OODBMS - systém POET, jeho struktura, možnosti a způsob využití

P. Matula

CIT Ostravské univerzity, Bráfova 5, 701 03 Ostrava 1, Česká republika
e-mail: matula@adam.osu.cz

Abstrakt

Materiál poskytuje stručný přehled základních obecných koncepcí, o které se opírají současné objektově orientované databázové systémy (OODBMS). Obecné koncepce jsou podrobněji diskutovány na příkladu komerčně úspěšného OODBMS POET, který byl vybrán jako modelový databázový systém na platformě Windows. Důraz je kladen na datové struktury databázových schémat (objekt, třída, kontejner, vztahy mezi objekty, závislé objekty) a mechanismus referenční integrity. Relativně samostatnou kapitolou je OQL (Object Query Language) standard ODMG (Object Database Management Group), který je využíván pro dotazy nad objektovou databází systému POET. Vedle jazyka C++ je možno pro tvorbu databázového schématu využít i další ODMG standard ODL (Object Definition Language), jehož hlavní konstrukty budou zkratkovitě popsány. Databázové operace a obecně manipulace s daty se děje prostřednictvím volání metod databázových tříd.

1. OBJEKTOVĚ ORIENTOVANÝ DATABÁZOVÝ SYSTÉM

V 90. letech se staly OO programovací jazyky standardem při vývoji aplikací. Relační struktura databáze je však diametrálně odlišná od objektové struktury aplikací a proto je nutné používat celou řadu umělých kroků podporujících spolupráci objektové aplikace s relační databází, jako je například modelování databáze v aplikačních třídách nebo naopak modelování aplikace v databázi [1]. Není třeba zdůrazňovat, že takovéto kroky komplikují strukturu výsledné aplikace, přispívají k horší výkonnosti takovéto aplikace a zvyšují riziko vzniku chyb a tedy i celkovou dobu potřebnou pro vývoj a odlazení databázové aplikace.

Z tohoto a jiných důvodů dochází již dnes k rozvoji OODBMS ve specifických oblastech nasazení IS. Nástup OODBMS však v žádném případě neznamená pomalý zánik RDBMS. Relační databáze jsou vhodné pro správu velkých objemů relativně jednoduchých dat, OODBMS dobře vystihují složité vztahy mezi daty [2]. RDBMS poskytují dobré prostředky pro selekci dat, kdežto manipulace s daty je komplikovaná. OODBMS umožňují flexibilní manipulaci s daty, kdežto dotazovací možnosti jsou pod úrovní standardního SQL.

Databázové schéma v prostředí systému POET je možno vytvořit pomocí C++ (používáme-li C++ SDK) nebo ODL. Standardní C++ deklarace třídy je rozšířena o několik klíčových slov a proto bývá soubor databázového schématu specifikován s příponou HCD.

POET je pasivní OODBMS a tedy ve třídním slovníku jsou uloženy pouze deklarace metod databázových tříd.

1.1 Architektura systému POET

OODBMS POET implementuje klient-server architekturu objektového serveru. Jednotkou přenosu mezi klientem a serverem je objekt nebo skupina objektů. Hlavní výhodou této architektury je zvýšení stupně konkurenčního přístupu k datům a snížení síťové zátěže. Na druhé straně architektura objektového serveru přináší komplikovanější management zámků.

V OODBMS funguje lokální cache klienta jako rozšířená vyrovnávací paměť serveru a část funkcionality serveru je tak přenášena přímo na klienta. Jedná se zejména o předzpracování dotazů, pomocné procedury při obnově dat, prohledávání indexů ap. Z těchto a jiných důvodů jsou OODBMS vhodným nástrojem pro tvorbu tzv. multi-tiered aplikací.

V současnosti se stále více prosazuje řešení webovského serveru, za nímž v pozadí stojí OODBMS. Webovský server pro ukládání persistentních dat využívá OODBMS a zároveň obsluhuje vlastní webovské klienty. Podobně jsou OODBMS využívány v CASE a systémech pro podporu rozhodování [3].

Objektový systém POET může běžet v rámci jednoho procesu spolu s klientskou aplikací nebo jako samostatný serverovský proces. Řešení v rámci jednoho procesu (aplikace + server) je vhodné pro aplikace, které vyžadují lokální persistentní uložení nebo pro doménově specifické serverovské systémy vyžadující vysoký výkon - CD-ROM aplikace, WWW servery a aplikace v prostředí CORBA. Tím, že POET a aplikace tvoří jediný proces, je eliminována komunikační zátěž.

1.2 Objektová identita

Relační databáze pro identifikaci daného řádku tabulky využívají koncept primárního klíče, který je založen na datech uložených v identifikovaném řádku. Identifikátor objektu (OID) není odvozován od atributů objektu ani od paměťové adresy (tak jako v C++), ale je systémově generován ve chvíli, kdy je persistentní objekt zapsán do databáze. OID je neměnný, unikátní a nerecyklovatelný. S využitím OID lze jednoduše konstruovat vztahy mezi objekty.

POET rovněž provádí tzv. *swizzling*, což je mapování meziobjektových vztahů z jejich diskové formy (reprezentované skrze OID tj. logický identifikátor) na pointery virtuální paměti v okamžiku, kdy je objekt natažen do paměti klienta. Je-li objekt natažen do paměti, poskytuje POET pro přístup a reference mezi objekty kombinaci fyzických adres (pro zrychlení dereference objektových pointerů v paměti) a logických identifikátorů. Při specifikaci schématu databáze je možno vztahy mezi

objekty popsat klasickým pointerem (využije se *swizzling*) nebo *onDemand* pointerem (po natažení do paměti bude zachována reprezentace meziobjektových vztahů pomocí OID). Jestliže je objekt natažen do paměti, pak všechny objekty referencované pomocí *non-onDemand* pointerů jsou rovněž nataženy do paměti. Je-li některý objekt referencován pomocí *onDemand* pointeru, pak není natažen do paměti v okamžiku, kdy je natažen jeho mateřský objekt, ale až ve chvíli, kdy je explicitně dereferencován [4]. Při ukládání objektů na disk je třeba provést inverzní operaci ke *swizzlingu* a to převod fyzických identifikátorů (paměťové pointery) na logické.

1.3 Základní konstrukty pro databázové schéma

```
class address               // non-persistentní třída
  {//je možno uvést atributy, konstruktory, destruktory, metody (i virtuální) ap.
   };
persistent class person      //persistentní třída
  { address Domicile; //vložený objekt, nemá OID, nemůže existovat bez objektu třídy
                      //person, je ukládán v rámci svého mateřského objektu        (1)
   public: void SetAddress(address* pAddress); //metoda pro update private atributu
    //...
   };
persistent class employee: public person //vztah dědičnosti person --> employee
 { public:department* office;  //vazba 1:1,employee-->department                   (2)
    transient dialog* pDialog; //transientní člen, který nebude uložen v databázi
   //...
  };
persistent class manager: public employee
 {public:  cset <ondemand<employee>> staff; //vazba 1:N, manager-->employee   (3)
   //...
  };
persistent class department
  {public: cset<employee*> members;  //vazba 1:N, department-->employee        (4)
    //...
   };
```

1.3.1 Kontejnery

POET disponuje množinou speciálních kontejnerů (cset, lset, hset), které jsou typicky využívány pro tvorbu 1:N vztahů a pro specifikaci vícehodnotových atributů. Kontejnery umožňují tvorbu složitých objektů nebo vztahů bez nutnosti provádět spojení nebo specifikaci cizích klíčů. Příklad využití kontejneru a o*nDemand* pointeru pro specifikaci 1:N vztahu je na značce (3) podkapitoly 1.3.

1.3.2 Vztahy

V rámci POET objektového modelu lze vztahy specifikovat zejména prostřednictvím:

- pointeru na objekt - viz značka (2) podkapitoly 1.3
- *onDemand* pointeru - viz značka (3) podkapitoly 1.3

- vloženého objektu - viz značka (1) podkapitoly 1.3

Vlastnosti *onDemand* pointerů lze rovněž využít při specifikaci *onDemand* množiny. Je-li množina klasickou množinou pointerů, pak všechny referencované objekty množiny jsou načteny do paměti v okamžiku, kdy je načtena samotná množina. Obsahuje-li množina *onDemand* pointery, pak objekty množiny jsou nataženy do paměti až v době, kdy jsou explicitně vyžadovány (viz (3) v 1.3).

1.4 Referenční integrita

Jeden z hlavních aspektů referenční integrity by měl zajistit, aby při vymazání objektu byly odstraněny i všechny reference na tento objekt. Jelikož POET neobsahuje žádný deklarativní jazyk, kterým by mohlo být nadefinováno takovéto referenční omezení, podporuje POET zajišťování referenční integrity na aplikační úrovni prostřednictvím metod. S každým objektem v paměti si POET udržuje speciální čítač linků (tzv. *link count*), který určuje aktuální počet aktivních (ne tedy pouze třídně deklarovaných) referencí na daný objekt. Dále POET obsahuje tzv. bezpečné funkce pro mazání, které umožní vymazat daný objekt pouze tehdy, jestliže je čítač linků asociovaný s daným objektem roven nule neboli objekt není cílem ukazatele žádného jiného objektu. Tímto mechanismem je eliminována možnost vzniku volně pověšených referencí, které mohou mít za následek chybu v provádění aplikačního programu.

POET také poskytuje mechanismus, který zajišťuje, aby v případě potřeby při vymazání objektu, který je složen z několika dalších objektů, byly vymazány i tyto dílčí objekty.

K dodržování referenční integrity přispívá také konstrukt závislých objektů. Závislý objekt nemůže existovat bez objektu, který jej vlastní. Je-li vlastník objektu vymazán, je vymazán i závislý objekt. Pro specifikaci tohoto konceptu používáme klíčového slova *depend* před deklarací vztahu.

Závislý objekt může být referencován pouze z jednoho mateřského objektu. Máme-li v deklaraci třídy pouze jeden pointer (nikoliv množinu pointerů) na závislý objekt, pak je výhodnější tento objekt vložit přímo do deklarace dané třídy. Následující příklad ukazuje deklaraci *onDemand* množiny závislých objektů:

```
depend lset <ondemand<Entry>> Entries;
```

1.5 Práce s databází na aplikační úrovni

V prostředí POET Developeru vytvoříme a zkompilujeme HCD soubor popisující databázové schéma. Výsledkem kompilace je vytvoření několika pomocných souborů pro integraci databáze a aplikace, které se vloží do projektového souboru aplikace. Vývojové prostředí lze upravit tak, aby kompilace databázového schématu probíhala společně s kompilací aplikace. Samotná práce s databází se děje prostřednictvím volání POET API funkcí tak, jak ilustruje několik následujících jednoduchých příkladů[1].

[1] Pro jednoduchost v uvedených příkladech neprovádíme kontroly úspěšnosti provedených operací.

Uložení persistentního objektu třídy *person* je možno provést následujícím fragmentem kódu (základní databázové metody (store, delete ap.) dědí každý persistentní objekt):

```
PtBase* pBase;
person* pPerson = new person(“George“, “Washington“);
address* pAddress = new address(“Janackova 8“,“Opava“,“CZ“,“74700“,“659002“);
pPerson->SetAddress(pAddress);
pPerson->Assign(pBase);    // asociace objektu a databáze
pPerson->Store();    // uložení objektu třídy person  v databázi
delete pPerson; delete pAddress; //uvolnění paměti po uložení objektu do databáze
```

Vymazání objektu z databáze lze provést například takto (vymažeme všechny objekty třídy *person*):

```
PersonAllSet allPersons(pBase);  //PersonAllSet je třída vytvořená kompilátorem
                //databázových schémat,  obsahuje  všechny objekty příslušné třídy
int numberOfPersons = allPersons.GetNum(); //počet objektů třídy person
person* pPerson = (person*) 0;
for (int i = numberOfPersons -1; i >= 0; --i)
  {  allPersons.Get(pPerson, i, PtSTART); // i určuje pořadové číslo objektu množiny,
                                //který bude načítán do pPerson relativně k počátku množiny
     pPerson->Delete(); // smazání objektu z databáze
     allPersons.Unget(pPerson); //uvolnění paměti alokované pro objekt
   }
```

Pro aktualizaci objektu je třeba objekt načíst do paměti, změnit jej a metodou store provést zpětné uložení do databáze. POET zjistí, že objekt s daným OID v databázi již existuje a proto místo vytvoření nového objektu provede aktualizaci existujícího objektu.

2. ODMG STANDARDY V SYSTÉMU POET

V systému POET je možno využít OQL a ODL, i když jejich implementace v systému není zdaleka úplná.

Pro pochopení základních konstruktů OQL si rozebereme jednoduchý dotaz, který vybere oddělení, na kterém pracuje zaměstnanec s příjmením Zach (použijeme kostru databázového schématu z podkapitoly 1.3).

```
define extent Depts for department;
select e.Dept_name from e in Depts, d in e.members   where d.Last_name = “Zach“;
```

Nejdříve je třeba definovat extent, tj. množinu objektů dané třídy, na které bude probíhat dotaz. Poté proběhne výběr objektů z extentu, které splňují podmínku.

Dotaz tohoto typu můžeme specifikovat ještě druhým způsobem. Dotaz, který využívá navigaci mezi objekty ve směru „objekt třídy employee“ --> „objekt třídy department“ zapíšeme v OQL takto:

```
define extent Emps for employee;
select e.office.Dept_Name from e in Emps  where e.Last_Name=“Zach“;
```

ODL je standard, který je využíván pro tvorbu databázového schématu. Pomocí ODL je možno definovat třídu, její atributy, metody, vztahy mezi třídami, dědičnost, kolekce a konstanty.

```
interface employee : person (extent employees) : persistent        (1)
{ const float coef = 11.25;                                        (2)
  attribute string Work_phone;                                     (3)
  relationship department office inverse department::members;      (4)   // vztah 1:1
  relationship manager chief inverse manager::staff;               (5)   // vztah 1:1
 };
 interface manager : employee (extent managers) : persistent       (6)
 { relationship list<employee> staff inverse employee::chief;      (7)   // vztah 1:N
   void Print();                                                   (8)
  };
```

V předcházejícím příkladu (velice zjednodušeno) jsou deklarovány 2 persistentní třídy ve vztahu dědičnosti a je mezi nimi zaveden inverzní vztah (obousměrný vztah). Pro specifikaci vztahu 1:N je použita kolekce. Kromě atributů je na řádku 8 deklarována metoda třídy.

LITERATURA:

[1] POET Software: Relational Databases & object oriented Languages - www.poet.com

[2] Matula, P.: Některé aspekty přechodu od relačních databází k objektově orientovaným se zaměřením na Borland Delphi, Diplomová práce, Přírodovědecká fakulta Ostravské univerzity, 1998

[3] Apostrophe Software Home Page - www.aposoft.com

[4] POET Software Inc.: POET C++ SDK Programmer’s Guide, 1997