

1.1. Principy databázových systémů

1.1.1. Databáze a databázový systém

V současné době je hojně využíván pojem databáze v různých významech a často je tímto pojmem myšlen databázový systém. Databázový systém (DBS) se skládá z databáze (DB) a systému řízení báze dat (SŘBD).

Databáze je uspořádaná množina dat. Skládá se z datových prvků, vztahů mezi nimi, integritních omezení a schématu. K zachycení elementárních hodnot slouží datové prvky. Těmito elementárními hodnotami mohou být například jméno, nadřízený a číslo_kanceláře. Vztahy mezi prvky dat jsou zachyceny pomocí složitějších datových struktur. Například skutečnost, že pan Alfa je nadřízeným slečny Bety by se dala zachytit pomocí datové struktury, která by obsahovala dvojici Alfa, Beta. Integritní omezení jsou podmínky, které musí data splňovat. Například číslo_kanceláře může být omezeno na číslo z intervalu <1,499>. Schéma je popis dat srozumitelný uživateli a odpovídajícímu software.

SŘBD je softwarové vybavení, které umožňuje definici, konstrukci a manipulaci s databází. Součástí SŘBD jsou jazyk pro definici dat (DDL – data definition language) a jazyk pro manipulaci s daty (DML – data manipulation language).

[1] [2]

1.1.2. Charakteristiky dat v databázích, transakční zpracování

Data v databázích by měla být perzistentní, spolehlivá, nezávislá a sdílená.

Perzistence znamená přetrvávání. Data existují nezávisle na programech a přetrvávají na paměťovém médiu nezávisle na tom, zda s nimi uživatel pracuje či nikoliv.

Na data uložená v databázi je kladen požadavek vysoké spolehlivosti. Data by měla být přesná, správná a platná. K narušení spolehlivosti může dojít jak náhodně, například při selhání paměťového média nebo při přenosu dat, tak záměrně nějakým útočníkem.

Nezávislost dat znamená, že programy přistupující k datům jsou nezávislé na tom, jak a kde jsou data uložená.

Data bývají dostupná více uživatelům, tedy jsou sdílená. S tím souvisí jednak snížení redundance (zbytečného opakování) dat, ale i potřeba paralelního přístupu, který bývá zajištěn zámky.

Se sdílením dat souvisí i transakční zpracování. Transakce je jistá posloupnost databázových operací, které se provedou z hlediska dalšího uživatele společně. Transakci je možné kdykoliv přerušit operací *ROLLBACK*, v takovém případě se všechny její efekty na databázi musí odstranit a databáze uvést do stavu před transakcí. Operace *COMMIT* naopak transakci potvrdí a provede. Snahou tedy je, aby došlo buď k převedení databáze z jednoho konzistentního stavu do druhého, nebo k zachování původního stavu.

Dále se sdílením dat souvisí potřeba rozlišovat několik základních typů uživatelů:

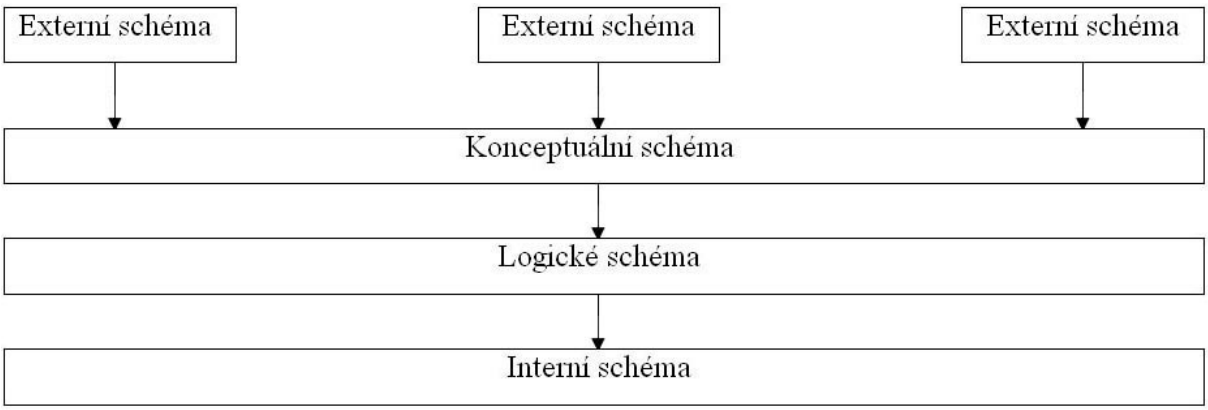
- správce databáze uděluje přístupová práva, vyhodnocuje využívání databáze, definuje a modifikuje schéma apod.,
- aplikační programátoři vytvářejí uživatelskou aplikaci,
- příležitostní uživatelé jsou schopni formulovat dotazy v některém z dotazovacích jazyků (např. SQL),
- naivní uživatelé neovládají dotazovací jazyk. Do databáze přistupují pomocí různých menu s předpřipravenými parametrizovatelnými dotazy.

[2]

1.1.3. Architektura DBS

Schéma čtyřvrstvé architektury databázových systémů ukazuje obr. 1.

- externí schémata jsou dílčími pohledy uživatelů na databázi,
- konceptuální schéma je pohledem na celou databázi, integrací externích schémat,
- logické schéma je již ovlivněno použitým databázovým modelem,
- fyzické schéma specifikuje vlastní uložení dat na paměťových médiích a metody přístupu k datům. Popisem databáze na této úrovni je fyzické schéma databáze.



Obr. 1 architektura DBS

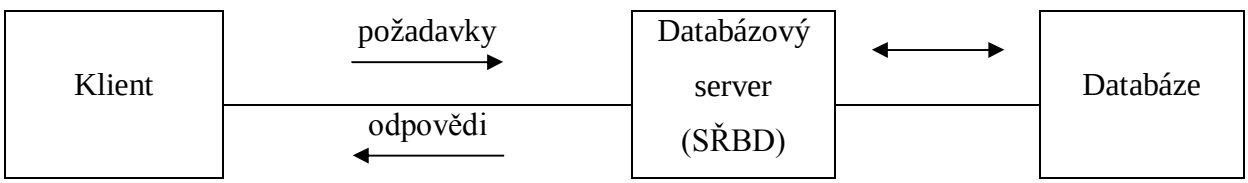
[3] [1]

1.1.4. Klient-server architektura

V dnešní době je klient-server architektura jednou z nejrozšířenějších. Na obr. 2 je znázorněné schéma dvouvrstvé klient-server architektury, na kterém je naznačena komunikace mezi klientem a serverem bez dalších aplikačních serverů. Klient posílá k serveru požadavky a server zasílá odpovědi. Mezi server a klienty se rozděluje šest typů služeb:

- prezentační služby (příjem vstupu, zobrazování výsledků)
- prezentační logika (hierarchie menu, obrazovek)
- logika aplikace (operace aplikačních programů),
- logika dat (podpora operací s daty, např. integritní omezení),
- datové služby (akce s databází, které nepatří do logiky dat, například transakční zpracování),
- služby zpracování souborů (získání dat z paměťových médií).

Podle rozdělení těchto služeb mezi klienta a server hovoříme buď o tlustém klientovi (aplikace a datové služby jsou soustředěny na klienta), nebo o tenkém klientovi (aplikace a datové služby jsou soustředěny na server). Pokud je mezi klienta a server přidán ještě aplikační server (případně servery), jedná se o třívrstvou (resp. n-vrstvou) architekturu klient-server.



Obr. 2 dvouvrstvá klient-server architektura

[1]

1.1.5. Datové modely

Pro potřeby této práce jsou významné hlavně relační datový model, objektově orientovaný datový model a objektově relační datový model. Uvedené modely jsou v této práci popsány podrobněji, ovšem existují i další modely, které jsou v této práci pouze zmíněny. Nejdříve se používalo tzv. hromadné zpracování dat, které pracuje na principu systému řízení souborů podporujících některé další techniky, jako např. sekvenční nebo indexované soubory. Poté přišly síťové a hierarchické datové modely, jenž jsou schopny velice rychle vracet dotazy, ale obtížně měnit a přidávat data. Poměrně novým přístupem je vznik nativních XML databází. Vzrůstající popularita XML vede k rostoucí potřebě ukládat XML dokumenty a data v nich obsažená do databáze, přičemž klasické relační databázové systémy k tomu nejsou vhodné.

[2] [4]

1.2. Relační model dat

S relačním modelem dat přišel v roce 1970 Dr. Edgar Frank Codd. Jeho rozšíření bránil nízký výkon tehdejších počítačů, protože relační model je výrazně pomalejší při provádění dotazů než tehdy rozšířené síťové a hierarchické databázové modely. Větší rozšíření tedy nastalo až koncem 80. let.

[4]

1.2.1. Pojem relace a její tabulkové zobrazení

Relační model dat je založený na relaci. Relace je libovolná podmnožina kartézského součinu[1] $D_1 \times \ldots \times D_n$, kde $n \geq 1$ a $D_1, \ldots, D_n$ jsou množiny hodnot jako např. množina křestních jmen. V databázové terminologii se těmto množinám říká domény (doména křestních jmen). Databázová relace má navíc dvě specifika. Prvky domén jsou atomické hodnoty, což znamená, že nemohou být rozděleny na menší části a tedy jsou v praxi představovány základními datovými typy jako např. *INTEGER* nebo *STRING*. Databázová relace je dále vybavena pomocnou strukturou, které se říká schéma relace. Toto schéma se skládá ze jména relace a jednotlivých atributů, přičemž každý atribut se skládá ze jména atributu a typu (jména) domény. Jménem atributu pak v praxi může být například ulice a typem domény datový typ *STRING* jako nepřesná reprezentace domény křestních jmen. Jednotlivým prvkům relace se říká n-tice.

[1] Kartézský součin množin X x Y je množinová operace, při které vznikne množina všech uspořádaných dvojic takových, že první položku tvoří prvek z množiny X a druhou položku prvek z množiny Y.

Za určitých okolností je možné relaci zobrazit jako tabulku. Záhlaví tabulky představuje schéma relace, jednotlivé řádky tabulky odpovídají n-ticím relace a sloupec tabulky trochu nepřesně představuje atribut (atribut zahrnuje celou doménu, ale sloupec tabulky jen prvky obsažené v dané relaci). Protože relace je vlastně množinou, a tedy každý prvek obsahuje maximálně jednou a nezáleží na pořadí jednotlivých prvků, tak v tabulkové reprezentaci může být každý řádek také nejvýše jednou a nezáleží na pořadí jednotlivých řádků ani sloupců. Všechny řádky musí být stejné struktury a položky jednotlivých sloupců tabulky musí být stejného typu (domény). Z atomicity prvků domén plyne i podmínka jedné hodnoty v každém poli tabulky. Tabulka i každý sloupec musí být jednoznačně pojmenován a z dále popsaných integritních omezení vyplývá i nutnost jednoznačné identifikace každého řádku, tedy nutnost existence klíče. V rámci zachování čitelnosti bude v této práci používáno spíše tabulkové názvosloví.

[1]

1.2.2. Integritní omezení a konzistence

Schématem relace je definována pouze struktura dat, ale pro praktické použití je potřeba také zajistit, aby se v relaci vyskytovala logicky správná data. Integritní omezení jsou většinou logické podmínky, které omezují hodnoty n-tic a vztahy mezi nimi.

Povinnou podmínkou je specifikace klíče. Klíč je minimální množina atributů, která je schopná jednoznačně určovat všechny n-tice relace (v nejhorším případě se jedná o množinu všech atributů). Klíčů může existovat více, a proto je vybírán tzv. primární klíč.

Dalším důležitým integritním omezením je referenční integrita. V relaci je atribut (resp. skupina atributů) vybrána za cizí klíč, což znamená, že musí odpovídat primárnímu klíči nějaké jiné relace. V praxi lze referenční integritou jednoduše docílit například toho, aby v relaci objednávka jednotlivé n-tice odpovídaly n-ticím v relaci zboží a nebyla v objednávce omylem obsažena položka, kterou firma neprodává.

Množina relací takových, že jejich prvky vyhovují všem definovaným integritním omezením se nazývá konzistentní a jedná se o tzv. relační databázi.

[1]

1.2.3. Relační algebra

Pro práci s tabulkami v relační databázi jsou k dispozici nástroje relační algebry. Vzhledem k tomu, že tabulky (resp. relace) jsou založené na množinách, jsou k dispozici

množinové operace sjednocení, průnik, rozdíl a kartézský součin. Dále jsou definovány další, databázové, operace mezi které patří projekce, restrikce (selekce) a spojení.

1.2.3.1. Množinové operace

Operace sjednocení, průnik a rozdíl mají smysl jen tehdy, když mají oba operandy množinové operace stejný počet atributů a odpovídající domény se rovnají. Až na tuto podmínku operace odpovídají svým předchůdcům z množinových operací. Jsou dány dvě tabulky A a B se stejně definovanými sloupci (doménami) a odlišnými řádkami. Sjednocením tabulek A $\cup$ B vznikne tabulka obsahující jak řádky z tabulky A, tak řádky z tabulky B. Průnikem tabulek A $\cap$ B je tabulka obsahující řádky nalézající se v obou tabulkách. Jako výsledek rozdílu tabulek A – B je tabulka, která obsahuje ty řádky tabulky A, které nejsou v tabulce B.

Na rozdíl od operací sjednocení, průnik a rozdíl má kartézský součin smysl s libovolnými tabulkami, není tedy potřeba, aby si domény odpovídaly a tabulky měly stejný počet sloupců. Výsledná tabulka má za schéma všechny atributy obou vstupujících tabulek a řádky tvoří všechny kombinace řádků obou tabulek.

1.2.3.2. Databázové relační operace

Mezi základní databázové operace patří projekce, restrikce a spojení.

Smyslem projekce je odstranění nežádoucích sloupců. Výsledná tabulka má stejný počet řádků jako tabulka původní, ale pouze ty sloupce, které je potřeba zobrazit.

Restrikce odstraňuje ty řádky, které nevyhovují dané podmínce, přičemž sloupce tabulky zůstanou nezměněny.

Spojení je kombinace kartézského součinu a restrikce. Výsledná tabulka má všechny sloupce vstupujících tabulek, ale pouze ty řádky, které vyhovují zadané podmínce. Nejčastější je spojení přes rovnost, kdy vzniklá tabulka obsahuje ty řádky, jejichž hodnota ve zvoleném sloupci odpovídá hodnotě ze sloupce druhé tabulky.

Jednoduché použití databázových operací v praxi je ukázáno dále v kapitole 1.3.2.

[1]

1.2.4. Coddova pravidla

Roku 1985 E.F. Codd publikoval dvanáct pravidel, které mají splňovat databázové systémy založené na relačním modelu dat. Jejich znění je následující:

- Informační pravidlo: všechna data musejí být reprezentována jako hodnoty v tabulkách.
- Pravidlo zajišťující přístup: každá hodnota musí být dosažitelná pomocí názvu tabulky, názvu sloupce a klíče.
- Zpracování neznámých hodnot: neznámé (*NULL*) hodnoty jsou podporovány pro vyjádření neznámé informace, a to nezávisle na datovém typu.
- Popis relačního katalogu: Popis celé databáze je na logické úrovni reprezentován stejně jako běžná data, tedy také jako tabulka.
- Pravidlo pro jazyk: musí existovat alespoň jeden jazyk, který podporuje DDL, DML, integritní omezení, práci s transakcemi a autorizaci.
- Pravidlo pohledů[2]: všechny pohledy, které jsou teoreticky aktualizovatelné jsou také systémem aktualizovatelné.
- Pravidlo operací: schopnost zpracování relace jako operandu je zachována nejen při čtení dat, ale i u vkládání, aktualizace a odstranění dat.
- Pravidlo fyzické nezávislosti dat: výsledky operací jsou nezávislé na fyzické datové struktuře.
- Pravidlo logické nezávislosti dat: výsledky operací jsou nezávislé na změně logické struktury.
- Pravidlo nezávislosti dat na integritních omezeních: integritní omezení musí být definovatelná prostředky relační databáze a musí být uchovatelná v katalogu a ne v aplikačním programu.
- Pravidlo nezávislosti dat na distribuci: výsledky operací jsou nezávislé na konkrétním rozmístění dat v distribuované databázi.
- Pravidlo nenarušitelnosti SŘBD: žádný uživatel (ani aplikace) nesmí obcházet ani narušovat rozhraní SŘBD.

[5] [4]

[2] Pohled je virtuální tabulka implementující uživatelské schéma. Pohled může například omezit sloupce tabulky, které uživatel může vidět apod.

1.3. SQL

Počátky jazyka SQL sahají do roku 1974, kdy vznikl jeho předchůdce, jazyk Sequel[3]. Jedná se o velmi rozšířený neprocedurální dotazovací jazyk. Důležité je, že jazyk SQL je součástí relačního SŘBD. Součástí SQL nejsou jen jazyky pro definici dat a manipulaci s daty, ale i další příkazy týkající se přístupových práv, transakcí, kódování znaků apod.

Jazyk SQL je standardizován. Jako první byl přijat standard v roce 1986, který bývá označován jako SQL86. Potřeba integritních omezení vyústila ve standard SQL89. Další vývoj vedl ke vzniku velmi rozšířeného SQL92. Objektové prvky a další vylepšení se projevily v SQL:1999. Od té doby vznikly ještě dva standardy, SQL:2003 a SQL:2006, týkající se hlavně podpory XML a technologií s XML souvisejících.

[1] [4] [6]

1.3.1. Definice dat

Těmito příkazy jsou vytvářeny, měněny a odstraňovány struktury databáze, jako jsou například tabulky, pohledy, ale i indexy nebo schémata. Příkaz *CREATE* vytvoří nový objekt, zatímco příkaz *ALTER* slouží pro jeho úpravu a příkaz *DROP* pro zrušení.

SQL podporuje klasické datové typy tak jak jsou známé z programovacích jazyků. Jedná se zejména o numerické typy (celočíselné i s desetinnou čárkou), znakové řetězce a časové datové typy.

1.3.2. Manipulace s daty

Příkazy pro manipulaci s daty jsou *SELECT*, *INSERT*, *UPDATE* a *DELETE*. Zatímco příkazy *INSERT*, *UPDATE* a *DELETE* slouží pro aktualizaci údajů, k zadávání dotazů je k dispozici pouze příkaz *SELECT*. Pro doplnění a lepší pochopení databázových operací relační algebry je v následujícím odstavci ukázán příklad projekce, restrikce i spojení.

Jsou dány tabulky Oddělení a Zaměstnanci, mají definované atributy a jsou naplněny daty tak, jak ukazuje obr. 3. Je potřeba vypsat jména všech mužských zaměstnanců a názvy oddělení ve kterých pracují. Pro tento dotaz je možné použít například příkaz *SELECT* jméno, název *FROM* Zaměstnanci *JOIN* Oddělení *ON* číslo_oddělení = číslo *WHERE* pohlaví = ‘M’;. Část jméno, název říká, že má být provedena projekce na výsledné sloupce tak, aby se zobrazily jen sloupce jméno a název. *JOIN* Oddělení *ON* číslo_oddělení = číslo definuje

[3] Právě návaznost na jazyk Sequel způsobuje, že někteří odborníci zkratku SQL nevyslovují jako es-kjů-el (resp. es-kvé-el), ale síkvl.

použité spojení a *WHERE* pohlaví = ‘M’ provede restrikci na ty řádky, u kterých pohlaví odpovídá M. Výsledek dotazu je vidět na obr. 4.

Zaměstnanci	jméno	číslo_oddělení	Pohlaví
	Petr	1	M
	Pavel	1	M
	Ondra	2	M
	Tereza	3	Ž

Oddělení	číslo	název
	1	Prodej
	2	IT
	3	Marketing

Obr. 3 tabulky Zaměstnanci a Oddělení

Výsledek	Jméno	název
	Petr	Prodej
	Pavel	Prodej
	Ondra	IT

Obr. 4 výsledek dotazu

1.4. Objektové principy

Přes ohromnou popularitu výše popsaných relačních databází se objevily úlohy, na které nestačily. Jedná se zejména o CAD/CAM[4], telekomunikace, prostorové objekty, multimédia a další, kdy jsou relační databáze příliš těžkopádné na práci s potřebnými datovými strukturami a vznikla potřeba složitějších operací, které je potřeba s těmito datovými strukturami provádět.

Data v objektově orientované databázi nejsou jen pouhými hodnotami seskupenými do datových struktur, ale jsou vztažena k abstraktním objektům, které mohou přímo odpovídat entitám[5] reálného světa.

Na rozdíl od relačních databází je možné provádět s objekty složitější operace a přenést podstatnou část aplikační logiky efektivně přímo na databázi, přičemž se výrazně zmenší nároky na aplikační software.

[1]

1.4.1. Charakteristiky objektově orientovaných SŘBD

Charakteristiky OOSŘBD[6] vychází z povinných charakteristik OOSŘBD uvedených v Manifestu skupiny ALTAIR z roku 1989[7]. Jedná se o objekty a jejich identitu, typy a třídy,

[4] Systémy počítačové podpory pro design a výrobu.

[5] Entita je cokoliv , co je dostatečně důležité na to, aby to bylo opatřeno jménem.

[6] OO značí objektově orientovaný.

zapouzdření, polymorfizmus a dědění. Další povinné charakteristiky (např. perzistence dat) jsou splněny tím, že se jedná o SŘBD.

Bohužel na poli objektových principů panuje zmatek v přesných významech jednotlivých termínů. Navíc objektové principy samozřejmě neplatí jen v OO databázích, ale i v objektově orientovaném programování, což vede ke vzniku dalších nepřesností. Objektem je vlastně cokoliv, co má svoji identitu. Instance je vytvořený konkrétní objekt některého z typů (tříd) objektů. Operací se rozumí souhrnně jakákoliv funkce nebo procedura (pro programátory metoda). Metodou nazývám operaci u konkrétního objektu (instance).

[1]

1.4.1.1. Objekty a jejich identita

Objekt je popsán čtyřmi základními vlastnostmi, kterými jsou identifikátor, jméno, perzistence a struktura. Identifikátor je unikátní označení objektu často označované jako OID[8]. Na rozdíl od klíče v relačních databázích OID nemůže být v průběhu života objektu měněno. Objekt může mít navíc své jméno, které je určeno k odkazování se na tento objekt v programu. Objekt, jenž není perzistentní, existuje jen po dobu běhu programu. Struktura určuje způsob vytvoření objektu a zdali je atomický nebo složitý. Atomické objekty jsou například *BOOLEAN*, *STRING* nebo *DOUBLE*, příkladem složitého objektu je kolekce.

Dále je potřeba rozlišovat mezi objektem a hodnotou objektu. Hodnota objektu může být struktura skládající se z dalších objektů nebo z literálů, které nejsou objekty a představují základní datové typy (*INTEGER*, *STRING* apod.). Identita objektu je nezávislá na jeho hodnotě a je implementována pomocí OID. Dva objekty (instance) tedy mohou být identické (jedná se o stejné objekty), nebo si mohou být rovny (mají stejnou hodnotu).

[1]

1.4.1.2. Typy a třídy

Typ v OOSŘBD vyjadřuje společnou strukturu skupiny objektů se stejnými charakteristikami. Kromě struktury jsou zadány operace, které lze s objekty daného typu provádět.

---

[7] Tento manifest je možné si přečíst např. na webové stránce http://www.cl.cam.ac.uk/teaching/2003/Databases/oo-manifesto.pdf [ov. 30.7.2007].

[8] Object Identifier.

Třída může být v různých OOSŘBD pojímána odlišně. Zahrnuje v sobě možnost vytvářet nové instance objektů, dále pak možnost sdružovat všechny potřebné instance objektu dané třídy pohromadě v tzv. kontejneru objektů, ke kterému může uživatel přistupovat pomocí daných operací.

[1]

1.4.1.3. Dědění

OOSŘBD dovolují uživateli odvozovat z existujících tříd nové třídy. Nová třída, tzv. podtřída (resp. potomek), pak zdědí po své nadtřídě (resp. rodiči), případně více nadtřídách, všechny atributy a operace. Třída může mít libovolné množství podtříd, ale někdy existuje omezení, že třída může mít jen jednu nadtřídu.

Při konfliktu ve jméně operace nebo atributu je použita operace nebo atribut dané třídy a tedy nedochází k dědění z nadtřídy.

[1]

1.4.1.4. Zapouzdření

Jedná se o princip přistupovat pouze přes dané rozhraní, tedy hodnoty atributů objektů nejsou obecně přístupné z programovacího jazyka. Rozhraní je dáno jménem a argumenty.

Zapouzdření dovoluje měnit programátorovi datovou část, aniž by se změnily programy pracující s odpovídajícími objekty, protože jméno z vnějšku přístupné operace a případné argumenty zůstanou stejné.

Operace mohou být znovu definovány pro různé typy objektů. Jméno operace tak může znamenat různé věci při aplikaci na různé objekty. Dále může být zděděná operace znovu nadefinována. Tomuto jevu se říká přetížení a vede k tomu, že připojení těla operace k objektu (resp. jménu operace) nastává až v době provádění programu.

[1]

1.4.1.5. Polymorfizmus

Polymorfizmus je schopnost operací fungovat na objektech více než jednoho typu (třídy). Polymorfizmus může být univerzální, kdy jde o operace s potencionálně nekonečným oborem typů a nebo ad hoc, u kterého operace pracují nad konečnou množinou typů.

[1]

1.4.2. Objektový model ODMG

Standardizací na poli OOSŘBD se zabývala skupina ODMG[9]. Standard ODMG 3.0 z roku 2000 obsahuje objektový model, definiční jazyk ODL, dotazovací jazyk OQL, formát pro výměnu objektů a vazby na programovací jazyky C++, Smalltalk a Java. Vybrané části standardu jsou stručně popsané dále.

[1]

1.4.2.1. Objekty a literály

Objekty a literály se podle standardu ODMG dělí do tří typů: atomické, kolekce a strukturované.

Literály představují hodnotu, která nemá OID. Atomickými literály jsou základní předdefinované datové typy jako *LONG*, *SHORT*, *FLOAT*, *DOUBLE*, *BOOLEAN*, *STRING* aj. Strukturované literály jsou hodnoty, které jsou vytvořeny pomocí speciálního n-ticového konstruktoru. Patří sem předdefinované struktury *DATE*, *TIME*, *TIMESTAMP*, *INTERVAL*, případně uživatelem nově vytvořené struktury.

Objekty mohou být buďto rozhraní, nebo třídy. Rozhraní je popis abstraktních objektů, které nemohou být instancovány (použity k tvorbě objektů) a slouží k popisu operací, které mohou být děděny. Pro instancování objektu je potřeba použít třídu. Díky tomu, že všechny objekty jsou odvozeny od rozhraní Object (ať již přímo či zprostředkovaně přes další objekty), obsahuje každý objekt několik základních operací, jakými jsou copy (vytvoří kopii objektu), delete (smaže objekt) a same_as (porovná objekt s jiným).

Kolekce představuje množinu objektů nebo hodnot stejného typu. Kolekcí může být objekt i literál a základními typy jsou *SET*, *BAG*, *LIST*, *ARRAY* a *DICTIONARY*. *SET* představuje množinu objektů nebo hodnot stejného typu. *BAG* představuje multimnožinu, tedy množinu, ve které mohou existovat opakující se prvky. *LIST* tvoří očíslovaný seznam prvků, prvky v této kolekci tedy mají svoji přesnou pozici v seznamu. *ARRAY* je podobný jako *LIST*, ale má pevnou velikost, nicméně existuje operace umožňující změnu této velikosti. *DICTIONARY* tvoří množinu párů <k, v>, kde k je unikátní klíč a v hodnota.

[1]

[9] Původně ODMG znamenalo Object Database Management Group, ale skupina se roku 1998 přejmenovala na Object Data Management Group.

1.4.2.2. Rozhraní pro kolekce

Každý objekt typu kolekce je podtřídou typu Collection, a tedy dědí některé základní operace společné pro všechny tyto objekty. Jedná se zejména o operace copy, delete a same_as zděděné z rozhraní Object, dále o operace vložení elementu do kolekce, odstranění elementu z kolekce, zjištění počtu prvků v kolekci, operace pro práci s iterátorem[10] apod.

[1]

1.4.2.3. Uživatelem definované objekty

V OOSŘBD je možné vytvořit uživatelem definované objekty. Definice těchto objektů obsahuje atributy, vztahy k ostatním objektům a operace. Jedná se o definici vlastní třídy.

Atributy slouží k popisu vlastností objektu a typický nabývají hodnot vybraného druhu literálů. Vztahy specifikují, které dva typy objektů jsou spolu propojeny a jak. Dále je možné definovat operace, které musí mít v rámci objektu unikátní jméno a mohou specifikovat hodnotu atributu, případně hodnotu vracet.

[1]

1.4.2.4. Rozhraní, třídy a dědičnost

V ODMG existují dva typy dědičnosti: rozhraní a třídy. Dědí se jak atributy, tak operace. Rozhraní nemohou být instancována, zatímco třídy ano. Rozhraní může dědit z více rozhraní, ale třída maximálně z jedné třídy.

[1]

1.4.2.5. Extenze a klíče

Uživatel může definovat extenzi pro každou třídu. Extenze obsahuje stávající perzistentní objekty daného typu třídy a má své jméno stejně jako např. objekty, atributy nebo vztahy. Každá třída s definovanou extenzí může obsahovat jeden nebo více klíčů, které jsou unikátní a mohou se skládat z atributů.

[1]

1.5. Objektově relační databáze

Jak je naznačeno v úvodu kapitoly 1.4, s postupem času se ukázalo, že s relačními databázemi jsou spojeny některé problémy. V relačních databázích není možné vytvořit

[10] Iterátor je objekt umožňující projít všechny prvky kolekce.

hnízděné struktury[11], existuje jen omezené množství datových typů, vznikají dlouhé sekvence SQL v dotazech obsahujících konceptuálně složitá data a datové struktury jsou odděleny od chování, což je v rozporu s objektovými principy uplatňovanými v objektově orientovaném programování. Proto je zřejmé, že je potřeba čistě relační databáze přizpůsobit, aby lépe vyhovovaly novým požadavkům kladeným na databázové systémy.

V praxi by bylo velmi nevhodné zahodit rozsáhlé investice do relačních technologií. Navíc objektové databáze měly a mají své problémy, např. neexistence teoretického základu, rozsáhlejší implementace standardizovaného jazyka aj. Většinou tedy dochází k rozšíření stávajících relačních databází o objektové prvky. Objektově relační databáze stále používají relační model dat jako svůj základ, ale je možné uživatelem definovat vlastní datové typy (včetně hnízděných), funkce a využívat dědění, polymorfizmu a objektové identity (odkazů). Rozšíření relačních databází o objektové prvky umožňuje do větší míry přesunout logiku aplikace na databázi, což vede k vyššímu výkonu zejména u aplikací vyžadujících pro své výpočty veliké objemy dat.

Standardizaci některých objektových prvků v sobě zahrnuje SQL99, nicméně tento standard (a ani pozdější standardy z roku 2003 a 2006) není příliš rozšířen. Jednotliví výrobci databázových systémů často implementují vlastní objektová rozšíření. V dalších částech této práce je prozkoumána podpora objektových principů databázovými systémy Oracle Database 10g a IBM DB2 9.

[7] [8]

1.5.1. Rozšířené datové typy

Nové datové typy zahrnují velké objekty, abstraktní datové typy (ADT), typ kolekce a tzv. odlišující typy.

Do relační databáze je možné umístit data reprezentovaná jako velké objekty, a to jak ve znacích (*CLOB*), tak i v binární verzi (*BLOB*). Standardní možnosti jazyka SQL jsou pro práci s těmito objekty většinou silně omezené, proto jsou dále zpracovány aplikačním software nebo uživatelem definovanými funkcemi. U databázových systémů jednotlivých firem bývají k dispozici moduly s uživatelem definovanými datovými typy a funkcemi, které je možné využít pro zpracování takových objektů z různých oblastí jako jsou např. prostorové objekty, obrázky, videa apod.

[11] Struktura schopná uchovat více atributů v jednom poli tabulky.

Abstraktní datové typy představují možný způsob reprezentace složitějších datových struktur v jednom poli tabulky. S ADT se pracuje stejně jako s typy vestavěnými do SQL. U ADT je možné definovat operace, a to včetně operací uživatelem definovaných. Abstraktní datové typy podporují dědičnost a jsou zapouzdřené. Řádky tabulky vzniklé na základě ADT mají objektovou identitu a typ odkazu na ně může být založený na nějakém jednoznačném atributu, uživatelsky definovaný nebo generovaný automaticky.

Kolekce v objektově relačních databázích odpovídají kolekcím z OO databází, ale jejich možnosti a druhy jsou omezenější.

Odlišující typy umožňují rozlišit mezi hodnotami, které mají stejné datové typy, ale přesto jsou neporovnatelné. Např. číslo domu i hodnota IQ pravděpodobně budou typu *INTEGER*, nicméně od jejich porovnání se mnoho informací očekávat nedá.

[7]

2. Realizace objektových principů v databázi Oracle

Databáze Oracle ve svých raných verzích byla první komerční relační databází. Firma se původně jmenovala Software Development Laboratories a založili ji v roce 1977 Larry Ellison, Bob Miner a Ed Oates. O rok později vznikl Oracle verze 1, který ale nebyl oficiálně vydaný. Roku 1979 vznikl Oracle verze 2, první komerční SQL relační databáze a v témže roce došlo k přejmenování firmy na Relational Software Inc. Jméno Oracle Systems je z roku 1983. Momentálně je Oracle největším světovým dodavatelem podnikových informačních systémů.

V této práci je prozkoumána databáze Oracle ve verzi 10g. Verze 11 je sice již ohlášena, ale není k dispozici komplexní dokumentace a ani by nemělo dojít k výrazným změnám objektově relačního modelu

Hlavním zdrojem v této kapitole je Oracle Database Documentation Library[12], zejména Application Developer's Guide – Object Relational Features, ale i jiné části zabývající se podrobněji vybranými charakteristikami. Pro ověření a pokusy je použita nejnižší edice databázového serveru Oracle, Oracle Database 10g Express Edition, která je k dispozici zdarma.

Cílem kapitoly je prozkoumání realizace objektových principů v databázi Oracle. V ukázkových příkladech je ukázáno vytvoření jednotlivých typů, objektů, metod aj. pomocí příkazu *CREATE*. Obecně ke změně vytvořených objektů slouží příkaz *ALTER* a pro jejich odstranění příkaz *DROP*, ale syntaxe těchto příkazů není v této práci podrobněji rozebírána, neboť nepřináší nic nového k uplatnění objektových principů. Jejich přesné použití je možné vyhledat v dokumentaci.

Z objektových prvků databáze Oracle podporuje objekty, typy objektů (uživatelem definované datové typy), metody (včetně uživatelem definovaných metod), ukazatele, dědičnost, polymorfizmus i zapouzdření.

[9] [10] [11]

[12] Kompletní dokumentace je k dispozici na webové stránce http://www.oracle.com/pls/db102/homepage [ov. 12.8.2007].

2.1. Objektové typy

Díky objektovým typům (object types) je možné v databázi Oracle definovat vlastní typy objektů a pracovat s nimi podobně jako se základními datovými typy. Typy představují jakousi předlohu, definici charakteristik a chování. Jsou částečně zapouzdřené, skládají se z veřejného rozhraní (public interface) a privátního těla (private body). Rozhraní je tvořeno deklaracemi jednotlivých atributů a metod, tělo obsahuje vlastní zdrojový kód deklarovaných metod.

Na př. 1 je vidět tvorba nového typu. Nový datový typ má reprezentovat osobu a uchovávat o každé osobě identifikační číslo, jméno, příjmení a kontaktní email. Aby bylo jasně patrné, že se jedná o objektový typ, je vhodné názvy typů odlišovat například přidáním _typ. Rozhraní nového typu osoba_typ je vytvořeno příkazem *CREATE TYPE*. Obsahuje atributy idno (základního datového typu *NUMBER*), jméno, příjmení a email (datového typu *VARCHAR2*, v závorce je udána délka textového řetězce). K vytvoření těla slouží příkaz *CREATE TYPE BODY*. Metoda get_idno vrací hodnotu atributu idno.

Př. 1 vytvoření nového objektového typu

```
CREATE TYPE osoba_typ AS OBJECT (
    idno            number,
    jméno           varchar2(20),
    příjmení        varchar2(20),
    email           varchar2(25),
    MEMBER FUNCTION get_idno RETURN number)
    NOT FINAL;
CREATE TYPE BODY osoba_typ AS
    MEMBER FUNCTION get_idno RETURN number IS
    BEGIN
            RETURN idno;
    END;
END;
```

Místo *CREATE TYPE* je možné použít *CREATE OR REPLACE TYPE*, což vede k tomu, že pokud existuje původní objektový typ se shodným jménem, bude přepsán. Odpadá tedy nutnost odstraňovat původní datový typ pokud existuje.

Odlišující datové typy nejsou podporovány a musí být emulovány objektovým typem o jednom sloupci, což vede ke zbytečnému znepřehlednění kódu v případech, kdy stačí jednoduchý literál a dochází ke zbytečné nutnosti pracovat s objektem.

2.1.1. Dědičnost

Podpora dědičnosti v databázi Oracle umožňuje definovat podtypy (subtypes), které po svých rodičích, tzv. nadtypech (supertypes), dědí atributy i metody. Vzniká tak možnost z obecných nadtypů postupně vytvářet specializovanější podtypy.

Typ osoba_typ umožňuje uchovávat data o různých osobách, ale například může být prospěšné o studentech uchovávat i název školy, ve které studují. Není potřeba vytvářet kompletně nový datový typ, ale je vhodnější podědit atributy a metody přidělené osobám (nadtyp) a rozšířit studenty (podtyp) o specifický atribut zaručující uchování a získání názvu školy, jak je ukázáno na př. 2. Pokud dojde ke změně atributů nebo metod v typu osoba_typ (obecně v nadtypu), projeví se tato změna i v podtypu student_typ (obecně v podtypech).

Př. 2 student dědí charakteristiky osoby a přidává vlastní atribut

```
CREATE TYPE student_typ UNDER osoba_typ (
    škola          varchar2(30));
```

Objektový typ může mít více podtypů, kromě student_typ není problém definovat další typ nebo typy reprezentující například důchodce nebo zaměstnance. Také není problém dědit zprostředkovaně, tedy vytvořit podtyp student_střední_školy_typ, který by dědil z typu student_typ a zprostředkovaně tedy z typu osoba_typ. V podtypech je obvykle možné předefinovat funkce poděděné z nadtypu. Podtyp může mít jako atribut svůj nadtypu.

Každý typ může dědit z nejvýše jednoho nadtypu. Není tedy možné definovat například typy pracující_typ a student_typ a děděním získat objektový typ pracující_student_typ, který by sdružoval atributy i metody obou nadtypů.

Klíčová slova *AS OBJECT* v př. 1 znamenají, že typ nedědí atributy ani metody po jiném typu, je tzv. kořenovým typem, naopak *UNDER* v př. 2 ukazuje, ze kterého objektového typu má vytvářený typ dědit. Lze výslovně zakázat další dědění konkrétního typu přidáním klíčového slova *FINAL*, případně dědění povolit přidáním *NOT FINAL*. Další možností je přidání *NOT INSTANTIABLE* nebo *INSTANTIABLE*. *NOT INSTANTIABLE* znamená, že objekt nemůže vytvořit konkrétní instance. Tato vlastnost se použije v případě, že typ nemá existovat samostatně, ale jedná se pouze o abstraktní typ sloužící čistě jen pro dědění svými podtypy. Standardně je tvorba instancí povolena, nicméně výslovně ji lze povolit klíčovým slovem *INSTANTIABLE* (použitelné pro lepší přehlednost nebo u nestandardně nastavených systémů). Ukázka tvorby obecného abstraktního typu sloužícího jen pro dědění a konkrétního typu, ze kterého už naopak nelze dále dědit, je vidět na př. 3.

Př. 3 tvorba abstraktního a konkrétního datového typu

```
CREATE TYPE abstraktní_typ AS OBJECT (
    společný_atribut_1      varchar(10),
    společný_atribut_2      varchar(10),
    MEMBER FUNCTION obecná_funkce RETURN varchar2)
    NOT FINAL NOT INSTANTIABLE;
CREATE TYPE konkrétní_typ UNDER abstraktní_typ (
    konkrétní_atribut_1     varchar2(10),
    MEMBER FUNCTION konkrétní_funkce RETURN varchar2)
    FINAL INSTANTIABLE;
```

2.2. Objekty

Objekty jsou konkrétní instance vytvořené na základě výše popsaných objektových typů. Na př. 4 je vidět ukázka jednoduché konstrukce objektu. Relační tabulka studenti má dva sloupce. Sloupec student používá objektový typ místo datového typu (takto použitému typu se říká sloupcový typ nebo typ sloupce). Druhý sloupec další_atribut je datového typu *VARCHAR2*. Příkazem *INSERT* je do tabulky vložen jeden řádek obsahující údaje o studentovi a textový řetězec. Na tomto příkladu je také vidět, jak vytvořit jednoduchou hnízděnou strukturu, protože strukturovaná data o studentovi zabírají pouze jedno políčko jednoho řádku tabulky studenti. Příkaz *SELECT* s.student.jméno, s.student.příjmení, s.student.škola, další_atribut *FROM* studenti s; vypíše jména, příjmení, školy a další_atribut všech uložených studentů.

Př. 4 tvorba tabulky a naplnění daty

```
CREATE TABLE studenti (
    student             student_typ,
    další_atribut       varchar2(20));
INSERT INTO studenti VALUES (
    student_typ('1','Jan','Novák','jnovak@lorem.ipsum', 'VŠE'),
    'Lorem ipsum');
```

Neznámý údaj je obvykle vyznačen hodnotou null. Sloupec tabulky, atribut objektu, kolekce nebo prvek kolekce může mít hodnotu null, pokud byl vytvořen tak, aby obsahoval null nebo vytvořený vůbec nebyl. Hodnota null je obvykle později nahrazena skutečnou hodnotou. Je potřeba rozlišovat objekty s hodnotou null a objekty, jejichž všechny atributy mají hodnotu null. Není problém pracovat s vytvořeným objektem, jehož všechny atributy mají hodnotu null, protože má alokovaný prostor a tyto hodnoty atributů mohou být měněny a metody objektu volány. Objekt s hodnotou null nemá alokovaný žádný prostor, a tedy nelze pracovat ani s jeho atributy, ani s metodami.

Pro přístup k atributům a metodám jednotlivých objektů se používá tzv. tečková notace, kdy například pro přístup k metodě get_xy instance o typu osoba se použije zápis o.get_xy.

2.2.1. Objektové tabulky

Zvláštním typem tabulky jsou tzv. objektové tabulky (object tables). Každý řádek takovéto tabulky reprezentuje objekt (instanci) některého z objektových typů a jedná se o tzv. typ řádku. To samozřejmě znamená, že každý řádek má vlastní unikátní OID. Jak je vidět na př. 5 objektové tabulky se tvoří příkazem *CREATE TABLE*. Na objektové tabulky je možné se dívat dvěma způsoby. Jednak jako na tabulky tvořené jedním sloupcem, kde každý řádek je instancí daného objektového typu nebo jako na tabulku s více sloupci, kde sloupce jsou tvořeny jednotlivými atributy daného objektového typu. K vypsání jmen je tedy možné použít jak příkaz *SELECT* jméno *FROM* osoby_ot;, tak příkaz *SELECT* o.jméno *FROM* osoby_ot o;.

Př. 5 tvorba a naplnění objektové tabulky

```
CREATE TABLE osoby_ot OF osoba_typ;
INSERT INTO osoby_ot VALUES (
    osoba_typ('2','Petr','Pokorný','ppokorny@lorem.ipsum'));
```

2.2.2. Substituce typů

Protože podtypy jsou v databázi Oracle jen modifikace svého nadtypu, je možné do objektové tabulky vkládat nejen řádky daného typu, ale i případných podtypů. Jak ukazuje př. 6, dá se do tabulky osoby_ot vložit řádek odpovídající typu student_typ, který je potomkem typu osoba_typ. Jedná se o tzv. substituci typů (types substitutability). Případné změny v jednotlivých nadtypech i podtypech se promítnou i do struktury objektových tabulek. Podobně je možné provádět také substituci sloupcových typů v klasických relačních tabulkách.

Př. 6 vkládání podtypu jako řádku do objektové tabulky tvořené řádkovým typem nadtypu

```
INSERT INTO osoby_ot VALUES (
    student_typ('1','Jan','Novák','jnovak@lorem.ipsum','VŠE'));
```

Substituce typů může být zakázána klíčovými slovy *NOT SUBSTITUABLE AT ALL LEVELS*. Omezení se provádí klíčovými slovy *IS OF*. V př. 7 je vytvořena tabulka osoby_nes, která má zakázanou substituci typů.

Př. 7 tvorba tabulek se zakázanou a s omezenou substitucí typů

```
CREATE TABLE osoby_nes OF osoba_typ
    NOT SUBSTITUTABLE AT ALL LEVELS;
```

2.3. Objektové metody

Objektové metody (object methods) jsou procedury a funkce, které zajišťují chování objektu. Deklarují se v definici typu (viz jednoduchá metoda get_idno v př. 1). V databázi Oracle existuje pět hlavních druhů metod. Jsou to členské metody, metody pro srovnání objektů, statické metody, konstruktory a externě implementované metody. Metody mohou být psány v PL/SQL[13], Javě nebo v jiném programovacím jazyku.

2.3.1. Členské metody

Členské metody (member methods) slouží k přístupu aplikace k datům instance objektu. Členským metodám je automaticky předáván parametr *SELF*, který ukazuje na instanci objektu, která metodu provádí. Pro zjednodušení není potřeba udávat *SELF* u každého atributu dané instance.

Jednoduchá členská metoda je ukázána v př. 8. Je definován objektový typ tlouštík_typ, u kterého je uchováváno identifikační číslo, výška a váha. Dále je deklarována funkce bmi, která vrací hodnotu body mass indexu[14]. Oba ukázané způsoby definování návratové hodnoty (se *SELF* i bez) jsou ekvivalentní.

Př. 8 tvorba nové členské funkce, varianta s i bez explicitně napsaného self

```
CREATE TYPE tlouštík_typ AS OBJECT (
    idno            number,
    výška           number,
    váha            number,
    MEMBER FUNCTION bmi RETURN number);
CREATE TYPE BODY tlouštík_typ AS
    MEMBER FUNCTION bmi RETURN number IS
    BEGIN
        -- RETURN SELF.váha / (SELF.výška * SELF.výška);
        RETURN váha / (výška * výška);
    END;
END;
```

[13] PL/SQL je procedurální rozšíření vytvořené firmou Oracle. Procedurální znamená, že je zadaný postup, jak dojít k výsledku. Procedurálními jazyky jsou například Java nebo C, neprocedurálním jazykem je například SQL.

[1]

[14] BMI je jednoduchý ukazatel obezity. Spočítá se jako podíl hmotnosti člověka v kilogramech a jeho výšky v metrech na druhou (BMI = $\frac{hmotnost(kg)}{výška(m)^2}$).

2.3.2. Metody pro třídění objektů

Hodnoty základních datových typů jako *CHAR* nebo *INTEGER* mají dané pořadí, které dovoluje jejich velikosti srovnávat. Objektové typy žádné předdefinované hodnoty pro srovnání nemají, proto je možné deklarovat metody, které jsou schopny tyto objekty porovnat podle velikosti. Databáze Oracle nabízí dva typy metod pro třídění objektů. Jsou to map metody (map methods) a order metody (order methods).

Typ může definovat pouze jednu metodu pro porovnání velikosti objektů, nelze definovat pro stejný typ map i order metodu. Pokud u kořenového typu[15] není definována metoda pro srovnání objektů, nelze ji definovat ani u podtypů. Pokud je order nebo map metoda deklarovaná, volá se (call) automaticky pokaždé, když mají být objekty porovnány. Není tedy potřeba metody explicitně volat, ale stačí zapsat porovnání jako například objekt_1 > object_2.

Map metody poskytují základ pro porovnání objektů tím, že mapují instance objektu na některý ze skalárních typů (*DATE, NUMBER, VARCHAR2*) nebo na SQL datové typy jakým je například *CHAR*. Pokud je definována map metoda nadtypu, lze ji předefinovat podtypem. Ukázka map metody je vidět na př. 9, kde dochází ke srovnání typu tlouštík_typ na základě hodnoty BMI.

Př. 9 tvorba map metody

```
CREATE TYPE tlouštík_map_typ AS OBJECT (
    idno        number,
    výška       number,
    váha        number,
    MAP MEMBER FUNCTION bmi RETURN number);
CREATE TYPE BODY tlouštík_map_typ AS
    MAP MEMBER FUNCTION bmi RETURN number IS
    BEGIN
        RETURN váha / (výška * výška);
    END;
END;
```

Order metody se používají tam, kde schéma srovnání je příliš komplexní na použití map metody. Order metoda přímo porovnává dva objekty. Na rozdíl od map metod nedojde k přesnému vyjádření velikosti, ale pouze k vyjádření toho, zda je velikost daného objektu větší, menší nebo rovna druhému objektu. Order metoda dostává jako parametr druhý objekt a

[15] Kořenový typ nedědí (nemá žádný nadtyp).

vrací kladnou (pokud je daný *SELF* objekt větší než druhý objekt), zápornou (daný objekt je menší než objekt z parametru) nebo nulovou hodnotu (objekty mají stejnou velikost). Na př. 10 je vidět tvorba order metody. Opět jsou srovnávány hodnoty BMI typu tlouštík_typ.

Př. 10 tvorba order metody

```
CREATE TYPE tlouštík_order_typ AS OBJECT (
    idno        number,
    výška       number,
    váha        number,
    ORDER MEMBER FUNCTION srovnej (t tlouštík_order_typ) RETURN integer);
CREATE TYPE BODY tlouštík_order_typ AS
    ORDER MEMBER FUNCTION srovnej (t tlouštík_order_typ) RETURN integer IS
    BEGIN
        IF váha / (výška*výška) < t.váha / (t.výška*t.výška) THEN
            RETURN -1;
        ELSIF  váha / (výška*výška) > t.váha / (t.výška*t.výška) THEN
            RETURN 1;
        ELSE
            RETURN 0;
        ENDIF;
    END;
END;
```

Pro porovnání více objektů (objekt_1 > object_2 > … > objekt_x) je vhodnější map metoda, protože na jedno zavolání jsou získány skalární hodnoty všech objektů. Order metoda se volá pro každé porovnání dvou objektů znovu.

2.3.3. Statické metody

Statické metody jsou volány s typem objektu (např. objektový_typ.statická_metoda(parametr)) a ne s konkrétní instancí. Statickým metodám (static methods) není předávaný *SELF* jako první parametr a všechny metody, které nevyužívají parametr *SELF* by měly být deklarovány jako statické.

2.3.4. Konstruktory

Každý typ objektu má metodu konstruktor (constructor), která je implicitně definována databází a pojmenovaná podle jména typu se všemi atributy jako parametry. Konstruktor je metoda, která vrací nově vytvořenou instanci objektu a nastaví její atributy.

Systémem definovanému konstruktoru musí být dodána hodnota pro každý atribut. Je možné vytvořit uživatelem definované konstruktory, kterým jsou dány jen některé atributy a jiným se přiřadí například standardní hodnota. Při pozdější změně typu (například po přidání nového atributu) se systémem definovaný konstruktor automaticky změní tak, že je potřeba ho volat i s hodnotou pro nový atribut, naopak uživatelem definované konstruktory se nemění.

2.3.5. Externě implementované metody

Jazyk PL/SQL může být použit pro zavolání externích podprogramů napsaných v jiných programovacích jazycích. Metody napsány v PL/SQL a v Javě jsou uloženy v databázi, metody napsané v jiném jazyku (např. v C) jsou uchovávány externě.

2.3.6. Dědičnost, předefinování a přetížení metod

Objektové typy dědí metody společně s atributy. Tyto metody nemusí vyhovovat podtypům, proto je obvykle možné je předefinovat (override). Předefinováním dojde ke změně zdrojového kódu zděděné metody při zachování stejného jména i struktury parametrů. Předefinování musí být vyznačeno klíčovým slovem *OVERRIDE*.

V případě, že je potřeba mít několik variant metody, které implementují podobné chování, ale liší se v parametrech, je možné metody tzv. přetížit. K přetížení dojde, pokud je pro typ vytvořena nová metoda se stejným jménem, ale odlišnými parametry. V takovém případě není možné rozeznávat metody na základě jejich jména, rozeznávají se podle signatur, které zahrnují nejen jméno metody, ale i počet, typy a pořadí parametrů.

Podobně jako objektové typy, lze i metody definovat jako *FINAL* nebo *NOT FINAL*. *FINAL* znamená, že metoda nemůže být v podtypu předefinována, *NOT FINAL* naopak výslovně umožňuje předefinování metody v podtypech. Metoda může být také deklarována jako *NOT INSTANTIABLE*, pokud ji není vhodné v daném typu implementovat a naopak se očekává, že každý podtyp metodu předefinuje. Typ, který obsahuje *NOT INSTANTIABLE* metodu musí být sám označen jako *NOT INSTANTIABLE*.

Na př. 11 je ukázáno jak v databázi Oracle deklarovat předefinování a přetížení. U typu obrazec_typ je deklarováno pouze to, že podtypy mají mít metodu obsah. Podtyp čtverec_typ dědí z typu obrazec_typ atribut popis a hlavně metodu obsah, kterou musí

předefinovat (nebo být také označený jako *NOT INSTANTIABLE*). Zároveň je metoda obsah přetížena o další metodu obsah s parametrem v datového typu *VARCHAR2*, který reprezentuje textový komentář k obsahu čtverce.

Př. 11 dědičnost, přetížení a předefinování metod

```
CREATE TYPE obrazec_typ AS OBJECT (
    popis           varchar2(50),
    NOT INSTANTIABLE MEMBER FUNCTION obsah RETURN number)
    NOT FINAL NOT INSTANTIABLE;
CREATE TYPE čtverec_typ UNDER obrazec_typ (
    strana          number,
    MEMBER FUNCTION obsah(v varchar2) RETURN varchar2,
    OVERRIDING MEMBER FUNCTION obsah RETURN number);
CREATE TYPE BODY čtverec_typ AS
    MEMBER FUNCTION obsah(v varchar2) RETURN varchar2  IS
    BEGIN
        return v || TO_CHAR(strana*strana);
    END;
    OVERRIDING MEMBER FUNCTION obsah RETURN number IS
    BEGIN
        RETURN strana*strana;
    END;
END;
```

2.4. Reference

Datový typ *REF* je logický ukazatel na řádkový typ vytvořený na základě OID objektu, na který je ukazováno. Ukazatele modelují vztahy mezi objekty bez nutnosti použití cizích klíčů. *REF* je možné použít k získání, prozkoumání i změně objektů, na které ukazuje. Pro získání objektu z ukazatele slouží *DEREF*.

V př. 12 je ukázána tvorba tabulky uchovávající informace o jménech zaměstnanců a jejich vedoucích. Pro odkázání se na vedoucího je použit ukazatel. Zaměstnanec Petr Pracovitý má vedoucího Jana Vysokého, který žádného svého vedoucího nemá. V příkladu je i dotaz na jména zaměstnanců a jména jejich vedoucích pomocí příkazu *SELECT* s využitím *DEREF*.

Př. 12 použití ukazatelů

```
CREATE TYPE zaměstnanec_typ AS OBJECT (
    jméno           varchar2(30),
```

```
    vedoucí          REF    zaměstnanec_typ);
CREATE TABLE zaměstnanci OF zaměstnanec_typ;
INSERT INTO zaměstnanci VALUES (
    zaměstnanec_typ ('Jan Vysoký', NULL));
INSERT INTO zaměstnanci
    SELECT zaměstnanec_typ('Petr Pracovitý', REF(z))
        FROM zaměstnanci z
        WHERE z.jméno = 'Jan Vysoký';
SELECT jméno, DEREF(vedoucí).jméno FROM zaměstnanci;
```

Ukazatele je možné omezit nejen na typ, ale i na konkrétní objektovou tabulku, do které mají ukazovat pomocí klíčových slov *SCOPE IS*. Tato objektová tabulka musí být tvořena řádkovými typy stejnými jako je typ, na který ukazatel ukazuje, případně jejich potomky (jde o další použití substituce typů). Takto omezený ukazatel se nazývá scoped ref.

Dangling ref je ukazatel ukazující na již neexistující objekt. K testování existence objektů, na které je ukazováno, slouží *IS DANGLING* (příp. *IS NOT DANGLING*). Například příkaz *SELECT* jméno, *DEREF*(vedoucí).jméno *FROM* zaměstnanci *WHERE* vedoucí *IS NOT DANGLING*; aplikovaný na tabulku z př. 12 vybere jen zaměstnance, kteří mají vedoucího).

2.5. Kolekce

Databáze Oracle podporuje dva typy kolekcí. Jedním typem je varray, který představuje očíslovaný seznam prvků o dané maximální velikosti (tuto velikost lze později změnit), druhým typem je hnízděná tabulka (nested table) tvořící neuspořádanou množinu prvků. Hnízděná tabulka je efektivnější z hlediska dotazů a DML operací. Lze tvořit složitější struktury vnořováním kolekce do kolekce.

Hnízděné tabulky mají definovány funkce pro práci s množinami jako počet prvků (*CARDINALITY*), ekvivalence (=), sjednocení (*MULTISET UNION*), průnik (*MULTISET INTERSECTION*) a rozdíl (*MULTISET EXCERPT*). Vytvoření hnízděné tabulky je ukázáno na př. 14. Tabulka sdružení ukládá atributy idno, název a osoby, které do sdružení patří. Takových osob může být samozřejmě několik, proto se hodí kolekce, v tomto případě hnízděná tabulka. Do tabulky sdružení je vloženo vzorové První sdružení se dvěma členy.

Př. 13 tvorba hnízděné tabulky a její naplnění daty

```
CREATE TYPE osoba2_typ AS OBJECT (
    idno            number,
```

```
    jméno          varchar2(30));
CREATE TYPE osoby_typ AS TABLE OF osoba2_typ;
CREATE TABLE sdružení (
    idno           number,
    název          varchar2(30),
    osoby          osoby_typ)
    NESTED TABLE osoby STORE AS osoby_nt;
INSERT INTO sdružení VALUES (
    1, 'První sdružení', osoby_typ(
        Osoba2_typ(1, 'Jan Novák'),
        Osoba2_typ(2, 'Petr Pokorný')));
```

Oproti hnízděné tabulce má každý prvek uložený ve varray číselný index vypovídající o pozici v poli, na které se nachází. Varray má při definici specifikované maximální množství prvků, které lze později změnit pomocí příkazu *ALTER*. Varray nemá definované funkce pro práce s množinami jako sjednocení, průnik a rozdíl.

Identická struktura jako v příkladu s hnízděnými tabulkami, ale tentokrát za použití varray je vytvořena v př. 14.

Př. 14 tvorba a naplnění daty varray

```
CREATE TYPE osoby_typ AS VARRAY(5) OF osoba2_typ;
CREATE TABLE sdružení (
    číslo          number,
    název          varchar2(30),
    osoby          osoby_typ);
INSERT INTO sdružení VALUES (
    1, 'První sdružení', osoby_typ(
        Osoba2_typ(1, 'Jan Novák'),
        Osoba2_typ(2, 'Petr Pokorný')));
```

Jednoduchý dotaz select * from sdružení; vypíše všechny sloupce tabulky sdružení včetně zaměstnanců, ale v poměrně nehezkém formátu, který je nutné dále zpracovat. Lepší výsledek dodá příkaz *SELECT* s.název,t.* *FROM* sdružení s, *TABLE*(s.osoby) t;. Tento příkaz vypíše každou osobu na vlastní řádek spolu s názvem sdružení, do kterého patří. Výrazu *TABLE* je třeba dodat právě jednu kolekci. Lze vnořovat kolekce do kolekcí a tvořit tak kolekce kolekcí.

2.6. Velké objekty

Velké objekty (large objects - LOB) jsou datové typy navržené pro ukládání velkých množství dat. Používají se pro ukládání polostrukturovaných dat (např. XML dokument zpracovaný aplikací) nebo nestrukturovaných dat (např. obrázek).

Databáze Oracle podporuje čtyři typy velkých objektů. *BLOB*, *CLOB*, *NCLOB* a *BFILE*. *BLOB* uchovává jakýkoliv typ dat v binárním formátu (např. multimédia). *CLOB* a *NCLOB* slouží k uchování dlouhých textových řetězců s tím, že *NCLOB* je určený k podpoře národních znakových sad. *BFILE* slouží k uchování souboru mimo databázi, pokud je databáze schopna k tomuto souboru přistupovat.

S LOBy je používán tzv. LOB lokátor (LOB locator), který odkazuje na uloženou hodnotu LOBu. Hodnota LOBu představuje uložená data.

Pro práci s multimediálními aplikacemi je určený Oracle interMedia, který používá LOBy k uchování a práci s multimediálními daty, jakými jsou dokumenty, zvuky, obrázky a video.

2.7. Shrnutí

Databáze Oracle nějakým způsobem podporuje každý ze základních objektových principů. Objekty mají unikátní identitu (OID) a je implementována podpora použití ukazatelů. Uživatel může definovat vlastní typy a pracovat s nimi stejně jako s předdefinovanými typy, ale není možné vytvářet přímo odlišující datové typy. Oracle nerozlišuje mezi typem a třídou, existuje pouze typ, ale je možné ho modifikovat tak, aby pracoval jako třída. Omezením je, že nelze dědit po více jak jednom předkovi. Zapouzdření objektů není úplné. Oracle sice podporuje tvorbu vlastních metod pro definici chování objektu, ale je možné k atributům přistupovat přímo. Metody mohou být děděny, přetěžovány i předefinovány, polymorfizmus je implementován.

3. Realizace objektových principů v IBM DB2

První dva velké projekty relačních databází byly vyvíjeny na University of California v Berkeley a ve firmě IBM. První relační databáze firmy IBM se jmenovala System-R (System Reational) a jejím jazykem byl Sequel[16]. Tehdejší management společnosti IBM nebyl přesvědčen o výhodnosti nasazení relačních databázových systémů na místo tehdejších technologií, proto nakonec došlo k tomu, že firma Oracle uvedla na trh relační databázový systém dříve (i přes původní inspiraci firmy Oracle databází System-R). Na trh byl relační databázový systém firmy IBM uveden pod názvem DB2, v současné době je prodávána devátá verze.

Podobně jako v předchozí části, tato kapitola vychází hlavně z dokumentace dostupné buď na webových stránkách Informačního centra DB2[17], nebo v souborech typu pdf[18]. K testování a pokusům je použit systém DB2 Express-C, který je dostupný zdarma.

Stejně jako v kapitole o DBS Oracle je zde ukázáno hlavně uplatnění jednotlivých objektových principů pomocí příkazu *CREATE* a není podrobněji rozebírána modifikace a odstranění jednotlivých typů, objektů apod.

[12] [13]

3.1. Datové Typy

DB2 podporuje jak odlišující datové typy (distinct types), tak strukturované datové typy.

Odlišující datové typy (jak je zmíněno v kapitole 1.5.1) představují datové typy se stejnou reprezentací v databázi, které nejsou navzájem kompatibilní, nemá smysl jejich vzájemné porovnání (např. výška a váha). Odlišující typy po svém zdroji nedědí operace (protože nemusí mít u nového typu smysl), nicméně pokud jsou přidána klíčová slova *WITH COMPARISONS*, dědí se porovnávací funkce a lze pak srovnávat velikost jednotlivých instancí odlišujícího typu. *WITH COMPARISONS* nelze použít u velkých objektů, *LONG VARCHAR* a *LONG VARGRAPHIC*. Na příkladu př. 15 je ukázáno jak vytvořit odlišný typ.

[16] Předchůdce jazyka SQL.

[17] http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.doc/welcome.htm [ov. 12.8.2007].

[18] Dokumentace v pdf verzi je k dispozici na stránce http://www-1.ibm.com/support/docview.wss?rs=71&uid=swg27009552 [ov. 11.8.2007].

Př. 15 vytvoření odlišujícího typu

CREATE DISTINCT TYPE *odlišný_typ* AS *smallint* WITH COMPARISONS*;*

Strukturované datové typy plní stejnou funkci jako objektové typy v DBS Oracle. Syntaxe příkazu *CREATE* se liší. Na příkladu př. 16 je vidět tvorba podobné struktury jako v př. 1 na straně 26, tedy jednoduchý strukturovaný typ pro uchování dat o osobách s atributy idno, jméno, příjmení, email a metodou get_idno vracející idno (tělo metody se musí dále definovat).

Př. 16 vytvoření strukturovaného datového typu

CREATE TYPE *osoba_typ* AS *(*

idno int,
jméno varchar(20),
příjmení varchar(20),
email varchar(25))

MODE DB2SQL
METHOD *get_idno()* RETURNS *int;*

Také v DB2 je možné deklarovat strukturovaný typ jako *NOT INSTANTIABLE* (případně explicitně určit *INSTANTIABLE*), tedy, že není možné vytvořit konkrétní instanci objektu. Je možné dědit po jednom předkovi pomocí klíčového slova *UNDER*, jak je ukázáno na př. 17.

Př. 17 tvorba podtypu

CREATE TYPE *student_typ* UNDER *osoba_typ* AS *(*

škola varchar(30))

MODE DB2SQL*;*

3.2. Tabulky

Databázový systém DB2 podporuje dva základní typy tabulek. Klasické relační tabulky a typizované tabulky (typed tables), které jsou podobné dříve popsaným objektovým tabulkám. Instance strukturovaných typů mohou být uloženy buď jako sloupce klasických tabulek, nebo jako řádky typizovaných tabulek.

Konstrukce relačních tabulek je stejná jako v databázi Oracle, ale plnění daty pomocí příkazu *INSERT* je syntakticky jiné a náročnější na psaní, jak ukazuje př. 18, navíc DB2 používá dvě tečky místo jedné. V tabulkách lze substituovat typy svými podtypy. Dotaz na jména, příjmení a hodnotu dalšího atributu provede příkaz *SELECT* osoba..jméno *AS* jméno, osoba..příjmení *AS* příjmení, další_atribut *FROM* osoby.

Př. 18 tvorba a naplnění daty tabulky se strukturovaným typem jako sloupcem

```
CREATE TABLE osoby (
    osoba osoba_typ,
    další_atribut varchar(20));
INSERT INTO osoby VALUES
    (osoba_typ()..idno(1)
    ..jméno('Jan')
    ..příjmení('Novák')
    ..email('jnovak@lorem.ipsum'),
    'Lorem ipsum');
INSERT INTO osoby VALUES
    (student_typ()..idno(2)
    ..jméno('Petr')
    ..příjmení('Pilný')
    ..email('ppilny@lorem.ipsum')
    ..škola('VŠE')
    'Lorem ipsum');
```

Při tvorbě typizovaných tabulek příkazem *CREATE* je na př. 19 vidět první odlišnost od objektových tabulek Oracle. Je potřeba specifikovat sloupec objektového identifikátoru (v tomto případě pojmenovaném oid). Následně je při vkládání dat nutné zadat oid, naštěstí není potřeba vymýšlet každou hodnotu zvlášť, ale lze využít sekvenci nebo funkci *GENERATE_UNIQUE*(), případně zkombinovat jedno z toho ještě s triggerem. Protože typizovaná tabulka přebírá definici sloupců z atributů daného typu, lze klást dotazy přímo bez použití tečkové notace. Dotaz *SELECT* jméno, příjmení *FROM* osoby_tt; tedy vypíše jména a příjmení.

Př. 19 tvorba a naplnění daty typizované tabulky

```
CREATE TABLE osoby_tt OF osoba_typ
    (REF IS oid USER GENERATED);
INSERT INTO osoby_tt VALUES
    (osoba_typ('a'), 1, 'Jan','Novák','jnovak@lorem.ipsum');
INSERT INTO osoby_tt VALUES
    (osoba_typ(GENERATE_UNIQUE()), 1, 'Petr', 'Smrček',
    'psmrcek@lorem.ipsum');
```

Do typizované tabulky nelze vkládat řádky reprezentující podtypy, na místo toho lze vytvořit podtabulky (subtables). Příkazy *SELECT*, *UPDATE* a *DELETE* se aplikují i na všechny podtabulky, případné omezení těchto příkazů se provede pomocí klíčového slova *ONLY* nebo *IS OF*. Na př. 20 je vidět vytvoření a naplnění daty typizované tabulky

studenti_tt. Následný příkaz *SELECT* jméno, příjmení *FROM* osoby_tt; vypíše jak jména a příjmení osob, tak i studentů uložených v tabulce studenti_tt. Pokud je potřeba vypsat tabulku i s obsahem podtabulek včetně sloupců specifikovaných v podtabulkách (resp. podtypech), použije se klíčové slovo *OUTER* (Například *SELECT * FROM OUTER*(osoby_tt);).

Př. 20 vytvoření podtabulky a její naplnění daty

```
CREATE TABLE studenti_tt OF student_typ UNDER osoby_tt
    INHERIT SELECT PRIVILEGES;
INSERT INTO studenti_tt VALUES
    (student_typ(GENERATE_UNIQUE()),
    1, 'Ondřej', 'Pilný', 'opilny@lorem.ipsum', 'VŠE');
```

3.3. Reference

Každý řádek v typizované tabulce má svůj referenční sloupec (v příkladech v této práci pojmenovaný oid). Ukazatel v DB2 ukazuje na hodnotu v tomto sloupci, a proto je tvorba ukazatelů názornější a jednodušší na představivost než v databázi Oracle, protože odpadá nutnost použití funkce *REF* v DML příkazech. Na př. 21 je vidět tvorba tabulky zaměstnanců, z nichž každý má své jméno a případného vedoucího. Odkaz je omezený na tabulku zaměstnanci pomocí klíčového slova *SCOPE*. Vypsání jmen zaměstnanců a jejich přímých nadřízených je pak možné pomocí příkazu *SELECT* jméno, vedoucí->jméno *AS* vedoucí *FROM* zaměstnanci;. Operátor -> představuje funkci *DEREF*, kterou lze také použít (stejného výsledku by tedy bylo dosaženo i příkazem *SELECT* jméno, *DEREF*(vedoucí)..jméno *AS* vedoucí *FROM* zaměstnanci;).

Př. 21 tvorba tabulky s ukazateli a její naplnění daty

```
CREATE TYPE zaměstnanec_typ AS (
    jméno varchar(30),
    vedoucí REF(zaměstnanec_typ))
    MODE DB2SQL;
CREATE TABLE zaměstnanci OF zaměstnanec_typ
    (ref is oid user generated, vedoucí with option scope zaměstnanci);
INSERT INTO zaměstnanci VALUES
    (zaměstnanec_typ(GENERATE_UNIQUE()), 'Jan Novák', NULL);
INSERT INTO zaměstnanci VALUES
    (zaměstnanec_typ(GENERATE_UNIQUE()), 'Petr Pilný',
    (SELECT oid FROM zaměstnanci WHERE jméno = 'Jan Novák'));
```

3.4. Metody

Uživatelem definované metody fungují podobně jako v databázovém systému Oracle. Metody se dědí a je možné je předefinovat i přetížit na základě počtu a druhu parametrů i návratové hodnoty. V DB2 není tělo typu, který by obsahoval definici metod, ale tělo metody, při jejíž deklaraci se specifikuje typ, pro který má metoda pracovat. Není implementováno přímé porovnávání uživatelem vytvořených typů tak, jak je tomu v databázi Oracle. Samozřejmě lze metody vracející hodnotu představující hodnotu dané instance, ale uživatel musí sám tyto metody volat. DB2 také podporuje definici uživatelem definovaných funkcí přes externí programovací jazyk, jako je C nebo Java.

Na příkladu př. 22 je ukázáno, jak předefinovat metody v podtypech. Typ člověk má atribut jméno a metodu příjem. Člověk je obecně považovaný za nezaměstnaného a funkce příjem vrací 0. Zaměstnanec dostává plat a vrací hodnotu atributu plat. Manažer dostává 1,5 násobek základního platu. Metoda příjem je předefinována pro každý typ tak, aby vracela správně vypočítanou velikost platu. Následný dotaz na jména lidí s platem vyšším než 900, jehož výsledek lze získat příkazem *SELECT* osoba..jméno *AS* jméno *FROM* lidé *WHERE* osoba..příjem() >900;, vypíše Petra Pilného a Jana Nováka. Dotaz na lidi s platem převyšujícím 2000 vrátí pouze Jana Nováka, který díky svému manažerskému bonusu má příjem 2250.

Př. 22 předefinování metod v podtypech

CREATE TYPE člověk *AS* (jméno varchar(30)) *MODE DB2SQL METHOD* příjem() *RETURNS* int;

CREATE METHOD příjem() *RETURNS* int *FOR* člověk *RETURN* (0);

CREATE TYPE zaměstnanec *UNDER* člověk *AS* (plat int) *MODE DB2SQL OVERRIDING METHOD* příjem() *RETURNS* int;

CREATE METHOD příjem() *RETURNS* int *FOR* zaměstnanec *RETURN* (*SELF*..plat);

CREATE TYPE manažer *UNDER* zaměstnanec *MODE DB2SQL;*

OVERRIDING METHOD příjem() *RETURNS* int;

CREATE METHOD příjem() *RETURNS* int *FOR* manažer *RETURN* (*SELF*..plat*1.5);

CREATE TABLE lidé (číslo int, osoba člověk);

INSERT INTO lidé *VALUES* (1,člověk()..jméno('Jiří Smrček'));

INSERT INTO lidé *VALUES* (2,zaměstnanec()..jméno('Petr Pilný')..plat(1000));

INSERT INTO lidé *VALUES* (3,zaměstnanec()..jméno('Jan Líný')..plat(800));

INSERT INTO lidé *VALUES* (4,manažer()..jméno('Jan Novák')..plat(1500));

3.5. Velké objekty

DB2 podporuje tři datové typy pro velké objekty, standardní *CLOB* a *BLOB* a dále double-byte LOB (*DBLOB*). LOB lokátor je v DB2 proměnná, jejíž hodnota reprezentuje jednu LOB hodnotu. Aplikace může uložit LOB hodnotu do LOB lokátoru a provést na něm operaci, jako například funkci *SUBSTR*, *CONCAT*, *VALUE*, *LENGTH*, prohledat hodnotu užitím *LIKE*, aplikovat uživatelem definovanou funkci aj. LOB lokátor reprezentuje hodnotu a ne řádek nebo polohu v databázi. LOB lokátory nejsou perzistentní, existují pouze po dobu konkrétní transakce.

3.6. Shrnutí

DB2 nepodporuje kolekce, což přináší omezení v tvorbě hnízděných struktur zejména v případech, kdy není dopředu jisté, kolik prvků hnízděná struktura bude obsahovat. Na druhou stranu IBM DB2 přímo podporuje odlišující datové typy, navíc lze tvořit uživatelem definované strukturované typy s vlastním OID. Nepříjemnou nutnost tvorby vlastního OID lze automatizovat použitím funkce nebo sekvence, případně pro úplné pohodlí kombinovat s triggerem. Na OID lze bez problémů, a přehledněji než v databázi Oracle, odkazovat pomocí ukazatelů. Dále je možné tvořit uživatelem definované metody, které lze navíc také zdědit, přetížit i předefinovat. Nelze dědit po více předcích.

Závěr

Moderní relační databázový systém musí být schopný nabídnout větší možnost abstrakce, což vede k implementaci objektových principů. Praktickým dopadem je potřeba podporovat objekty, jejich identitu a ukazatele, metody pro definici chování objektů a komplexní datové struktury schopné lépe modelovat entity reálného světa. Uživatel musí mít možnost vytvářet vlastní datové struktury a definovat jejich chování.

Kapitoly 2 a 3 se zaměřují na prozkoumání realizace objektových principů v komerčních databázových systémech Oracle a DB2. Obě databáze mají podporu objektových principů na podobné úrovni. DB2 je, na rozdíl od databáze Oracle, schopna přímo podporovat odlišující datové typy. V databázi Oracle lze odlišující datové typy emulovat přes jednoduchý strukturovaný typ. DB2 naopak vůbec neobsahuje kolekce, čímž silně omezuje tvorbu hnízděných datových struktur v případě, že není předem jasné kolik prvků bude hnízděná struktura obsahovat. Pro realizaci zapouzdření je možné v obou systémech tvořit uživatelem definované metody a případně je zdědit, přetížit i předefinovat. Ani jedna databáze neumožňuje dědit po více předcích, ale až na toto omezení je dědičnost plně implementována.

Bohužel oba databázové systémy implementují objektové principy rozdílně. V lepším případě se jedná o drobnou změnu v syntaxi příkazů, v horším případě o celkově trochu jiný přístup k dané charakteristice. Příkladem odlišného přístupu může být různá realizace objektové identity a ukazatelů. Je tedy potřeba se zaměřit hlavně na celkové sjednocení postupů a syntaxe, vytvořit komplexní a široce podporovaný standard. Otázkou je, zda k celkové standardizaci někdy dojde, protože jediný standard pro objektově relační databáze, SQL:1999, není ani po téměř deseti letech dostatečně rozšířený.

Další zkoumání problematiky by se mohlo zaměřit na jeden databázový systém a hlubší prozkoumání možností využití objektových principů v relačních databázích v praxi.

Použitá literatura

[1] POKORNÝ, Jaroslav. *Dotazovací jazyky*. Praha. Nakladatelství Karolinum, 2007, 256 s., ISBN 978-80-246-0497-8.

[2] POKORNÝ, Jaroslav, HALAŠKA, Ivan. *Databázové systémy*. 2. vyd. Praha: Vydavatelství ČVUT, 2004, 148 s. ISBN 80-01-02789-9.

[3] ŠEDA, Miloš. *Databázové systémy*. VUT v Brně, 2002.

http://www.uai.fme.vutbr.cz/~mseda/DBS02_BS.pdf [ov. 29.7.2007]

[4] VOSTROVSKÝ, Václav, MERUNKA, Vojtěch. *Databázové systémy*. ČZU, 1998.

[5] *Codd's rules*. 2004.

http://www.wildewood.co.uk/comp/more/codds_rules.html [ov. 1.8.2007]

[6] Wikipedia – *SQL*.

http://en.wikipedia.org/wiki/SQL [ov. 29.7.2007]

[7] POKORNÝ, Jaroslav. Objektově relační databáze. In: *Datakon 2002*. Masarykova univerzita v Brně, 2002, s. 57-76.

[8] STONEBRAKER, Michael, BROWN, Paul. *Objektově relační SŘBD, analýza příští velké vlny*. Softwarové aplikace a systémy, BEN - technická literatura: Praha, 2000. ISBN 80-901-5074-8 (Softwarové aplikace a systémy), ISBN 80-860-5694-5 (BEN-technická literatura).

[9] *O Oracle*.

http://www.oracle.com/global/cz/corporate/index.html [ov. 6.8.2007]

[10] *Oracle's 30th Anniversary Timeline*.

http://www.oracle.com/oramag/profit/07-may/p27anniv_timeline.pdf [ov. 6.8.2007]

[11] *Oracle Database Documentation Library*.

http://www.oracle.com/pls/db102/homepage [ov. 7.8.2007]

[12] ŽÁK, Karel. *Historie relačních databází*. 2001.

http://www.root.cz/clanky/historie-relacnich-databazi/ [ov. 11.8.2007]

[13] *Informační centrum DB2*.

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.doc/welcome.htm [ov. 12.8.2007]

Terminologický slovník

ADT	Abstrakt data types
	Abstraktní datový typ
BLOB	Binary large object
	Velký binární objekt
CLOB	Charakter large object
	Velký znakový objekt
DB	Database
	Databáze – báze dat – datová základna
DB2	Databázový systém společnosti IBM
DBS	Database system
	Databázový systém
DDL	Data definition language
	Jazyk pro definici dat
DML	Data manipulation language
	Jazyk pro manipulaci s daty
LOB	Large Object
	Polostrukturovaný nebo nestrukturovaný velký objekt
ODL	Object definition language
	Jazyk pro definici dat v OOSŘBD
ODMG	Object data management group
	Skupina zabývající se standardizací na poli OOSŘBD
OID	Object identifier
	Jedinečný identifikátor objektu
OO	Object-oriented
	Objektově orientované
OOSŘBD (OODMBS)	Objektově orientovaný systém řízení báze dat
	Object-oriented database management system

OQL	Object query language
	Dotazovací jazyk pro OOSŘBD
PL/SQL	Procedural language/structured query language
	Procedurální nadstavba jazyka SQL od firmy Oracle
SQL	Structured Query Language
	Velmi rozšířený jazyk používaný u relačních databází
SŘBD (DBMS)	Systém řízení báze dat (database management systém)