# OBJEKTOVĚ ORIENTOVANÉ DATABÁZE

Jaroslav Pokorný

## 1 ÚVOD

Úspěch v používání SŘBD v praxi není zdaleka tak jednoznačný, jak naznačují propagační či komerční periodika komentující šíření jednotlivých produktů. Současný vývoj SŘBD je ovlivněn dalšími technologiemi, z nichž nejznámější (hlavně u nás) se stává objektově orientovaný (OO) přístup.

Podobně jako v jiných oblastech lidské činnosti, novou technologii si do jisté míry vynutila praxe. Pro nové typy aplikací (multimedia, hypermedia, kancelářské systémy požadující bohatě strukturovaný a odkazy provázaný text, obrázky, CAD aplikace apod.) se totiž stávající (většinou relační) SŘBD ukázaly příliš těžkopádné, ba nevhodné. Dlouhé transakce či zpracování verzí (projektů, dokumentace apod.) také nevyhovují operačním charakteristikám současných SŘBD.

OO přístup k databázím reaguje na obecnější směr rozvoje informatiky – OO programování (viz dnes často používané programovací jazyky jako SMALLTALK, C++), ze kterého přebral i většinu pojmů. Dalším zdrojem idejí se staly populární sémantické či konceptuální modely používané v analýze a návrhu při budování IS.

OOSŘBD představují zatím dost mladé odvětví informatiky, uvážíme-li, že první komerční systémy GemStone a VBase (předchůdce ONTOSu) se objevily na trhu r. 1987. V současnosti je na trhu několik desítek komerčních systémů a další se vyvíjejí.

Cílem příspěvku je podat základní informace o podstatě OOSŘBD (odst. 2), vlastnostech OO dotazovacích jazyků (odst. 3) a úvahy nad dalším vývojem OOSŘBD. Podrobné informace lze nalézt např. v [Com91] [US91].

## 2 CHARAKTERISTIKY OOSŘBD

Na rozdíl od dobře definovaného relačního modelu dat (RMD), nelze říci, že existuje jeden OO datový model. To samozřejmě nevylučuje existenci desítek různých teoretických přístupů k objektové orientaci. V praxi jsou OOSŘBD obvykle identifikovány Manifestem skupiny ALTAIR z r. 1989 [Atk89]. Dostupné produkty odrážejí více či méně rysy z Manifestu a mají zatím řadu problémů jak koncepčních, tak i implementačních.

Hlavní změnou v myšlení, kterou přináší OO přístup, je pohled data v databázi. Data nejsou dána pouze hodnotami seskupenými do datových struktur, jako např. relace v RMD, nýbrž jsou vztažena k objektům, které přímo odpovídají entitám reálného světa.

Tento prvek je v datovém inženýrství znám již od poloviny 70. let, kdy se objevily E–R modely (či později obecnější konceptuální a sémantické modely). Entity uvažované v těchto modelech mohly být strukturálně složité a popis struktury byl také hlavním předmětem zájmu těchto modelů.

Modelovací aparát ovšem zaplňoval pouze jedno stádium návrhu uživatelské aplikace. Posléze bylo nutné transformovat takové schéma do komerčního software, kde se entity a sémantika jejich vztahů „ztratily" a zůstala pouze data zobrazená v záznamech či n–ticích relací.

Pojem objektu v OO přístupu jde však ještě dál. Zahrnuje i chování objektu, které např. v relačních systémech je dáno aplikačními programy pracujících ovšem nad relacemi a nikoliv explicitně nad objekty. Tím, že se funkčnost aplikace přenáší přímo k objektům, je možné zredukovat vlastní kód aplikačních programů až na 20% [Eng92].

Zdálo by se, že v prvním plánu stačí vytvořit horní vrstvu relačního systému „kopírující" přístup sémantických modelů a vše je hotovo. Naneštěstí se tento přístup ukázal jako neefektivní. OOSŘBD vyžadují novou technologii i v ukládání dat a v dalších aspektech souvisejících s jejich provozem (dlouhé a krátké transakce na objektech, údržba verzí objektů apod.).

Objektově orientovaný SŘBD by měl vyhovovat dvěma kritériím [Ban88]:

- 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, složité objekty, rozšiřitelnost a výpočetní úplnost.

V [US91] se tato definice rozšiřuje ještě o další rysy související s novými typy aplikací. Jedná se o operační charakteristiky, jako je řízení verzí, kooperativní zpracování, dlouhé a hnízděné transakce, možnosti formulace a použití IO a konstrukce speciálních metod tzv. triggerů (mechanismů spouštění jistých akcí zabezpečujících integritu databáze při aktualizaci).

Starší OOSŘBD, jako např. GemStone, zatím nezahrnují novou operační technologii, zatímco např. ORION podporuje jak verze, tak dlouho trvající transakce.

## 2.1 Identita objektů

Je obvyklé rozlišovat mezi objektem a hodnotou objektu. V OOSŘBD se objekty chápou jako abstraktní objekty vyjádřené pomocí tzv. **identifikátorů objektů** (angl. zkratka je OID). OID zůstává stále stejný, mění se pouze hodnota objektu, která vlastně reprezentuje **stav** objektu a který se může měnit. OID nelze zaměňovat s pojmem klíče v RMD. Tam šlo o hodnotu jednoho nebo více (relačních) atributů, kterou bylo možné bez problémů modifikovat. Hodnotou objektu může být struktura skládající se jednak z objektů, jednak z hodnot běžných primitivních datových typů (**real, integer, string** apod.). V některých OOSŘBD je objektem všechno, tedy i hodnoty. Častější je však případ, kdy hodnoty primitivních typů jsou reprezentovány přímo bez OID. Zase – strukturován není objekt sám, ale jeho hodnota. Hodnotu objektu lze např. popsat množinou atributů. Atributům se také někdy říká **instanční proměnné** (instance variables).

Identita je tedy nezávislá na hodnotě objektu. Dva objekty tedy mohou být identické (jde o týž objekt) nebo si mohou být rovny (mají stejnou hodnotu). Např. objekt Novák typu HRÁČ popsaný atributy odrážejícími situaci v sportovním klubu může být týž objekt jako Novák typu ZAMĚSTNANEC zaměstnaný u dané firmy. Z toho plyne i sdílení objektů. Dva různé objekty mohou sdílet stejný objekt – komponentu. Mohou existovat dva různé objekty se stejnou hodnotou.

Pomocí identity objektů lze snadno řešit problém referenční integrity známé z relačních databází. To, co se např. v SQL musí zapisovat explicitně (viz konstrukty REFERENCES či FOREIGN KEY v definici tabulek), není v OOSŘBD nutné, protože pomocí OID jsou objekty automaticky fyzicky „provázány" v souladu s vazbami specifikovanými definicích objektů. V relační databázi je vazba daná referenční integritou pouze logická.

## 2.2 Třídy a typy

**Typ** v OOSŘBD sumarizuje společnou strukturu množiny objektů se stejnými charakteristikami. Typ koresponduje pojmu abstraktní datový typ používanému v informatice ještě před formulací OOSŘBD. Kromě struktury jsou zadány operace, které lze s objekty daného typu provádět. Pojem **třídy** je něco více než pojem typu a obvykle je v různých OOSŘBD pojímán různě. Zahrnuje v sobě možnost produkovat nové objekty (objekt factory), tj. nějakou formu operace **new**, dále pak možnost sdružovat všechny potřebné objekty (**instance třídy**) pohromadě v tzv. **kontejneru objektů** (object container). Kontejner může uživatel přistupovat pomocí daných operací. Množina všech (právě daných) objektů dané třídy se také nazývá **extenzí třídy**. Typ je tedy obecně vztažen spíše ke konceptuálnímu pohledu, třída je použita při implementaci typu.

Z uvedeného plynou odlišnosti, které mohou nastat při použití typu a třídy. Je-li spojen typ vždy s třídou, nelze např. jednoduše hovořit o hodnotách atributu typu ADRESA pro

např. objekt typu ZAMĚSTNANEC. Je třeba vytvořit třídu ADRESA, ve které existují adresy nezávisle na zaměstnancích. V OOSŘBD $O_2$ [BCD89] lze definovat třídy **with extension** (korespondují pojmu třída) a **without extension** (korespondují pojmu typ). V prvním případě se každý nový objekt automatický stane persistentní, v druhém případě jen tehdy, je-li tento objekt explicitně pojmenován. Hodnoty typu (jak již víme) nemusí mít OID. Lze tedy používat objekty, jejichž stav je tvořen opět z objektů, ale i z hodnot.

*Příklad 1*: V OOSŘBD ORION slouží třída nejen k popisu struktury a chování, ale i jako kontejner objektů.

**make-class** <jméno-třídy>
**:superclass** <seznam-nadtříd>
**:attributes** <seznam-atributů>
**:methods** <seznam-specifikací-metod>

Na rozdíl od např. OOSŘBD GemStone, jsou atributy staticky typované.

GemStone vůbec nerozlišuje mezi třídami a typy. Třída slouží k popisu společného chování objektů třídy a není automaticky instalován kontejner objektů. Také IRIS používá pouze pojem typ, a to pro pojmenovanou kolekci objektů, jejíž členové mají společné vlastnosti.

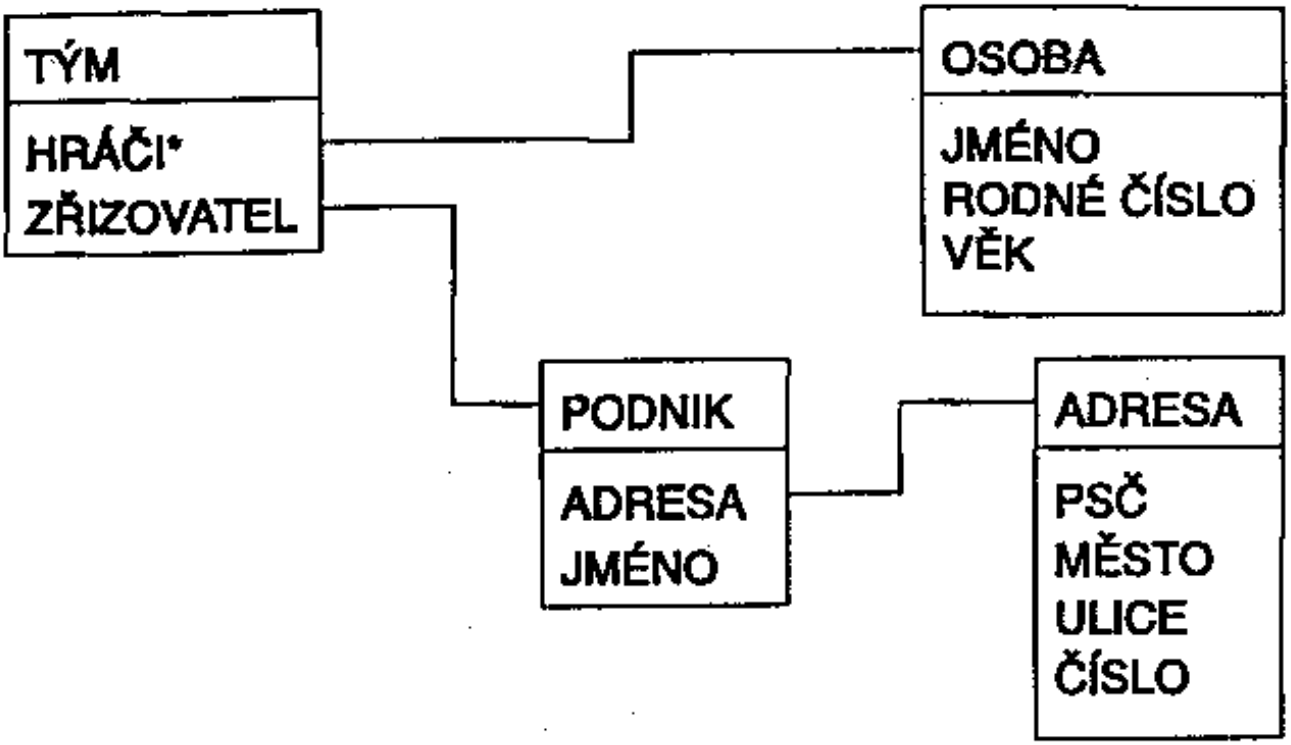
Oba příklady zastupují dvě kategorie OOSŘBD – s typovanými třídami a s netypovanými třídami. V prvním případě jde o tzv. **statické typování** (lze je kontrolovat v čase kompilace), v druhém případě jde o **dynamické typování** (v průběhu provádění programu).

Na třídu se lze totiž dívat opět jako na objekt, tj. lze vytvořit třídu tříd – **metatřídu** (pojem často používaný v umělé inteligenci). K popisu metatříd slouží **atributy tříd**. Např. počet prvků třídy by mohl být takovým atributem. Uvedené atributy a metody instance tříd metatřídy nedědí. Většina OOSŘBD však metatřídy nepoužívá. V ORIONu existuje systémem definovaná třída CLASS tvořící jak třídu všech tříd, tak kořen hierarchie všech tříd, tj. je nadtřídou každé definované třídy.

OO databázové schéma může být zadáno např. grafickým způsobem (popisuje třídy objektů).

*Příklad 2:* Schéma objektů na obrázku 1 ukazuje strukturovaný pohled na prostředí sportovních týmů.

Obdélníky označují třídy, tj. i adresy jsou samostatné objekty. * označuje množinu (lze s ní tedy vyjádřit vícehodnotový atribut). Seznam metod je u každé třídy vynechán.




*Obr. 1*

### 2.3 Zapouzdření

Jde o princip, že objekty dané třídy (či kontejner třídy) mohou být přistupovány pouze přes dané rozhraní a ne jiným způsobem, tj. hodnoty atributů objektů nejsou obecně přístupné z programovacího jazyka. Uživatel tedy nemá jinou možnost, než používat nabídnuté operace, což vede k plně zabezpečené ochraně dat.

Objekt má rozhraní a implementaci. Rozhraní je dáno specifikací operací, které mohou být prováděny na objektu. Operace tvoří jedinou viditelnou část objektu. Implementace má část datovou a procedurní, které slouží k reprezentaci stavu objektu a implementaci každé operace. Operace s v OOSŘBD obvykle nazývají **metody** (methods). Použití metod se děje prostřednictvím **zpráv** (messages), daných např. možností volání funkcí či příkazů konkrétního jazyka. Místo o vyvolávání procedury se hovoří o **posílání zpráv**. Na rozdíl od volání procedur v běžných programech, zde záleží na příjemci zprávy, tj. aktuální kód zprávy je znám až při provádění OO programu. Metody mohou být spojeny jen s jednotlivým objektem, častěji však s třídou objektů, existují i přístupy s metodami pro třídy v metatřídě.

Architektury aplikace pod klasickým SŘBD a OOSŘBD se výrazně liší, jak ukazuje obr. 2.

Zapouzdření dovoluje měnit programátorovi datovou část, aniž by se změnily programy pracující s odpovídajícími objekty. Implementačními jazyky jsou např. C (používá ho Vbase a $O_2$), Lisp (v ORIONu) či varianty SMALLTALKu (např. jazyk OPAL v GemSto-


*Obr. 2*

ne), či C++ (ONTOS, ObjectStore, Objectivity/DB) dnes *de facto* standardní prostředek pro vývoj software.

Metody mohou být redefinovány pro různé typy, to znamená, že jméno metody může znamenat různé věci při aplikaci na různé objekty. Tomuto jevu se říká **přetížení** (overloading) a vede k tomu, že připojení procedury ke jménu metody nastává až v době provádění programu. Jde o tzv. **opožděné vazbení** (late binding).

V $O_2$ se připouštějí i výjimky pro chování objektů dané třídy. Instance třídy může mít ještě další atributy a metody. Přídavné metody charakterizují vyjímečné chování instance.

### 2.4 Polymorfismus

**Polymorfismus** je schopnost operací fungovat na objektech více než jednoho typu (nebo patřících do více než jedné třídy). Např. metoda realizující třídění může být implementována jednou procedurou jak pro atribut JMÉNO–ZAMĚSTNANCE, tak pro atribut ADRESA–ZAMĚSTNANCE. Funkce COUNT může počítat mohutnost množin, které jsou podmnožinami množin různých typů.

Při přetížení se stejnému jménu funkce přiřazuje různá procedura. Jindy se používají tzv. **koercitivní operace** pro konverzi argumentů tak, aby mohl být využit daný kus kódu. Je–li např. sčítání definováno na číslech **real**, dochází při argumentech **integer** a **real** ke kolizi. Při koerci dojde ke konverzi argumentu typu **integer**, při přetížení se zvolí operace připouštějící smíšené argumenty.

### 2.5 Dědění

OOSŘBD dovolují uživateli odvozovat z existujících tříd nové třídy. Nová třída – **podtřída** (subclass) dané třídy nebo několika daných tříd – **dědí** (inherits) všechny atributy a metody existující třídy zvané **nadtřída** (superclass) nové třídy. Uživatel může specifikovat pro podtřídu přídavné atributy a metody. Třída může mít libovolné množství podtříd. Někdy v OOSŘBD existuje omezení, že třída může mít jen jednu nadtřídu. V opačném případě hovoříme o **vícenásobném dědění** (multiple inheritence). Systémy, které neumožňují vícenásobné dědění podporují vytváření hierarchií tříd.

Dědění může přinášet konflikty v použití jmen atributů i metod. Je–li použito stejné jméno atributu či metody pro třídu, jaké se vyskytuje v její nadtřídě, dává se přednost výskytu v třídě a atribut ani metoda se z nadtřídy nedědí. Podobně může nastat konflikt při stejných jménech u vícenásobného dědění.

Přestože jsou s vícenásobným děděním problémy, OOSŘBD přijímají tento rys pro pohodlnější modelování objektů. Také jazyky C++ a současné verze SMALLTALKu jej zahrnují.

Je vhodné předpokládat, že objekt je instancí právě jedné třídy, a to té na nejnižší úrovni. Objekt ale může být členem několika tříd. Být členem třídy C znamená být instancí C nebo její nějaké podtřídy. Tak je tomu např. u ORIONu, ne však u IRISu, kde všechny instance podtypu jsou i instancemi nadtypu.

### 2.6 Složité objekty

Každý, kdo používá SŘBD založený na RMD, dříve nebo později odhalí jeho charakteristickou slabinu – požadavek na 1NF relací. Odstraníme–li předpoklad 1NF, lze konstruovat datové struktury, kde prvkem řádku tabulky může být množina, n–tice, seznam, relace apod. Využít lze i funkce. Hnízděným použitím těchto konstruktů lze získat velmi složité datové struktury vhodné pro modelování takových objektů, jako jsou formuláře, přístroje, obvody, mapy, obrazy apod. Inspirací pro tento přístup jsou (ne OO!) databázové modely zvané **složité objekty** lišící se volbou základních konstruktů, ze kterých se složité objekty konstruují, ale i konstrukty sémantických databázových modelů jako je dnes např. všeobecně používaný E–R model.

### 2.7 Rozšiřitelnost

**Rozšiřitelnost** znamená možnost definovat nové základní typy a posléze nové typy pomocí konstruktů pro vytváření typů (externě definované typy). To vede k požadavku mít možnost i vytvořit efektivní implementaci takových typů. Rozšiřitelnost je spíše požadavek na architekturu než na OO přístup (vyskytuje se i v rozšířených relačních systémech). Existuje zatím málo OOSŘBD, které umožňují externě definované typy (viz např. GENESIS, EXODUS).

Pojem rozšiřitelnost se spíše používá ve funkčním smyslu, tj. jde o zahrnutí uživatelem specifikovaných operací. Problémem ovšem je, na které úrovni databázové architektury je vhodné je zahrnout: např. prostá příprava nové operace v termínech starých operací může být neefektivní.

# 3 MANIPULAČNÍ JAZYKY V OOSŘBD

Databázové jazyky obvykle nejsou výpočetně úplné, tj. nelze s nimi vypočítat libovolnou vyčíslitelnou funkci, jako tomu je u programovacích jazyků. Problém síly databázového jazyka se obvykle obvykle řeší (podobně jako u SQL) začleněním do programovacího jazyka nebo 4GL jazyka, což zase vede k tomu, že se získá strukturálně i sémanticky nehomogenní prostředek. (Tomuto problému s v angl. říká impedance mismatch.) Cílem OO projektů je vytvořit výpočetně úplný OOSŘBD, a to tak, aby jeho databázový jazyk byl v konstruktech dostatečně homogenní.

Složitá struktura objektů vnořených do sebe svádí k představě navigačního prostředí pro manipulaci objektů. Poměrně jednoduché řešení může právě být rozšíření některého OO orientovaného programovacího jazyka o databázové konstrukty. Např. ORION takto využívá Common LISP, GemStone rozšiřuje SMALLTALK, ObjectStore či Ontos využívají C++. problémem je, že uvedené jazyky jsou natolik obecné, že nejsou příliš vhodné pro tvorbu databázových aplikací. Vyvíjejí se i nadstavby C++ (jako např. O++) řešících tento problém.

Na druhé straně se pro pohodlí koncových uživatelů vytvářejí jazyky, které jsou deklarativní, tj. připomínají neprocedurální relační jazyky. Tyto jazyky se v mnohém liší od známých relačních jazyků. Tak např. díky existenci OID je třeba rozlišit více typů rovnosti: rovnost hodnot a rovnost identit. Vzhledem k použití složitých objektů je možné vytvářet výrazy popisující cestu „strukturou" objektu k nějaké jeho části a tu porovnávat s jiným objektem.

*Příklad 3:* Uvažujme následující složitý objekt (syntaxe vychází ze systému $O_2$):

```
add class TÝM
      type tuple (HRÁČI: set(OSOBA)
                  ZŘIZOVATEL: tuple (ADRESA:tuple(PSČ:string
                                                  MĚSTO:string
                                                  ULICE:string
                                                  ČÍSLO:string)
                                     JMÉNO: string ))
      with extension
add class OSOBA
      type tuple (JMÉNO: string
                  RODNÉ-ČÍSLO: string
                  VĚK: number
```

TÝM má atribut HRÁČI (tj. množina objektů typu OSOBA), atribut ZŘIZOVATEL daný jménem (JMÉNO) a adresou (ADRESA) strukturovanou jako <PSČ,MĚSTO,ULICE,ČÍSLO>. Je-li proměnná x typu TÝM, pak porovnání

x.ZŘIZOVATEL.ADRESA.MĚSTO = Praha

dává test na město zřizovatele daného týmu (obsaženého v proměnné x). Porovnání lze přijmout jako logickou formuli eventuálního dotazovacího jazyka.

Této syntaxe lze využít i v OO jazycích založených např. na populárním SQL. Vytvořme nejprve pojmenovaný objekt Podskaláci pomocí příkazu

**add object** Podskaláci: TÝM

Pak výraz

```
select x.RODNÉ–ČÍSLO
from x in Podskaláci.OSOBY
where x.VĚK > 30
```

označuje dotaz „Najdi rodná čísla hráčů z týmu Podskaláci starších 30 let."

Jiný způsob přístupu k objektům používají OOSŘBD založené na konstruktu funkce. Zahnízděným voláním funkcí lze získat potřebný objekt.

*Příklad 4:* Předpokládejme třídy ZAMĚSTNANEC, jejíž podtřídou je VEDOUCÍ, dále třídu SEKRETARIÁT s atributem ADRESA. SEKRETARIÁT je atributem objektů třídy VEDOUCÍ. ADRESA má jeden z atributů ULICE. ZAMĚSTNANEC má atribut PLAT. Chceme vyjádřit dotaz „Najdi ulice sekretariátů všech vedoucích těch zaměstnanců, jejichž plat je větší než 10000 Kčs". V jazyku Object SQL (další varianta objektově orientovaného SQL) v OOSŘBD IRIS lze dotaz vyjádřit jako

```
select ULICE(ADRESA(SEKRETARIÁT(VEDOUCÍ(z))))
foreach ZAMĚSTNANEC z
where PLAT(z) > 10000
```

V normalizovaných relacích je obvykle nutné používat nějakou formu explicitního spojení relací (totéž se vyskytuje i v standardním SQL). V OOSŘBD je tato potřeba dost eliminována právě hnízděnou strukturou složitých objektů, kde implicitní spojení vlastně již existují a stačí pouze definovat potřebnou cestu. Také ne všechny OO dotazovací jazyky připouštějí explicitní spojení (bez cesty), tj. porovnávání dvou objektů na identitu či rovnost hodnot. Zřejmě se tím eliminují jisté neužitečné dotazy vedoucí k vytváření kartézských součinů objektů.

Dalším zajímavým prvkem je možnost obdržet s dotazem na třídu i prvky jejích podtříd. Např. v ORIONu lze zadáním jména třídy ZAMĚSTNANEC požadovat pouze prvky této třídy, zadáním ZAMĚSTNANEC* jsou uvažovány i prvky jejích podtříd.

V [BM91] jsou uvedeny některé základní otázky týkající se OO dotazovacího jazyka:

*(1) Může dotazovací jazyk porušit zapouzdření?*

Rozumnou odpovědí je zřejmě ANO pro ad hoc dotazovací jazyky a NE pro dotazovací jazyky zahrnuté v hostitelském jazyku. Znamená to, že objekty mohou být uvažovány jako hodnoty a tedy je možné se dotazovat na jejich strukturu. Jinak by objekt byl přístupný pouze pomocí metod a hodnoty dat by byly neviditelné.

*(2) Na co se dotazovat? Na data, metody, nebo na obojí?*

Jde o to, jak využít v dotazovacím jazyku i metody, a nejen „fixované" konstrukty jazyka. Např. v definici třídy OSOBA by mohl být zadán atribut DATUM–NAROZENÍ a VĚK by mohla být metoda počítající věk osoby z data narození a současného data. Pak je užitečné využít metodu VĚK přímo v dotazovacím jazyku.

Toto řešení by vlastně dovolovalo použít v dotazovacím jazyku libovolně složitou metodu a tím jej i náležitě zesložitit pro uživatele, což poněkud odporuje představě o uživatelském dotazovacím jazyku. Řízení použití metod lze ovlivnit pomocí deklarací metod jako privátních nebo veřejných, kde privátní jsou viditelné z datového typu a veřejné realizují uživatelské rozhraní.

*(3) Co je odpověď na dotaz?*

Výsledkem dotazu může být nejen množina objektů dané třídy, tj. odpověď může tvořit podtřídu třídy, ale i nové objekty, ke kterým je třeba zkonstruovat novou třídu spolu s množinou metod. Toto řešení je zřejmě komplikované. Jiným extrémem je řešení (použité např. v IRISu) kde výsledkem dotazu jsou pouze hodnoty nebo množiny hodnot, tj. OOSŘBD v sobě zahrnuje klasické relace. Tento přístup generuje hodnoty.

Další a častější možností je generování objektů omezeným způsobem. Výsledek dotazu sestává z nových objektů vytvářejících třídu následující přímo za kořenem hierarchie. Třídy–odpovědi mohou také tvořit metatřídu, avšak těžko použitelnou, protože její prvky nemají homogenní strukturu a je tedy těžké znovu využít výsledky dotazů pro další zpracování. Jedinými metodami mohou být ty, které zobrazují nebo tisknou výsledek.

*(4) Měl by dotazovací jazyk být integrován v programovacím jazyku?*

Odpověď ANO zcela vyhovuje záměrům OOSŘBD. Integrovaný databázový jazyk obsahující rysy dotazování a mající sílu programovacího jazyka řeší „impedance mismach" relačních jazyků, uživatel není nucen se učit dva různé jazyky.

*(5) Jaký by měl být vztah k typovému systému?*

Pro formulaci dotazů není nutné používat explicitně informaci o typu použitých struktur. Veškeré takové informace by si měl OOSŘBD odvodit v čase provádění dotazu. Odvození typu v čase kompilace je nutné tehdy, počítá-li se s optimalizací dotazu. Přesně tak se chová dotazovací jazyk $O_2$Query.

## 4 Současný stav a výhled OOSŘBD

Předchozí diskuse o OO přístupu naznačuje, že jen obtížně lze dospět k jeho standardizaci. V různorodosti existujících přístupů lze vždy nalézt možná pro a proti, nicméně každý z nich může vést k funkčnímu systému.

Srovnání komerčních produktů je dost obtížné. Většina starších OOSŘBD (např. GemStone) má blíže k základní definici, ORION již podporuje verze a rozšířený model transakcí. Protože není dost zkušeností s vývojem OO aplikací, každý krok je učením (pro tvůrce aplikací i výrobce OOSŘBD). Zkušenosti nejsou zatím s velkými objemy dat a ani z aktivními OO databázemi (četné a časté souběžné transakce).

OOSŘBD jsou dnes v r. 1993 relativně ve stejném stavu jako relační produkty v r. 1985 – 6 let po zavedení konvenčních relačních produktů ORACLE a INGRES. Vzestup relační technologie také započal až po zjištění skutečných přínosů relačních systémů. Zdá se, že podobně je tomu i u OOSŘBD.

Masovější nástup samotných OOSŘBD se očekává v polovině 90.let. Vzhledem k jejich složitosti nelze asi hned očekávat velkou efektivnost těchto systémů. Vzpomeňme však na RMD a 70. léta, která vlastně byla celá investována do vývoje relačních systémů.

Problémy OOSŘBD však nesouvisí jen s implementací. Otevřené výzkumné otázky zůstávají i nadále v optimalizaci dotazů, dynamice schématu OO databáze, schází i dostatečně uspokojivé formální modely, vyvíjejí se metody návrhu OO aplikací. Pro srovnání OOSŘBD zatím nejsou k dispozici vhodné srovnávací testy (benchmarks).

Za zmínku stojí i chování některých tradičně relačních výrobců. Např. ORACLE Corp. ohlásila, že verze 8 jejich relačního produktu bude objektově orientovaná. Existují také pokusy standardizovat OO přístup. Pracovní skupina ANSI X3H7 vyvinula referenční model pro řízení objektových dat, skupina ANSI X3H2 vyvíjí objektové rozšíření SQL v rámci standardu SQL3.

## Literatura


[Atk89] Atkinson, M., et al:
The Object–Oriented Database Systems Manifesto. Proc. First Int. Conf. Deductive and Object–Oriented Databases. Elesevier Sc.Pub. B.V. Amsterdam, 1989.

[Ban88] Bancilhon,F.: Object–oriented database systems. Proc. of Symp. on PODS, Austin, Texas, 1988.

[BCD89] Bancilhon,F., Cluet,S., Delobel,C.:
A Query Language for the $O_2$ Object–Oriented Database System. RT Altair 35–89, 1989.

[BM91] Bertino, E., Martino,L.:
Object–Oriented Database Management Systems: Concepts and Issues. Computer, April 1991.

[CADF90] Committee for Advanced DBMS Function: Third – Generation Data Base System Manifesto. Memorandum No.UCN/ERLM90/28, University of California, Berkeley, 1990.

[Com91] Communication of the ACM, Vol.34, No.10, 1991.

[Eng92] English,L.P.: Object Databases at Work. DBMS, 5,11,1992.

[Po92] Pokorný,J.: Databázové systémy a jejich použití v informačních systémech. ACADEMIA, 1992.

[US91] Unland, R., Schlageter,G.:
Object–Oriented Database Systems: State of the Art and Research Problems (rukopis), 1991.


---


**Autor:** RNDr. Jaroslav Pokorný, CSc.
katedra softwarového inženýrství
MFF UK
118 00 Praha 1
tel. 532136