

Objektově relační databáze a SQL 3 a vyšší

V současné době směřuje trend ve vývoji databázových systémů také směrem k objektovým SŘBD, neboť objektové SŘBD umožňují snadněji a přesněji modelovat většinu z různých tříd aplikací. Proto se výrobci relačních SŘBD snaží své produkty rozšířit o některé základní vlastnosti objektových SŘBD, čímž vznikají tzv. *objektově-relační SŘBD*. Zmíněný trend se promítá i do standardu **SQL 3** (aktuální verze je SQL 2011), jehož součástí jsou i abstraktní datové typy, persistentní programové moduly a vhnížděné tabulky. Jedním z objektově-relačních SŘBD je i systém Oracle (od verze 8i), na kterém si nyní ukážeme některá objektová rozšíření relačního SŘBD.

Abstraktní datové typy

Abstraktní datový typ lze definovat příkazem `CREATE TYPE`. Tento uživatelem definovaný ADT lze pak použít všude, kde jsou používány standardní datové typy SQL. Následující příklad ukazuje definici typu `T_ADRESA` reprezentujícího adresu obyvatele. Ukázána je i definice tabulky, kde jedním z atributů je objekt typu `T_ADRESA` :

```
CREATE TYPE t_adresa AS OBJECT (  
    ulice      VARCHAR2(30),  
    cislo      NUMBER(3),  
    obec      VARCHAR2(30),  
    psc        NUMBER(5)  
);  
/
```

```
CREATE TABLE obyvatele (  
    prijmeni  VARCHAR2(30),  
    jmeno     VARCHAR2(30),  
    adresa    t_adresa  
);
```

Informace o struktuře tabulky lze opět získat příkazem `DESC obyvatele`. Strukturu objektu zobrazíme rovněž příkazem `DESC t_adresa`.

Práce s objekty v SQL je, až na malé výjimky, víceméně intuitivní. Vkládání záznamů ukazuje spodní příklad. Instance objektu třídy `T_ADRESA` je vytvořena voláním implicitního konstruktora :

```
INSERT INTO obyvatele (jmeno, prijmeni, adresa)  
VALUES ( 'Jan', 'Pavlasek', t_adresa('Na hrazi', 15, 'Liptakov', 32100) );
```

S objekty lze snadno manipulovat jako s ostatními datovými typy, lze je i porovnávat :

```
SELECT prijmeni, adresa FROM obyvatele;  
  
SELECT prijmeni, jmeno  
FROM obyvatele  
WHERE adresa = t_adresa('Na hrazi', 15, 'Liptakov', 32100);
```

V případě, že chceme přistupovat k jednotlivým datovým prvkům objektu, musíme použít aliasy :

```
SELECT o.jmeno AS jmeno,  
       o.prijmeni AS prijmeni,  
       o.adresa.obec AS obec,  
       o.adresa.psc AS psc  
FROM obyvatele o;
```

Obdobně provedeme i aktualizaci objektu :

```
UPDATE obyvatele o
  SET o.adresa.ulice = 'Pod hrazi',
      o.adresa.cislo = 8
WHERE prijmeni = 'Pavlasek'
AND jmeno = 'Jan';
```

```
UPDATE obyvatele
  SET adresa = t_adresa('Na hrazi', 15, 'Liptakov', 32100)
WHERE prijmeni = 'Pavlasek'
AND jmeno = 'Jan';
```

Objekt lze zrušit takto :

```
UPDATE obyvatele
  SET adresa = NULL
WHERE prijmeni = 'Pavlasek'
AND jmeno = 'Jan';
```

Poznámka: V tomto případě nelze již provést první příkaz UPDATE z předchozího příkladu, protože objekt již neexistuje. Druhý příkaz UPDATE naopak vytvoří instanci novou.

Metody objektů

Samozřejmě, že objekty v SQL 3 mohou mít kromě datových prvků i metody. Následující ukázka slouží pro ilustraci definice objektu s několika metodami. Implementace metod je definována v příkazu CREATE TYPE BODY.

Rozhraní objektu :

```
CREATE OR REPLACE TYPE t_osoba AS OBJECT (
  jmeno    VARCHAR2(30),
  prijmeni VARCHAR2(30),
  naroz    DATE,
  plat     NUMBER(7,2),

  MEMBER FUNCTION vek RETURN INTEGER,
  MEMBER PROCEDURE zvysh_plat(castka IN NUMBER),
  MEMBER PROCEDURE sniz_plat(castka IN NUMBER)
);
/
```

Implementace metod objektu :

```
CREATE OR REPLACE TYPE BODY t_osoba AS

  MEMBER FUNCTION vek RETURN INTEGER IS
  BEGIN
    RETURN TO_NUMBER(SYSDATE - naroz)/365;
  END;

  MEMBER PROCEDURE zvysh_plat(castka IN NUMBER) IS
  BEGIN
    plat := plat + castka;
  END;

  MEMBER PROCEDURE sniz_plat(castka IN NUMBER) IS
  BEGIN
    IF (plat - castka < 0) THEN
      plat := 0;
    END IF;
  END;
```

```

ELSE
    plat := plat - castka;
END IF;
END;

END;
/

```

Řádek v tabulce může být reprezentován i objektem. Tabulku pak nadefinujeme jednoduše takto :

```
CREATE TABLE osoby OF t_osoba;
```

Pro zajímavost si zkuste zobrazit strukturu ADT T_OSoba příkazem DESC T_OSoba a také strukturu tabulky OSOBY příkazem DESC OSOBY.

Nyní vložíme do tabulky několik záznamů a zkusíme zavolat metodu věk() :

```

INSERT INTO osoby (jmeno, prijmeni, naroz, plat)
VALUES ('Jarda', 'Kabrnak', TO_DATE('1.1.1980', 'DD.MM.RRRR'), 1234.56);

INSERT INTO osoby (jmeno, prijmeni, naroz, plat)
VALUES ('Josef', 'Jiricka', TO_DATE('1.1.1970', 'DD.MM.RRRR'), 6543.21);

SELECT o.prijmeni,
       o.jmeno,
       o.vek() AS vek
FROM osoby o;

```

Volání metod uvnitř PL/SQL bloku je stejné jako u ostatních objektově-orientovaných jazyků :

```

DECLARE
    clovek t_osoba;
    c_ref  REF t_osoba;

BEGIN
    clovek := t_osoba('Jan', 'Machacek', TO_DATE('1.1.1975', 'DD.MM.RRRR'), 1122.33);
    clovek.zvys_plat(1000.0);

    INSERT INTO osoby o VALUES(clovek) RETURNING REF(o) INTO c_ref;

    UPDATE osoby o
        SET plat = plat - 100.0
        WHERE REF(o) = c_ref;

END;

```

Pole jako atributy

V praxi se často vyskytují případy, kdy je třeba uložit do jednoho atributu více hodnot. Typickým případem jsou alternativní čísla telefonu, na kterých je dosažitelná určitá osoba. Relační model dat nás nutí vytvořit novou entitu pro telefonní čísla, což se nám může oprávněně zdát poněkud nepřírozené. V takovýchto případech lze s výhodou použít pole proměnné délky - VARRAY. Pole VARRAY může obsahovat různý počet položek stejného datového typu (tzn. i objekty), který nesmí překročit definovanou velikost pole. Způsob použití polí je ukázán v následujícím příkladě :

```

CREATE TYPE t_tel_seznam AS VARRAY(5) OF VARCHAR2(30);
/

CREATE TYPE t_potomek AS OBJECT (
    jmeno VARCHAR2(30),
    vek   NUMBER(3)

```

```

);
/

CREATE TYPE t_pot_seznam as VARRAY(10) OF t_potomek;
/

CREATE TABLE personal (
  jmeno    VARCHAR2(30),
  prijmeni VARCHAR2(30),
  telefony t_tel_seznam,
  deti     t_pot_seznam
);

```

Způsob vkládání a výběru záznamů se nemění:

```

INSERT INTO personal (jmeno, prijmeni, telefony, deti)
VALUES ( 'Jan', 'Kokoska',
        t_tel_seznam('123', '124', '125'),
        t_pot_seznam(t_potomek('Jirka', 10), t_potomek('Jakub', 16)));

SELECT * FROM personal;

```

K jednotlivým záznamům pole lze přistupovat tak, že pole "přetypujeme" na tabulku (*un-nesting*). Všimněte si, že následující dotazy implicitně provádí spojení bázové a vnořené tabulky reprezentované polem:

```

SELECT p.prijmeni,
       d.jmeno,
       d.vek
FROM personal p, TABLE(p.deti) d;

SELECT p.prijmeni,
       COUNT(d.jmeno) AS pocet_deti
FROM personal p, TABLE(p.deti) d
GROUP BY p.prijmeni
ORDER BY p.prijmeni;

```

Nevýhodou polí je, že z hlediska aktualizace se s nimi pracuje jako s atomickými hodnotami. Ve výše uvedeném příkladě tedy nelze jednoduše přidat, změnit nebo odstranit telefonní číslo. Aktualizaci lze provést nejlépe následujícím způsobem:

```

UPDATE personal
  SET telefony = t_tel_seznam('222', '333')
  WHERE prijmeni = 'Kokoska';

```

V PL/SQL je práce s polem VARRAY velmi snadná. K jednotlivým položkám lze přistupovat přes index. Kromě toho každé pole obsahuje atribut count, jehož hodnota udává počet prvků pole:

```

DECLARE
  i INTEGER;
  p personal%ROWTYPE;

BEGIN
  SELECT * INTO p
  FROM personal
  WHERE prijmeni = 'Kokoska';

  dbms_output.new_line;
  dbms_output.put_line('Pocet telefonu : ' || p.telefony.count);

  dbms_output.put_line('Deti :');
  FOR i IN 1..p.deti.count LOOP
    dbms_output.put_line(p.deti(i).jmeno);
  END LOOP;

```

END;

Vhnížděné (vnořené) tabulky

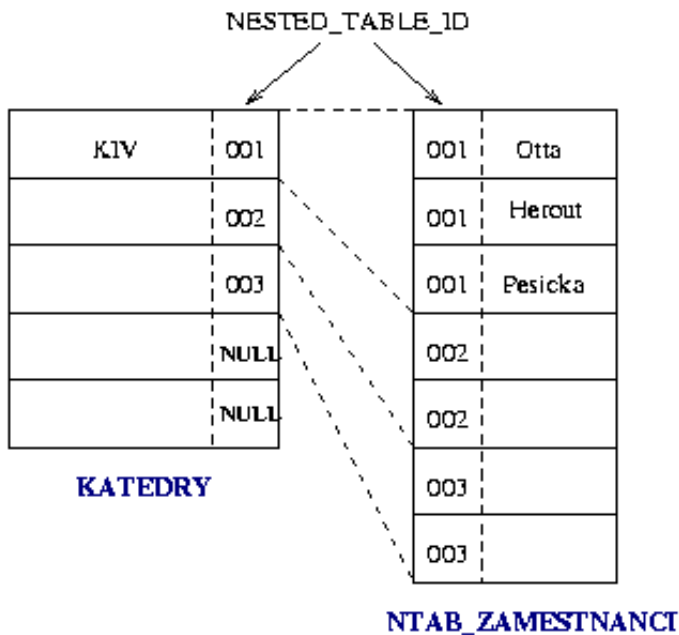
Vnořené tabulky mají oproti polím tu výhodu, že z hlediska aktualizace lze přistupovat k jednotlivým řádkům, jejichž počet není nijak omezen (teoreticky). Nevýhodou však je, že prvky (řádky) v tabulce nemají definované pořadí, jak tomu je u polí. Vnořenou tabulku vytvoříme tak, že nadefinujeme uživatelský datový typ podobně jako u polí:

```
CREATE TYPE t_zamestnanec AS OBJECT (  
  jmeno    VARCHAR2(30),  
  prijmeni VARCHAR2(30),  
  plat     NUMBER(5)  
);  
/  
  
CREATE TYPE tab_zamestnanci AS TABLE OF t_zamestnanec;  
/
```

Definujeme-li tabulku, která obsahuje vnořenou tabulku, musíme definovat jméno tabulky, kde budou fyzicky uloženy záznamy z vnořených tabulek. V uvedeném příkladě to je tabulka NTAB_ZAMESTNANCI:

```
CREATE TABLE katedry (  
  cislo_kat  NUMBER(5),  
  nazev      VARCHAR2(50),  
  zamestnanci tab_zamestnanci  
) NESTED TABLE zamestnanci STORE AS ntab_zamestnanci;
```

Ve skutečnosti jsou vnořené tabulky řešeny "klasickým způsobem" - vazba mezi mateřskou a vnořenou tabulkou je realizována přes skrytý sloupec NESTED_TABLE_ID, jak ukazuje spodní obrázek :



Přestože tabulka NTAB_ZAMESTNANCI je viditelná v katalogu (příkazem SELECT * FROM tab;), není přístupná pro příkazy DML. Zrušíme-li mateřskou tabulku KATEDRY, je automaticky zrušena i tabulka NTAB_ZAMESTNANCI. Vkládání záznamů je stejné jako u polí:

```
INSERT INTO katedry (cislo_kat, nazev, zamestnanci)  
VALUES (111, 'KIV',
```

```

        tab_zamestnanci(t_zamestnanec('Max', 'Otta', 12345),
                       t_zamestnanec('Pavel', 'Herout', 12345),
                       t_zamestnanec('Ladislav', 'Pesicka', 12345)
                       )
);

```

Dotazy nad vnořenými tabulkami:

```
SELECT * FROM katedry;
```

```
SELECT z.*
FROM katedry k, TABLE(k.zamestnanci) z
WHERE k.cislo_kat = 111;
```

Aktualizace dat vnořených tabulek:

```
INSERT INTO TABLE( SELECT zamestnanci
                    FROM katedry
                    WHERE cislo_kat = 111 )
VALUES ('Martin', 'Simek', 12345);
```

```
UPDATE TABLE( SELECT zamestnanci
               FROM katedry
               WHERE cislo_kat = 111 )
SET plat = 9999
WHERE jmeno = 'Max';
```

```
UPDATE TABLE( SELECT zamestnanci
               FROM katedry
               WHERE cislo_kat = 111 ) z
SET VALUE(z) = t_zamestnanec('Martin', 'Simacek', 1234)
WHERE z.prijmeni = 'Simek';
```

```
DELETE FROM TABLE( SELECT zamestnanci
                    FROM katedry
                    WHERE cislo_kat = 111 )
WHERE jmeno = 'Max';
```

Vnořenou tabulku zrušíme takto:

```
UPDATE katedry
SET zamestnanci = NULL
WHERE cislo_kat = 111;
```

Opětovné vytvoření prázdné, nebo naplněné vnořené tabulky:

```
UPDATE katedry
SET zamestnanci = tab_zamestnanci()
WHERE cislo_kat = 111;
```

```
UPDATE katedry
SET zamestnanci = tab_zamestnanci(t_zamestnanec('Max', 'Otta', 9999))
WHERE cislo_kat = 111;
```

Reference na objekty

V některých případech nechceme v atributu uchovávat celý objekt, ale pouze referenci (ukazatel, odkaz) na něj. Proto byl v SQL3 zaveden typ *reference*. Uvažme případ, že máme danou databázi pracovišť a předmětů (nábytek apod.) umístěných na jednotlivých pracovištích. Vzhledem k centrální správě předmětů by bylo poněkud nevýhodné mít u každého pracoviště vnořenou tabulku předmětů. Navíc předměty mohou být rozděleny do několika kategorií a tedy i tabulek. Následující příklad

ukazuje nastíněnou situaci, kdy předměty jsou ukládány v jediné tabulce a u každého předmětu je reference na pracoviště, na němž se nachází:

```
CREATE TYPE t_pracoviste AS OBJECT (  
    zkratka  VARCHAR2(3),  
    nazev    VARCHAR2(50)  
);  
/
```

```
CREATE TABLE kancelare OF t_pracoviste;
```

```
CREATE TYPE t_predmet AS OBJECT (  
    nazev      VARCHAR2(30),  
    ev_cislo   NUMBER(5),  
    pracoviste REF t_pracoviste  
);  
/
```

```
CREATE TABLE predmety OF t_predmet;
```

Několik poznámek k referencím: každý objekt v SŘBD je jednoznačně identifikován pomocí **OID** - *Object Identifier*. OID je jednoznačný v rámci celého SŘBD, v distribuovaných SŘBD v rámci celé sítě. Je-li určitý objekt zrušen, jeho OID již není nikdy použito pro jiný objekt. Typicky je OID generován na základě jednoznačného identifikátoru hostitelského uzlu (např. HW adresa síťového adaptéru uzlu), časové značky (aktuální čas na uzlu) a sekvenčního čísla.

Pro ilustraci si vyzkoušejte vytvořit nový objekt a vypsát jeho OID (referenci):

```
INSERT INTO kancelare (zkratka, nazev)  
VALUES ('KIV', 'Kancelar KIV');
```

```
SELECT REF(k)  
    FROM kancelare k  
    WHERE zkratka = 'KIV';
```

Reference na objekty nelze považovat za referenční integritní omezení. Referenční IO omezují rozsah hodnot atributu (většinou cizího klíče), kdežto reference je chápána jako ukazatel na určitý objekt, který ovšem nemusí existovat (vzpomeňte si na ukazatele v C ;-). Nicméně u referencí lze specifikovat rozsah, a to ve smyslu množiny objektů ve specifikované tabulce:

```
DROP TABLE predmety;
```

```
CREATE TABLE predmety OF t_predmet (SCOPE FOR (pracoviste) IS kancelare);
```

Referenci na určitý objekt vložíme do záznamu takto:

```
INSERT INTO predmety (nazev, ev_cislo, pracoviste)  
VALUES ('Zidle', 555, (SELECT REF(k) FROM kancelare k WHERE k.zkratka = 'KIV'));
```

V případě, že se budeme snažit vložit do sloupce PRACOVISTE v tabulce PREDMETY referenci na objekt z jiné tabulky než KANCELARE, databáze vkládání záznamu odmítne hláškou:

```
ORA-22889: REF value does not point to scoped table
```

Přístup k objektům přes reference (dereference) je velmi snadný:

```
SELECT p.nazev,  
    p.ev_cislo,  
    p.pracoviste.zkratka  
    FROM predmety p;
```

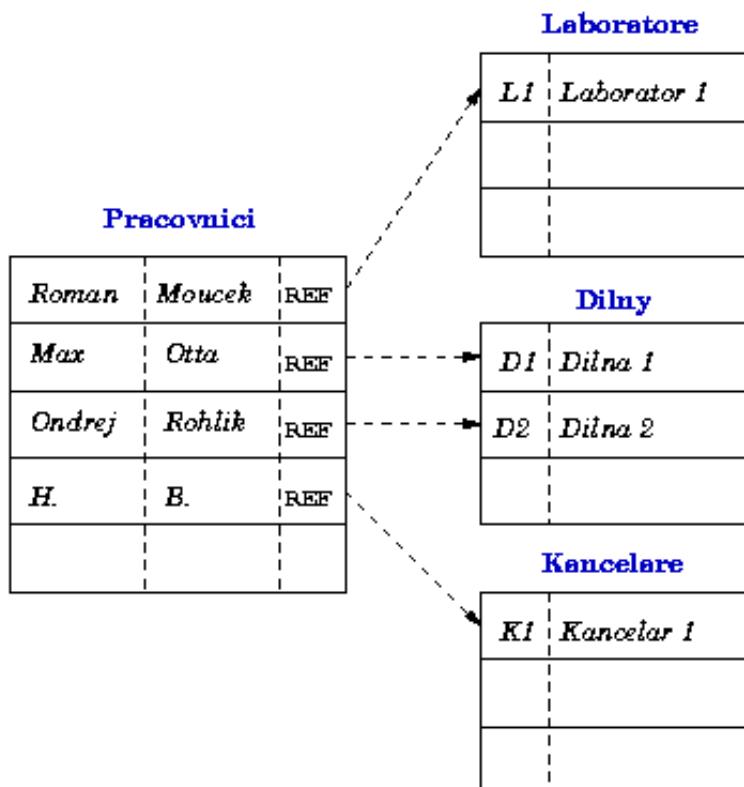
Reference na neexistující objekty lze najít také snadno:

```
DELETE FROM kancelare;
```

```
SELECT p.nazev  
FROM predmety p  
WHERE p.pracoviste IS DANGLING;
```

Příklad : evidence zaměstnanců a jejich pracovišť

Uvažme databázi nějakého ústavu, kde jsou centrálně evidováni zaměstnanci, jednotlivá pracoviště jsou evidována zvlášť. Vazba mezi pracovníkem a jeho pracovištěm je realizována referencí u záznamu pracovníka na záznam jeho pracoviště. Celou situaci ilustruje obrázek :



Z předchozích příkladů využijeme typ T_PRACOVISTE a tabulku KANCELARE. Zbývající tabulky vytvoříme takto:

```
CREATE TABLE dilny OF t_pracoviste (  
    PRIMARY KEY (zkratka)  
);
```

```
CREATE TABLE laboratore OF t_pracoviste (  
    PRIMARY KEY (zkratka)  
);
```

```
CREATE TYPE t_pracovnik AS OBJECT (  
    prijmeni    VARCHAR2(30),  
    jmeno      VARCHAR2(30),  
    pracoviste REF t_pracoviste  
);  
/
```

```
CREATE TABLE pracovníci OF t_pracovnik;
```

Tabulky ještě naplníme daty:


```
INSERT INTO dilny VALUES ('D1', 'Dilna 1');
INSERT INTO dilny VALUES ('D2', 'Dilna 2');
INSERT INTO kancelare VALUES ('K1', 'Kancelar 1');
INSERT INTO laboratore VALUES ('L1', 'Laborator 1');
```

```
INSERT INTO pracovnici
VALUES ('Max', 'Otta', (SELECT REF(p) FROM dilny p WHERE zkratka = 'D1'));
```

```
INSERT INTO pracovnici
VALUES ('Ladislav', 'Pesicka', (SELECT REF(p) FROM dilny p WHERE zkratka = 'D1'));
```

```
INSERT INTO pracovnici
VALUES ('Ondrej', 'Rohlik', (SELECT REF(p) FROM dilny p WHERE zkratka = 'D2'));
```

```
INSERT INTO pracovnici
VALUES ('Helena', 'Benesova', (SELECT REF(p) FROM kancelare p WHERE zkratka = 'K1'));
```

```
INSERT INTO pracovnici
VALUES ('Roman', 'Moucek', (SELECT REF(p) FROM laboratore p WHERE zkratka = 'L1'));
COMMIT;
```

Seznam všech zaměstnanců včetně jejich příslušnosti k pracovišti na závěr vypíšeme jistě elegantním způsobem:

```
SELECT p.prijmeni,
       p.jmeno,
       p.pracoviste.nazev
FROM pracovnici p;
```