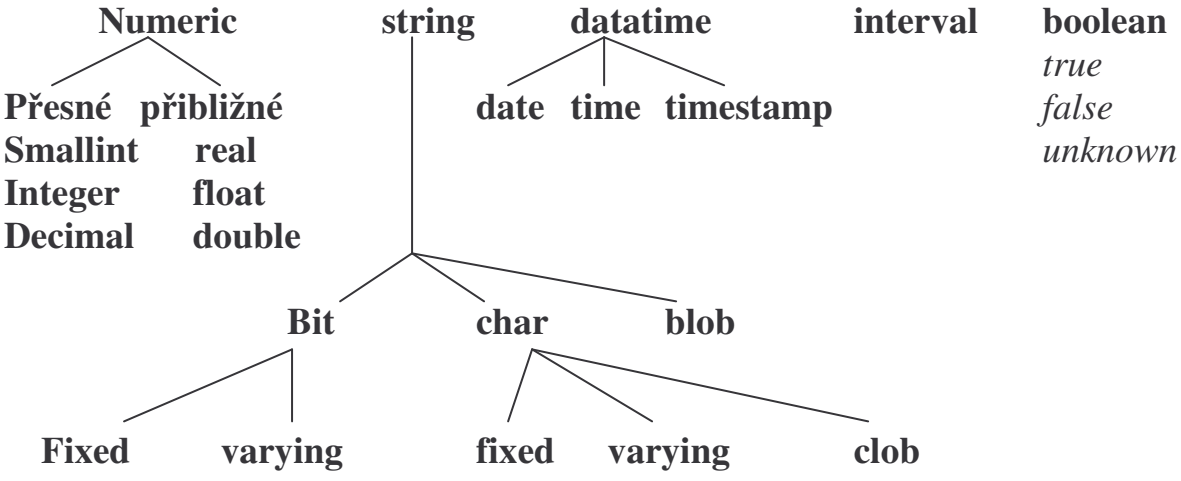
Objektově relační database

Vývoj standardů SQL:

SQL86
SQL89
SQL92 (dosud ne zcela implementován, např. možnosti IO)
SQL/Call Level Interface 95
SQL/Persistent Stored Module Language Interface 96
SQL/Java
SQL99
SQL/Object Language Bindings 2000
SQL/Management External Data 2000
SQL/OLAP
SQL/temporal
SQL/Schemata
SQL/XML
SQL/MM -Framework (základy)
-Full Text
-Spatial
-Still Image
-General Purpose Facilities (společné ostatním částem, např. datové typy)

Pracovní názvy: SQL1 (>SQL86)
SQL2 (>SQL92),
SQL3 (>SQL99),
SQL4

Předdefinované typy SQL99



Nové typy SQL99

Konstruované atomické typy:

-reference

-odlišující typy

např.CREATE TYPE nějaký_typ
AS CHAR(5) FINAL;

Konstruované kompozitní typy:

-array (jsou podtypem collection)

-row

-ADT

př. CREATE TABLE knihy (
cena INTEGER
autoři VARCHAR (30) ARRAY[8]
titul VARCHAR (50)
)

-poziční přístup ke složkám, např. autoři [2]

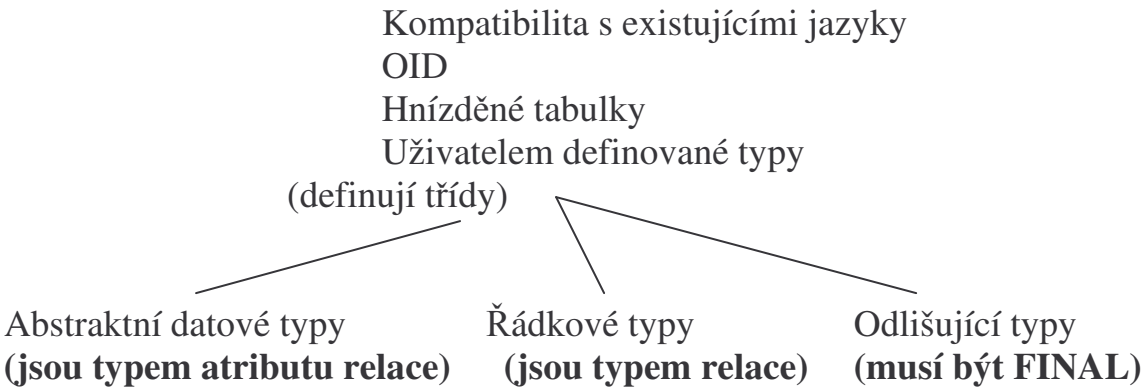
-odhnízdění pomocí UNNEST

např. SELECT a.jméno
FROM knihy AS z,
UNNEST (Z.autoři) AS a(jméno)

Př. Použití velkých objektů

```
CREATE TABLE Filmy (
    Jméno     CHAR VARYING (30)  NOT  NULL,
    Režiser   CHAR VARYING (20),
    Rok       INTEGER,
    Scénář    CLOB (20M),
    Video     BLOB (3G),
    PRIMARY KEY (jméno)) ;
```

Objektové vlastnost SQL99



-UDT mohou být organizovány do hierarchií s děděním
-chování UDT je realizováno pomocí procedur, funkcí a (metod u ADT)

Objekty v SQL99 pracují s relacemi

Řádkové typy

CREATE ROW TYPE jméno (deklarace komponent)

př.) Řádkový typ reprezentující herce

```
CREATE ROW TYPE typadresa ( ulice CHAR VARYING ( 50 ) ,
                            město CHAR VARYING ( 20 )
                          ) ;
CREATE ROW TYPE typherec ( jméno CHAR VARYING ( 30 ) ,
                           adresa typadresa
                         ) ;
```

Deklarace relace s pojmenovaným řádkovým typem

např.) CREATE TABLE Filmovýherec OF typherec ;
obdoba extent třídy: Herec ---- typherec

Deklarace relace s nepojmenovaným řádkovým typem

př.) CREATE TABLE Filmovýherec (
jmeno CHAR VARYING (30),
adresa ROW (
ulice CHAR VARYING (50),
město CHAR VARYING (20)
)) ;

Zpřístupnění komponent řádkového typu

př.) Najdi jméno a ulici každého herce z Plzně

SELECT Filmovýherec . jméno , Filmovýherec . adresa . ulice

FROM Filmovýherec

WHERE Filmovýherec . adresa . město = ´ Plzeň ´ ;

Abstraktní datové typy v SQL99

umožňují zapouzdření atributů a operací (na rozdíl od řádkových typů)
Hodnoty jejich typů mohou být umístěny do sloupců tabulek

Definice ADT :
(1) CREATE TYPE jméno typu AS

(2) seznam atributů a jejich typů

(3) nepovinná deklarace metod

(4) údaje o děditelnosti a instalovatelnosti

- Další funkce lze deklarovat vně příkazu CREATE TYPE
(nejsou svázány s ADT)

Př. CREATE TYPE typzaměstnanec AS (
č_zam INTEGER,
jméno CHAR (20),
adresa typadresa,
vedoucí typzaměstnanec,
datum nástupu DATE,
základní plat DECIMAL(6,2))
INSTANTIABLE NOT FINAL,
METHOD odpr_léta( ) RETURNS INTEGER; /*jen signatury*/
METHOD mzda ( ) RETURNS DECIMAL;

CREATE METHOD odpr_léta
FOR typzaměstnanec
BEGIN … END ;

CREATE METHOD mzda
Lze uvést **Language** ***progr. jazyk*****,**
FOR typzaměstnanec
BEGIN … END;

Instance ADT vznikají:

1. **konstruktorem** *jménotypu( )*
2. **operátorem** *NEW jméno hodnota*
   např. …WHERE vedoucí = NEW typzam(10234, ‘ Petr Nový...
3. **příkazem INSERT**
   např. INSERT INTO osoby
   VALUES (10234,‘Petr Nový‘, …) ;

Pro každý atribut jsou k dispozici funkce:

-implicitně či explicitně zavedené porovnání
-zjištění hodnoty atributu z objektu
jméno atributu(jméno objektu)
stejné i pro aplikaci metod
např. odpracovaná_léta(X) **-virtuální atribut**
možná i tečková notice **X.odpracovaná_léta**

<u>Funkce, procedury a metody</u>

-vyjádřeny v SQL/PSM, nebo C/C++, Fortran, ADA, Java, …
-metody jsou svázány s ADT
-Uživatelem definovaný typ je vždy prvním (! nedeklarovaným !) argumentem metody (viz. odpracovaná_leta(X))
-metody jsou uloženy ve schematu typu definovaném uživatelem
-metody se dědí
-metody i funkce mohou být polymorfní (liší se způsobem výběru)
-funkce a procedury se daklarují zápisem
CREATE FUNCTION resp. CREATE PROCEDURE

Př. CREATE PROCEDURE zjisti_cenu
(IN číslo INTEGER, OUT c DOUBLE PRECISION)
SELECT cena INTO c
FROM knihy WHERE inv_číslo=číslo;

Př. CREATE FUNCTION zjisti_cenu
(číslo INTEGER) RETURNS DOUBLE PRECISION
BEGIN
DECLARE c DOUBLE PRECISION;
SELECT cena INTO c
FROM knihy WHERE inv_číslo=číslo;
RETURN c
END

Volání procedur příkazem CALL zjisti_cenu(12134, z) ;

Podtypy

CREATE TYPE typkulisák UNDER typzaměstnanec AS (
Další atributy a metody
) ;

- **jen jednoduché dědění**
- **dědí atributy i metody svých nadtypů**
- **strukturované typy musí být NOT FINAL**
- **odlišující typy musí být FINAL**

Podtabulky

-dědí atributy, IO, triggery, … z dané nadtabulky
-mohou mít další sloupce
-každému řádku podtabulky odpovídá právě jeden řádek nadtabulky

př. **CREATE TABLE osoba (**
Jméno CHAR(30),
Sex CHAR(1),
Věk INTEGER) ;

CREATE TABLE zaměstnanec UNDER osoba (
Mzda FLOAT) ;

CREATE TABLE klient UNDER osoba (
Č_účtu INTEGER) ;

Reference

REF USING ***předdefinovaný typ***
REF (***jméno UDT*** **)**
REF IS SYSTÉM GENERATED může být také pojmenován
REF IS USER GENERATED **„ ---„**
nahrazují OID tj. odkaz na hodnotu (n-tici) UDT

př.) Do tabulky Filmovýherec chceme zaznamenat nejlepší hercův film

```
CREATE   TYPE  Filmtyp  AS  (
                                titul   CHAR ( 20 ) ,
                                rok     INTEGER ,
                                vBarvě BIT (1)
                              ) ;

CREATE  TABLE  Film  OF   Filmtyp ;

CREATE   TYPE   Typherec  AS (
                                jméno  CHAR ( 30 ) ,
                                adresa   typadresa,
                                nejlepšífilm  REF ( Filmtyp )
                               ) ;
```

Pro vyjádření relace M : N

ODL dovoluje množinu objektů jako komponentu obj. typu

SQL99 vyžaduje reprezentaci dodatečnou relací

př.) Vyjádřit relaci M : N mezi filmy a herci

```
CREATE  TYPE  Filmtyp AS (
        titul CHAR(30),
        rok  INTEGER,
        vBarve  BIT(1)
);

CREATE  TYPE  typadresa AS (
        ulice  CHAR(50),
        město  CHAR(20)
);

CREATE  TYPE  typherec  AS(
        jmeno  CHAR(30),
        adresa  typadresa
);

CREATE  TYPE hrajevtyp AS (
        herec  REF(typherec),
        film  REF(Filmtyp)
);

CREATE TABLE  Film  OF   Filmtyp;
CREATE TABLE  Filmovyherec  OF   typherec;
CREATE TABLE  hrajev  OF   hrajevtyp;
```

Dereference

Je-li x typu odkaz na n-tici t ,
a je atributem t

pak x -> a značí hodnotu atributu a v n-tici t

př.) Najdi tituly všech filmů, ve kterých hraje Chaplin

```
SELECT   Film -> titul
    FROM hrajev   WHERE herec -> jméno = ´Chaplin´ ;
```

pozn. -> lze použít jen k odkazu na n-tici

Pro jednoznačnou identifikaci tabulky, na kterou ukazatel odkazuje lze použít:

atribut WITH OPTIONS SCOPE tabulka

př.)
```
CREATE TYPE hrajevtyp AS (
                herec  REF ( typherec ) ,
                film   REF ( Filmtyp )
                ) ;

CREATE TABLE Hrajev OF TYPE hrajevtyp
                herec WITH OPTIONS SCOPE Filmovýherec ,
                film WITH OPTIONS SCOPE Film ;
```

OID hodnota v SQL99

zpřístupněna klauzulí REF IS SYSTEM GENERATED
REF IS USER GENERATED

```
CREATE TYPE osoba_t AS (
        Jméno  VARCHAR(30),
        Rok_narození INTEGER );

CREATE TYPE zaměstnanec_t UNDER osoba_t AS (
        Plat  INTEGER );

CREATE TYPE student_t  UNDER osoba_t AS (
        Specializace VARCHAR(30) );

CREATE TYPE katedra_t AS (
        Jméno VARCHAR(20),
        Rozpočet INTEGER,
        Vedoucí REF (zaměstnanec_t)) ;

ALTER TYPE zaměstnanec_t
        ADD ATTRIBUTE katedra REF (katedra_t) ;

CREATE TABLE osoby OF osoba_t
        (REF IS oid USER GENERATED);

CREATE TABLE zaměstnanci OF zaměstnanec_t  UNDER osoby
        (katedra WITH OPTIONS SCOPE  katedry);

CREATE TABLE studenti OF student_t UNDER osoby ;

CREATE TABLE katedry OF katedra_t
        ( REF IS oid USER GENERATED,
          vedoucí WITH OPTIONS SCOPE zaměstnanci) ;
```

```
SELECT  z.* FROM zaměstnanci z
        WHERE z.rok_narozeni > 1980 AND plat > 55000;


INSERT INTO zaměstnanci
        VALUES  (zaměstnanec_t('z100'), 'Jan Kozina', 1966, 44000,
                (SELECT oid FROM katedry WHERE jméno = 'KIV')) ;


UPDATE osoby SET rok_narození = 1959
        WHERE jméno = 'Jan Kozina';


SELECT  Z.jméno, Z.katedra -> jméno
        FROM zaměstnanci Z
        WHERE Z.katedra -> vedoucí -> jméno = 'Jan Sladký';
```

Podpora typově závislých dotazů

```
SELECT *
        FROM ONLY (zaměstnanci) Z
                WHERE Z.katedra -> rozpočet > 10000000 ;


SELECT jméno
        FROM osoby O
        WHERE DEREF(oid) IS OF TYPE
                        (ONLY zaměstnanec_t, student_t) ;


SELECT  type _name (DEREF (Z.oid)), Z . *
        FROM  OUTER (zaměstnanci) Z
        WHERE Z.oid = zaměstnanec_t('z1225');
```

Př. použití objektových typů včetně metod

```
CREATE TYPE filmtyp AS (
        Titul CHAR(30),
        Rok INTEGER )
        INSTANTIABLE NOT FINAL
        REF IS SYSTEM GENERATED
        METHOD rating ( ) RETURN DECIMAL (2,1);

CREATE TYPE typherec AS (
        Jmeno CHAR(30),
        Adresa typadresa,
        Film REF (filmtyp) )
        INSTANTIABLE NOT FINAL
        REF IS SYSTEM GENERATED;

CREATE TABLE Filmy OF filmtyp
        (REF IS film_id SYSTEM GENERATED);

CREATE TABLE herci OF typherec
        (REF IS herec_id SYSTEM GENERATED,
        Film WITH OPTION SCOPE  Filmy);
```

Použití:

```
SELECT Film → rok
                FROM herci
                WHERE jmeno = 'Werich';
```

Objektově relační vlastnosti Oracle

Definice typů **je podobná SQL99. Syntax má tvar:**

```
CREATE TYPE t AS OBJECT (
       list of attributes and methods
);
/
```

Př.
```
CREATE TYPE PointType AS OBJECT (
             x NUMBER,
             y NUMBER
);
 /
```

Objekt může být použit jako ostatní typy v deklaracích objektových typů nebo tabulkových typů

Př.
```
CREATE TYPE LineType AS OBJECT
     ( end1 PointType,
       end2 PointType
      );
/

  CREATE TABLE Lines (
     lineID INT,
     line   LineType
   );
```

Odstranění typů:

```
DROP TYPE Linetype;
```

Vytváření hodnot objektů - konstruktory:

```
INSERT INTO Lines
      VALUES(27, LineType(
                      PointType(0.0, 0.0),
                      PointType(3.0, 4.0)
                       )
         );
```

Deklarace a definice metod:

Deklarace pomocí - MEMBER FUNCTION
- MEMBER PROCEDURE
v CREATE TYPE příkazu

Definice metody (kód) je separována do CREATE TYPE BODY příkazu

Proměnná *SELF* je v metodě k dispozici pro odkaz na aktuální n-tici

Př. Přidat k *LINETYPE* funkci *length*

```
CREATE TYPE LineType AS OBJECT (
   end1 PointType,
   end2 PointType,
   MEMBER FUNCTION length(scale IN NUMBER)
                                RETURN NUMBER,
   PRAGMA RESTRICT_REFERENCES(length, WNDS)
 );
 /
```

- **Specifikace argumentů - *in, out, inout***
- **Bezargumentové metody - *foo( )***
- **Pragma zajistí, že length nebude modifikovat databázi (WNDS = write no database state). Je to nutné, použijeme-li *length* v dotazu**

Všechny metody pro typ jsou definovány v jednom *CREATE BODY* příkazu.

Př.
```
CREATE TYPE BODY LineType AS
    MEMBER FUNCTION length(scale NUMBER)
                RETURN NUMBER IS
        BEGIN RETURN scale *
            SQRT((SELF.end1.x-SELF.end2.x)
                *
                (SELF.end1.x-SELF.end2.x)
                +
                 (SELF.end1.y-SELF.end2.y)
                *
                (SELF.end1.y-SELF.end2.y)
            );
    END;
    END;
    /
```

Dotazování relací s definovanými typy

Př. Nalezení délek všech úseček v relaci *Lines* s použitím faktoru měřítka 2.

```
SELECT lineID, ll.line.length(2.0)
       FROM Lines ll;
```

- **Nutnost použití alias**
- **Jestliže N odkazuje na objekt O typu T, a jedna z komponent (metoda nebo atribut) typu T je A, pak N.A odkazuje na tuto komponentu objektu O**

Př. Zjištění souřadnic *x* a *y* začátků všech úseček

```
SELECT ll.line.end1.x, ll.line.end1.y FROM Lines ll;
```

Př. Výpis konců všech úseček jako hodnot typu *PointType* (ne dvojice čísel)

```
SELECT ll.line.end2 FROM Lines ll;
```

Typ v Oracle9 může být i relačním schématem **- Soulad s SQL99 typy**

- **Abstraktní datové typy**
- **Řádkové typy - tj. typ relace, konstrukcí** ***CREATE TABLE … OF***

Př. Vytvoření relace, jejíž každá n-tice je dvojicí bodů

```
CREATE TABLE Lines1 OF LineType;
          ≡
CREATE TABLE Lines1
     ( end1 PointType, end2 PointType );
```

Metoda ***length je*** **k dispozici odkazujeme-li na n-tici z** ***lines1***

```
SELECT AVG(ll.length(1.0))
     FROM Lines1 ll;
```

Odkazy (reference) jako typy

Pro každý typ T je ***ref T*** **typ referencí (OID chceme-li) na hodnoty typu T.**

Př. Vytvoření relace ***Lines2,*** **jejíž n-tice jsou dvojice odkazů na body**

```
CREATE TABLE Lines2 (
        end1 REF PointType,
        end2 REF PointType
     );
```

Předp. existenci relace ***Points,*** **jejíž n-tice jsou objekty typu** ***PointType***

```
CREATE TABLE Points OF PointType;
```

Př. Naplnění *Lines2* dvojicemi bodů (úsečkami, tvořenými body z *Points*)

```
INSERT INTO Lines2
          SELECT REF(pp), REF(qq)
          FROM Points pp, Points qq
          WHERE pp.x < qq.x;
```

Nelze vložit ale
```
VALUES(REF(PointType(1,2)),
       REF(PointType(3,4)))
```
protože body jako *PointType(3,4)* nejsou součástí relace

K odkazům na složky objektů je použita "." notace

Př. Výběr souřadnic x všech krajních bodů úseček v *Lines2*

```
SELECT ll.end1.x, ll.end2.x
      FROM Lines2 ll;
```

Hnízděné tabulky **Typ sloupce může být tabulka**

a	B		
-	x	y	z
	-	-	-
	-	-	-
	-	-	-
-	x	y	z
	-	-	-
-	x	y	z
	-	-	-
	-	-	-

CREATE TYPE PolygonType AS TABLE OF PointType;
/

Př. Deklarace relace, jejíž sloupce mají hodnoty polygonů

```
CREATE TABLE Polygons (
          name    VARCHAR2(20),
          points PolygonType)
          NESTED TABLE points STORE AS PointsTable;
```

pozn. hodnotami sloupce points jsou tabulky dvojic bodů. Řádky této vnořené tabulky jsou uloženy ve zvláštní tabulce definované konstrukcí NESTED TABLE jméno_sloupce STORE AS uložení_sloupce. Při použití více sloupců ve tvaru vnořených tabulek, je třeba definovat takovou "ukládací tabulku" pro každý typ vnořené tabulky. Data vnořené tabulky jsou uložena mimo rodičovskou tabulku. Propojení si Oracle zajistí sám.

Vkládání do relací se sloupci typu hnízděné relace je prováděno pomocí konstruktoru typu hnízděné relace (zde *PolygonType*)
Vkládané hodnoty jsou rovněž označeny typem

Př. Vložení polygonu "square"

```
INSERT INTO Polygons
VALUES( 'square',
 PolygonType(PointType(0.0, 0.0), PointType(0.0,
1.0),
              PointType(1.0, 0.0), PointType(1.0, 1.0)
             ) );
```

Dotaz na rohy čtverce: `SELECT points FROM Polygons WHERE name = 'square';`

Př. Dotaz na body polygonu (čtverce), ležící na hlavní diagonále (tj. x=y)

```
SELECT ss.x
     FROM THE(SELECT points
              FROM Polygons
              WHERE name = 'square'
             ) ss
     WHERE ss.x = ss.y;
```

Kombinace hnízděných relací a referencí

atributy hnízděných tabulek nemají jméno

Oracle - *COLUMN_VALUE*

Př.

```
CREATE TYPE PolygonRefType
           AS TABLE OF REF PointType;
/
```

```
CREATE TABLE PolygonsRef (
    name VARCHAR2(20),
    pointsRef PolygonRefType)
    NESTED TABLE pointsRef STORE AS PointsRefTable;
```

pozn. PointsRef je sloupec obsahující tabulky referencí na vrcholy mnohoúhelníků. PointsRefTable je "ukládací" tabulkou pro data vnořené tabulky pointsRef.

Dotaz na body hlavní diagonály v hnízděné tabulce použije k referenci *COLUMN_VALUE*

```
SELECT ss.COLUMN_VALUE.x
     FROM THE(SELECT pointsRef
              FROM PolygonsRef
              WHERE name = 'square'
             ) ss
     WHERE ss.COLUMN_VALUE.x = ss.COLUMN_VALUE.y;
```

Konverze normálních relací na objektové relace

standardní SQL typy → uživatelem definované objektové typy

př. Předpokládejme relaci ***LinesFlat***

```
CREATE TABLE LinesFlat(
            id INT,
            x1 NUMBER, y1 NUMBER,
            x2 NUMBER, y2 NUMBER
);
```

```
INSERT INTO Lines
   SELECT id,
        LineType(PointType(x1,y1),
        PointType(x2,y2))
   FROM LinesFlat;
```

např. vložení bodu (2,0, 3,0) do polygonu "triangle"

```
INSERT INTO THE(SELECT points
                  FROM Polygons
                  WHERE name = 'triangle'
                )
     VALUES(PointType(2,0, 3.0));
```

Př. Předpokládejme relaci ***PolyFlat,*** **reprezentující body polygonů**

```
CREATE TABLE PolyFlat (
          name VARCHAR2(20),
          x    NUMBER,
          y    NUMBER
     );
```

Jsou-li body čtverce reprezentovány v *PolyFlat* lze je kopírovat do *Polygons* pomocí kroků:

1. **Dotázat se v *PolyFlat* na body čtverce**

2. **Klíčovým slovem *MULTISET* konvertovat kolekci odpovědí na relaci**

3. **Konvertovat relaci na hodnotu typu *PolygonType* pomocí výrazu *CAST… AS PolygonType***

4. **Použít 'square' a hodnotu konstruovanou v (3) jako argumenty výrazu *VALUES***

```
INSERT INTO Polygons VALUES('square',
          CAST(
              MULTISET(SELECT x, y
                       FROM PolyFlat
                       WHERE name = 'square'
                      )
              AS PolygonType
          )
);
```

Př. Kopírování dat z PolyFlat do Polygons (všechny polygony a jejich množiny bodů) - téměř funguje

```
INSERT INTO Polygons
         SELECT pp.name,
             CAST(
                 MULTISET(SELECT x, y
                          FROM PolyFlat qq
                          WHERE qq.name = pp.name
                         )
                 AS PolygonType
             )
         FROM PolyFlat pp;
```

fungující řešení

```
INSERT INTO Polygons
    SELECT pp.name,
        CAST(
            MULTISET(SELECT x, y
                        FROM PolyFlat qq
                        WHERE qq.name = pp.name
                       )
            AS PolygonType
        )
    FROM PolyFlat pp
    WHERE NOT EXISTS(
          SELECT *
          FROM PolyFlat rr
          WHERE   rr.name = pp.name AND
                  rr.x < pp.x OR
                  rr.x = pp.x AND rr.y < pp.y
    );
```