```
persistent class Person {
        PtString Name;              /* PtString je datový typ POETu pro řetězce proměnné
    délky */
};
```

Pak prekompilátor vygeneruje následující třídu:

```
class PersonQuery: public PtQuery {
 public:
        SetName(PtString& param, PtCmpOp co = PtEQ);
};
```

Nyní můžeme použít tuto dotazovací třídu pro vyhledání všech lidí, jejichž jméno začíná písmenem "A". Potřebujeme *PersonAllSet*, protože potřebujeme prohlédnout všechny lidi a také proto, že potřebujeme množinu pro obsažení výsledků.

```
typedef lset<Person*> PersonSet;

PersonAllSet* all = new PersonAllSet(pb);
PersonSet* HusbandsOfEsthers(pb);
PersonQuery husband, wife;

wife.SetName(PtString('Esther') , PtEQ);      /* manželky se jménem Esther */
husband.SetSpouse(&wife);                /* manželé jejichž manželky se jmenují Esther */
all->Query(&husband,HusbandsOfEsthers);
```

3.4. Množiny

Předpokládejme, že chceme, aby každá instance třídy *Person* obsahovala odkazy na své děti. Protože *Person* může mít i více dětí, jeden odkaz nestačí. Pole také není řešením, protože předem nemůžeme určit maximální počet dětí. POET nabízí řešení v podobě nového datového typu zvaného *set*. Deklarace :

```
cset<Person*> people;
cset<int> integers;
```

Množin je několik typů ,*cset* je compact set - jsme omezeni velikostí segmetu (na PC), mimo compact set existují i další - large a huge.

Použijme nyní cset v naší třídě person :

```
persistent class Person {
public:
        cset<Person*> children;
}
```

Nyní můžeme děti do množiny přidávat i z ní číst:

```
Adam.children.Append(&Cain);     /* Cain je dítě 0 */
Adam.children.Append(&Abel);     /* Abel je dítě 1 */

Person* second_child;
Adam.children.Get(second_child, 1L, PtSTART);
```

3.5. Inicializace, otevření a uzavření databáze

Předpokládejme, že chceme napsat aplikaci v POETu. Prvním krokem je inicializace POETu pomocí funkce *InitPoet()*, která vytváří globální objekt *PtRoot*. Tato funkce požaduje jeden volitelný parametr - jméno klienta - které může být zobrazováno ostatním procesům v síti, pracujete-li v prostředí client/server. Jakmile jste zavolali *InitPoet()*, můžete se připojit na server a otevřít databázi použitím *GetBase()*.

```
InitPoet("Client name");
PtBase* pObjBase;
int err = PtBase::POET()->GetBase("LOCAL", "base", pObjBase);
if (err < 0) {
        PtBase::POET()->UnGetBase(pObjBase);
        ErrorExit("Could not open database", err);
}
```

K ukončení se pak používá následující sekvence :

```
PtBase::POET()->UngetBase(pObjBase);
DeinitPOET();
```

3.6. Metoda STORE()

Každý persistentní objekt ví, jak sám sebe uložit, protože je odvozen z *PtObject*. Nejprve je však nutno mu sdělit, do jaké databáze se má uložit. To mu sdělí metoda *Assign()*. Pak lze teprve volat metodu *Store()*.

```
void  Application::NewPersonEntry() {

        int             select;
        Person*       pObj;
        int             err;

 pObj = new Person;

 pObj.name = 'Petr'
 pObj->Assign(this->pObjBase);

 if (err = pObj->Store())
        ErrorMessage("Store error :  ",err);

}
```

3.7. Metoda DELETE()

Persistentní objekty také vědí, jak se vymazat z databáze používajíce *PtObject::Delete()*. Vymazání objektu jej však neodstraní z RAM, k tomu slouží metoda *PtObject::Forget()*.

```
void  Application::DeletePerson() {
Person*  pObj = 0;

if (pObj = FindPerson()) {
if ( Pause ("Delete this object ? Y/N") == 'Y')
pObj->Delete();  /* výmaz  z  databáze  */

pObj->Forget(); /* odstanění  z  RAM,  jestliže  na  něj  neexistují  žádné  další  odkazy  */
}
}
```

3.8. Verze tříd

Class dictionary, který obsahuje popis schématu databáze umožňuje ukládát několik verzí pro popis každé třídy. Takovým způsobem je zajištěna kompatibilita se staršími verzemi aplikací a není nutné tyto aplikace rekompilovat při každé změně schématu. Aplikace, které používají novější verzi tříd mají přístup i k instancím tříd starší verze. Starší verze schématu jsou nutná pro přístup k datům, která byla uložena s použitím těchto schématů. Konverze databáze do poslední verze schématu je možná pomocí funkce *Versall()*.

3.9. Indexy

Používání indexů pro danou třídu se povolí při deklaraci třídy použitím atributu *useindex* se jménem příslušného indexu. Deklarace indexu je uvozena klíčovým slovem *indexdef*. Index lze vytvářet pro jeden atribut, pro skupinu atributů, u každého textového atributu lze zadat kolik znaků zleva bude platných pro vytváření indexu.

```
persistent class Person {
        char        name[30];
        Address     address;

        useindex    PersonIndex;
};

indexdef PersonIndex: Person {
        name[[10]];                    /* Index bude využívat pouze prvních 10 znaků */
}
```

Uspořádání u složených indexů probíhá odshora dolů (nejprve je tedy indexováno podle první položky indexu, pak podle druhé, atd. až podle poslední položky). Následuje deklarace pro index založený na jménu a PSČ. Tento index deklarujeme také jako *unique*, což způsobí, že metoda *Store* skončí chybou při ukládání duplicity.

```
unique indexdef PersonIndex : Person {
        name[[10]];
        address.psc;
}
```

Je možno vytvářet indexy i pro atributy referencovaných objektů. Pro každou třídu lze definovat libovolný počet indexů

3.10. Transakční zpracování

POET používá implicitní transakce na úrovni systému při všech operacích zápisu. Změny provedené v databázi jsou zaznamenávány do žurnálu, který je použit v případě zotavení z chyb. Použití žurnálu musí být explicitně povoleno nastavením *RecoveryFile = 1* v konfiguračním souboru PTSERVER.CFG.
Uživatelsky definované transakce jsou uvozeny příkazy *BeginTransaction(), CommitTransaction(), AbortTransaction(), TransactionLevel()*. Uživatelsky definované transakce lze do sebe vnořovat, potvrzení veškerých změn v databázi se uskuteční až v okamžiku, kdy je potvrzena transakce definovaná na nejvyšší úrovni. Uživatelsky definované transakce jsou realizovány různými verzemi objektů, po neúspěšném ukončení transakce lze obnovit poslední platnou verzi objektu příkazem *PtObject::Refresh*.

3.11. Zamykání objektů

Zamykání objektů v databázi je umožněno prostřednictvím třídy *PtLockSpec*. Uzamykat lze celý objekt, objekt včetně všech závislých objektů, celý objekt včetně všech referencovaných objektů,

které nejsou definovány *OnDemands*, celý objekt včetně všech referencovaných objektů. Definovaný zámek je použit jako parametr příkazu Get() pro čtení hodnoty objektu, zámek je nutno explicitně zrušit použitím zámku jako parametru příkazu *Unget()*. Uváznutí při zamykání objektů musí předcházet programátor.
Při použití příkazů *Store()* a *Delete()* uvnitř uživatelem definované transakce je nastaven implicitní zámek na daný objekt, který je systémem uvolněn až po ukončení transakce.
Zámek je možno definovat i na úrovních *Set*, *ObjectSet*, *AllSet* a *OnDemandSet* metodami *Lock* nebo *Unlock*, nebo může být použit i jako parametr příkazů *Get()* a *Query()* na příslušnou množinu objektů.

Příklad:

```
PtLockSpec  LockSpec(PtLK_WRITEvWRITE, PtFLAT);
MyAllSet.Get(&p, 0, PtSTART, &LockSpec);
p->ChangeYourself();
p->Store();
p->Unget(&p, &LockSpec);
```

3.12. Funkce vyvolané událostmi

Funkce (*CallBack Functions*), které reagují na události při práci s daty jsou součástí modulu nazvaného exception manager. Každá taková funkce může na zahájení databázové operace, její ukončení a může být opakovaně používána při zpracování dat.
Model *Watch & Notify* umožňuje reagovat na situace, kdy jsou data v databázi zpracovávána jiným uživatelem. Lze reagovat na ukládání, aktualizaci a mazání dat a na zamykání dat. Jako reakce na tyto události je vyvolán apříslušná funkce *Notify()*. K volání funkce *Notify()* dojde pouze pokud není instalována odpovídající *CallBack* funkce.

3.13. Přístupová práva

Po vytvoření je databáze všeobecně přístupná, používání přístupových práv je třeba nejdříve povolit příkazem *EnableAuthorization()*. Administrátorem databáze se pak stane uživatel POETADM, který má oprávnění přidávat a odebírat uživatele nebo skupiny uživatelů a přidělovat jim jednotlivá přístupová práva na úrovni tříd objektů nebo hodnot atributů instancí objektů. S přístupovými právy lze pracovat v grafickém prostředí správy databáze, nebo přímo z aplikace.

4. Dotazovací jazyk OQL

Deklarativní dotazovací jazyk OQL lze používat ve vývojovém prostředí POETu nebo v hostitelském prostředí C++. POET 3.0 implementuje část OQL popsaného ve standardu ODMG-93 [3].

Některá omezení:
1. ve where části dotazu není povoleno testování identity objektu
2. výsledkem dotazu je *set<X>*, *bag<X>* není možno používat
3. není implementováno *group-by, sort-by*

4. nejsou implementovány podmínky *in, exists-in, forall*
5. není možno dynamicky vytvářet objekty, struktury a kolekce
6. není možno volat metody

Pro vazbu OQL a C++ používá své vlastní rozhraní. Každý dotaz je definován jako instance třídy *OQL_Query*. Tato instance se pak stane parametrem funkce *oql_execute*, která vyvolá dotaz na databázi a vrátí výsledky dotazu.

Následující příklad vybere z databáze všechny osoby, které se jmenují Krbec, jsou starší než 70 roků a mají plat vyšší než 5000:

```
define allKrbec as
        select x from x in Persons where x.name = "Krbec";
define oldKrbec as
         select y from y in allKrbec
                        where y.age > 70;
 select z from z in oldKrbec
         where z.salary > 5000
```

5. Závěr

POET 3.0 je sým pojetím zaměřen převážně na možnost persistentního ukládání objektů z prostředí C++ do objektově orientované databáze. Vlastní databáze není přístupna z jiných aplikací. Databázové vlastnosti, které POET nabízí jej řadí do kategorie desktopových databází.
V současné době byla vyvinuta nová verze tohoto databázového systému. POET 4.0 se převážně zaměřil na rozšíření týkají se otevřenosti systému. Je plně podporován jazyk OQL dle standardu ODMG-93, dále je možné přistupovat k datům přes standardní rozhraní ODBC a je zakomponována podpora pro OLE technologii.

6. Literatura

[1]Atkinson M. et al.: The Object-Oriented Database System Manifesto.
In: 1st International Conference on Deductive & Object-Oriented Databases, Kyoto, 1989.

[2]Biliris A., Gehani N. H., Jagadish H. V., Roome W. D.: ODE Object Database and Environment.
AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

[3]Cattel R.G.G.(ed.): The Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, 1994.

[4]Kroha P.: Objects and Databases. McGraw Hill 1993

[5]Kroha P.: Management of Persistent Objects. DATASEM'95.

[6]Kuklová J. Popelinsky L.: Our Experiences with Public-Domain Object-Oriented Databases. In: Sborník konference SOFSEM'93 Hrdoňov 1993.

[7]Dokumentace k systému POET 3.0.