

Objektum-relációs adatbázisok

Felhasználói típusok (User-Defined Types)

Objektum ID-k

Beágyazott táblák (Nested Tables)

Relációs és az O-O modell egyesítése

- Az O-O modell több érdekes adattípust támogat – nem csak egyszerű állományokat
 - Térkép, multimédia, stb.
- A relációs modell magas szintű lekérdezéseket támogat
- Objektum-relációs adatmodell egy olyan kísérlet, amely mindkét világból a legjobbat szeretné nyújtani

Az adatbázis-kezelő rendszerek (DBMS) fejlődése

- Az O-O adatbáziskezelő rendszerek sokáig nem mutattak olyan hatékonyságot a jól bevált relációsokkal szemben, amely lehetővé tette volna az elterjedésüket.
- A relációs DBMS-ek objektum-relációs kiterjesztése az O-O megközelítés több előnyös tulajdonságát megragadja, mégis megtartja a relációt mint az alapvető absztrakciós mechanizmust és adatszerkezetet

SQL-99 és az ORACLE szolgáltatásai

- SQL-99 több objektum relációs szolgáltatás leírását tartalmazta.
- Azonban mivel viszonylag új gondolat és szabvány volt abban az időben, minden gyártó a saját megközelítését és megvalósítását használta.
 - A példákban általában az ORACLE szolgáltatásait és szintaxisát használjuk

Felhasználó által definiált adattípus

- **Felhasználó által definiált adattípus (UDT** rövidítés), egy O-O *osztály* definíciója, amely egy adatszerkezet és metódusai.
 - Azonos „típusú” objektumok egy osztály definiálnak
 - Viselkedés: metódusok halmazával kifejezve, amelyek az osztályhoz tartozó objektumokon hajthatóak végre

Felhasználó által definiált adattípus

- Két használati módja van:
 1. *Sortípus*, vagyis egy relációt, mint adattípust kezelünk.
 2. Egy reláció attribútumának a típusa.

Felhasználó által definiált adattípus

```
CREATE TYPE <typename> AS (  
    <list of attribute-type pairs>  
);
```

- **ORACLE-ben:** CREATE TYPE <typename>
AS OBJECT
- Utána lehet a típust eltávolítani.

Példa: UDT létrehozásra

```
CREATE TYPE BarType AS (  
    name CHAR(20),  
    addr CHAR(20)  
);
```

```
CREATE TYPE BeerType AS (  
    name CHAR(20),  
    manf CHAR(20)  
);
```


Példa: UDT létrehozásra Oracle-n belül

```
CREATE TYPE SDO_POINT_TYPE AS OBJECT  
(  
    X          NUMBER,  
    Y          NUMBER,  
    Z          NUMBER  
);
```

Hivatkozások

- Ha T egy UDT, akkor $\text{REF } T$ a T –re történő hivatkozás típusa, vagyis egy mutató egy T típusú objektumra.
- Ezt „objektum azonosítónak” (OID) is hívják O-O rendszerekben.
- Gyakorlatilag az OID élete végéig azonosít egy objektumot, függetlenül a komponenseinek/mezőinek értékeitől

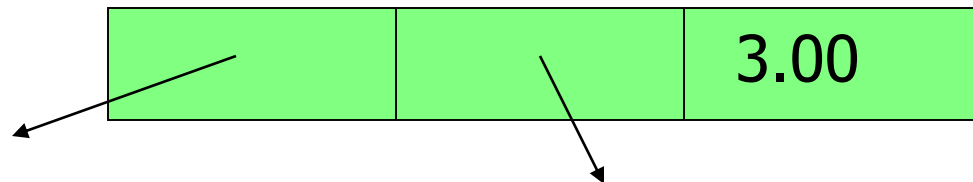
Hivatkozások

- Azonban az *OID*-től eltérően – amelyek alapértelmezésben nem láthatók -, *REF* látható, bár általában nehezen értelmezhető.

Példa: REF

```
CREATE TYPE MenuType AS (  
  bar    REF BarType,  
  beer   REF BeerType,  
  price  FLOAT  
);
```

- MenuType objektum valahogy így néz ki:



Egy *BarType*
objektumra hivatkozás

Egy *BeerType*
objektumra hivatkozás

UDT-k, mint sortípusok

- Egy relációs táblát egy *sortípus* segítségével mint sémával lehet definiálni, az elemeinek felsorolása helyett
- Szintaxis:

```
CREATE TABLE <table name> OF  
    <type name>;
```

Példa: Egy reláció készítése

```
CREATE TABLE Bars OF BarType;
```

```
CREATE TABLE Beers OF BeerType;
```

```
CREATE TABLE Sells OF MenuType;
```

Sortípusú relációk értékei

- A *kocsmák (Bars)* relációt lehet, úgy definiálni, hogy a típusa a *KocsmiTípus (BarType)*, ez egy unáris reláció - nem párok halmaza -, amelynek a sorai két komponenst/mezőt tartalmaznak: *név* és *cím*.
- Mindegyik *UDT*-nek van egy típus konstruktora, amely összefogja ehhez a típushoz tartozó objektumokat.

Példa: típuskonstruktor

- Lekérdezés

```
SELECT * FROM Bars;
```

- Eredmény sora:

BarType('Joe''s Bar', 'Maple St.')

Sortípus értékeinek elérése

- ORACLE-ben a pont („.”) az elvártaknak megfelelően működik.
 - Azonban az ORACLE-ben kötelező minden relációra egy alias-t használni akkor, amikor az O-R szolgáltatásokkal kezeljük (pl. amikor az objektum mezőire hivatkozunk)
- Példa:

```
SELECT bb.name, bb.addr  
FROM Bars bb;
```

SQL-99 jellegű megközelítés

- SQL-99-ben, mindegyik *UDT*-nek vannak *generátorai* (vedd ki az értéket) és *mutátorai* a (változtasd meg az értéket), amelyeknek mint metódusoknak a nevei megegyeznek a mezők neveivel.
 - Pl . Az *A* mező *generátorának* nincs argumentuma *A()*.
 - Az *A* mező *mutátorának* az új érték az argumentuma pl. *A(v)*.

Példa: SQL-99 jellegű adatelérés

- Az előbbi lekérdezés SQL-99-ben:

```
SELECT bb.name(), bb.addr()  
FROM Bars bb;
```

Sortípusú érték beillesztése

- ORACLE-ben a szabványos INSERT-et használják
 - De ne feledjük, hogy egy sortípusú reláció unáris, és ezért szükség van a típuskonstruktorokra.
- Példa:

```
INSERT INTO Bars VALUES (  
    BarType('Joe' 's Bar', 'Maple  
St. ' )  
    ) ;
```

Értékek beszúrása SQL-99 stílusban

- Egy alkalmas típusú X változót hozzunk létre, használva e típus típuskonstruktorát, mint metódust.
- Használjuk a *mutátor* metódust az attribútumokra azért, hogy az X változó mezőinek értékét megadhassuk.
- Illesszük be az X változó értékeit a relációba

SQL-99 beillesztés példa

- Ez a lekérdezés egy *eljárás* része lehet, ezért van egy új változó, *newBar*.
- A *mutátor* metódusok megváltoztatják a név és cím komponenst.

```
SET newBar = BarType();  
    newBar.name('Joe''s Bar');  
    newBar.addr('Maple St.');  
INSERT INTO Bars VALUES(newBar);
```

UDT-k, mint oszloptípusok

- UDT lehet egy attribútum típusa.
- Akár egy UDT deklarációban, vagy egy CREATE TABLE utasításban, az UDT típus neve úgy használható mint az attribútum típusa.

Példa: oszloptípus

```
CREATE TYPE AddrType AS (  
  street  CHAR(30),  
  city    CHAR(20),  
  zip     INT  
);  
CREATE TABLE Drinkers (  
  name    CHAR(30),  
  addr    AddrType,  
  favBeer BeerType  
);
```

Az *addr* és
favBeer attribútumok
értékei objektumok,
3 illetve
2 mezővel



Mező elérés problematikája az ORACLE-ben

- Egy objektum F mezőjét $A.F$ kifejezéssel elérhetjük, amelynek ez az értéke
- Azonban egy aliaszt kell használni, pl. rr , R relációra, annak A attribútumára mint pl. $rr.A.F$

Példa: Oracle-ben mezők elérése

- Rossz:

```
SELECT favBeer.name  
FROM Drinkers;
```

- Rossz :

```
SELECT Drinkers.favBeer.name  
FROM Drinkers;
```

- Jó:

```
SELECT dd.favBeer.name  
FROM Drinkers dd;
```

A REF-k (hivatkozások) követése: SQL-99 stílus

- $A \rightarrow B$ csak akkor értelmes ha:
 1. Ha A egy **REF T** típusú.
 2. A **T** típusú objektum mezője (komponense) B .
- Az A által hivatkozott, mutatott objektum B ***mezőjének*** értékét jelöli

Példa: REF-k (hivatkozások) követése

- Emlékezzünk rá, hogy az **Sells (Értékesítés)** egy olyan reláció egy olyan sortípussal ahol **MenuType(bar, beer, price)**, és ahol bar (kocsmá) és beer (sör) REF-ek, hivatkozások a **BarType** és **BeerType** típusú típusú objektumokra.

◆ Keresd meg a Joe által felszolgál söröket: A nyilat követve kapjuk meg a hivatkozott „*kocsmá*”-t és „*sör*”-t

```
SELECT ss.beer()->name  
FROM Sells ss  
WHERE ss.bar()->name = 'Joe"s Bar';
```

Először használjuk a generátor metódust, hogy hozzáférjünk a *kocsmá* és *sör* komponenshez

Oracle stílusban REF (hivatkozás) követése

- REF követése implicit a pontban.
- A REF-t nyomon követni: egy „elem” után egy pont, majd a megjelölt objektum mezőjének, amire hivatkozik, követésével kapjuk meg az értéket
- Példa:

```
SELECT ss.beer.name  
      FROM Sells ss  
      WHERE ss.bar.name = 'Joe''s Bar';
```

Oracle Deref művelete - motiváció

- Ha a Joe által értékesített sörökre mint sör objektumok halmazára van szükségünk, megpróbálhatjuk az alábbi:

```
SELECT ss.beer  
FROM Sells ss  
WHERE ss.bar.name = 'Joe''s Bar';
```

- Legális SQL, de *ss.beer* maga egy hivatkozás, ezért egy zagyvaság.

DEREF használata

- Ahhoz, hogy a **BeerType** objektumait láthassuk:

```
SELECT Deref(ss.beer)
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
```

- Egy ilyen értéket állít elő:

BeerType('Bud', 'Anheuser-Busch')

Metódusok – ORACLE szintaxis

- Az osztályok többek mint adatszerkezetek; lehetnek metódusaik.
- Tanulmányozni fogjuk az Oracle szintaxisát

Metódus definíció (Oracle)

- A metódusok deklarálhatjuk a CREATE TYPE-ban
- Definiálhatjuk a CREATE TYPE BODY utasításban
 - Használva a PL/SQL szintaxisát a metódusokra
 - SELF változó arra az objektumra vonatkozik, amelyre a metódust alkalmazni kívánjuk.

Példa: metódus deklaráció

Adjuk hozzá *priceInYen*-t, *MenuType*-hoz.

```
CREATE TYPE MenuType AS OBJECT (  
  bar      REF BarType,  
  beer     REF BeerType,  
  price     FLOAT,  
  MEMBER FUNCTION priceInYen(rate IN FLOAT) RETURN  
  FLOAT,  
  PRAGMA RESTRICT_REFERENCES(priceInYen, WNDS)  
);  
/
```

Oracle ezt nevezi metódusnak

Vagyis *priceInYen* nem fogja módosítani az adatbázis állapotát

Metódus definíció – Oracle stílusban

- ◆ A *create-body* utasítás formája:

```
CREATE TYPE BODY <type name> AS
```

```
  <method definitions = PL/SQL  procedure definitions, using  
    “MEMBER FUNCTION” in place of  
    “PROCEDURE”>
```

```
END;
```

```
/
```

Példa: Metódus definíció

```
CREATE TYPE BODY MenuType AS
  MEMBER FUNCTION
    priceInYen(rate FLOAT) RETURN FLOAT IS
      BEGIN
        RETURN rate * SELF.price;
      END;
END;
/
```

Az (IN)
nincs a „body”-ban,
csak a deklarációban

Csak akkor használjunk
zárójelet, ha legalább
egy argumentum van

Metódus használata

- ◆ Az objektum neve után legyen egy pont, majd a metódus neve, és végül az argumentumok, ha egyáltalán vannak.

- Példa:

```
SELECT ss.beer.name,  
       ss.priceInYen(114.0)  
FROM Sells ss  
WHERE ss.bar.name = 'Joe''s Bar';
```

Rendező metódusok: SQL-99

- Mindegyik T UDT két metódust definiálhat **EQUAL** és **LESSTHAN**.
 - Mindegyik metódus egy T típus argumentumot kap bemenetként és egy másik T típusú objektumra alkalmazza.
 - TRUE értéket ad vissza akkor és csak akkor ha a cél objektum = (vagy <) mint az az argumentumban szereplő objektum.
- Lehetővé teszi, hogy T típusú objektumokat hasonlítsunk össze =, <, >=, stb. segítségével a WHERE záradékban és a rendezésben (ORDER BY).

Rendező metódusok: Oracle

- Bármilyen UDT típusra bármelyik metódust *rendező metódusnak* deklarálhatjuk.
- A rendező metódusok visszatérő értéke <0 , $=0$, vagy >0 lehet, ahogy a SELF objektumhoz viszonyítva az argumentum értéke $<$, $=$, vagy $>$

Példa: Rendező metódusok deklarálás

◆ Rendezd a BarType objektumokat név szerint:

```
CREATE TYPE BarType AS OBJECT (  
  name    CHAR(20),  
  addr    CHAR(20),  
  ORDER MEMBER FUNCTION before(  
    bar2 IN BarType) RETURN INT,  
  PRAGMA RESTRICT_REFERENCES(before,  
    WNDS, RNDS, WNPS, RNPS)  
);  
/
```

Nincs adatbázis/csomag állapot változás. Egy „csomag” eljárások és változók gyűjteménye.

Példa: Rendező metódusok definiálás

```
CREATE TYPE BODY BarType AS
  ORDER MEMBER FUNCTION
    before(bar2 BarType) RETURN INT IS
  BEGIN
    IF SELF.name < bar2.name THEN RETURN -1;
    ELSIF SELF.name = bar2.name THEN RETURN 0;
    ELSE RETURN 1;
    END IF;
  END;
END;
/
```

Oracle beágyazott táblák

- Megengedi, hogy a sorok egyes komponensei teljes relációk legyenek.
- Ha T egy UDT, létrehozhatunk egy S típust, amelynek az értékei relációk, amelyeknek a sortípusa viszont T :

```
CREATE TYPE S AS TABLE OF T;
```

Példa: beágyazott tábla típusok létrehozása

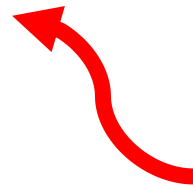
```
CREATE TYPE BeerType AS OBJECT (  
    name  CHAR(20),  
    kind  CHAR(10),  
    color CHAR(10)  
);  
  
/  
CREATE TYPE BeerTableType AS  
    TABLE OF BeerType;  
/  

```

Példa -- folytatása

- ◆ **BeerTableType-t** használjuk **Manfs** relációban, amelyik a sörök gyártóit tárolja, mindegyik gyártó egy sorban.

```
CREATE TABLE Manfs (  
    name          CHAR(30) ,  
    addr          CHAR(50) ,  
    beers beerTableType  
);
```



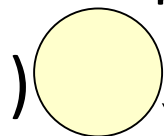
Ez így még nem lesz jó!
Ld. később a helyes szintaxist!

A beágyazott relációk eltárolása

- Oracle valójában nem tárolja el a beágyazott relációkat külön relációkként – még ha így is tűnik.
- Ehelyett, egy R reláció van, amelyben egy A attribútumra az összes beágyazott táblázatot és azok összes sorát eltárolja.
- Deklaráció a CREATE TABLE:
NESTED TABLE A STORE AS R

Példa: Beágyazott táblák tárolása

```
CREATE TABLE Manfs (  
    name    CHAR(30),  
    addr    CHAR(50),  
    beers   beerTableType
```



```
NESTED TABLE beers STORE AS BeerTable;
```



A pontosvessző (;) vessző használatára figyelni!

Beágyazott táblák lekérdezése

- Bármely beágyazott táblázat ugyanúgy jeleníthető meg, nyomtatható ki mint bármilyen más érték.
- Azonban ennek az alábbi két értéknek van két típuskonstruktor:
 1. A tábláknak-
 2. A soroknak a táblákban

Példa: Beágyazott táblák lekérdezése

- Anheuser-Busch söreit keressük ki:

```
SELECT beers FROM Manfs  
WHERE name = 'Anheuser-Busch';
```

- Egy értéket eredményez:

```
BeerTableType(  
  BeerType('Bud', 'lager', 'yellow'),  
  BeerType('Lite', 'malt', 'pale'),...  
)
```


Beágyazott táblán belüli lekérdezés

- Egy beágyazott táblát hagyományos relációvá lehet konvertálni a TABLE() alkalmazásával
- Ezt a relációt, ugyanúgy mint bármely másikat, a FROM záradékban lehet alkalmazni.

Példa: TABLE() használata

- ◆ Keresd meg Anheuser-Busch által gyártott „ale”-ket:

```
SELECT bb.name
```

```
FROM TABLE(
```

```
  SELECT beers
```

```
  FROM Manfs
```

```
  WHERE name = 'Anheuser-Busch'
```

```
) bb
```

```
WHERE bb.kind = 'ale';
```

Beágyazott tábla
Anheuser-Busch
sörökre

Alias a névnélküli
beágyazott táblára

Még egy példa

```
CREATE TYPE cim_t AS OBJECT (  
    utca          VARCHAR2(30),  
    varos         VARCHAR2(20),  
    ir_szam       CHAR(4) );  
  
/  
CREATE TYPE cim_tab IS TABLE OF cim_t;  
  
/  
CREATE TABLE vevok (  
    vevo_id       NUMBER,  
    cimek         cim_tab )  
NESTED TABLE cimek STORE AS vevo_cimek;
```

Még egy példa

```
INSERT INTO vevok VALUES (1,  
    cim_tab(  
        cim_t('Malom utca 2.', 'Varos', '9999'),  
        cim_t('Nagy utca 1.', 'Falu', '0000')  
    ) );  
  
INSERT INTO vevok VALUES (2,  
    cim_tab(  
        cim_t('Pajta utca 1.', 'Varos', '9999') ) );
```

Még egy példa

```
SELECT * FROM vevok;
```

```
vevo_id cimek
```

```
-----  
1 CIM_TAB(CIM_T('Malom utca 2.', 'Varos', '9999'),  
          CIM_T('Nagy utca 1.', 'Falu', '0000'))  
2 CIM_TAB(CIM_T('Pajta utca 1.', 'Varos', '9999'))
```

```
SELECT v.vevo_id, u.* FROM vevok v, TABLE(v.cimek) u;
```

```
vevo_id utca          varos ir_szam
```

```
-----  
1 Malom utca 2. Varos 9999  
1 Nagy utca 1.  Falu  0000  
2 Pajta utca 1. Varos 9999
```

Relációk beágyazott táblává alakítása

- Bármely reláció megfelelő számú attribútummal és azok illeszkedő adattípusaival egy beágyazott tábla értékei lehetnek.
- Használjuk a `CAST(MULTISET(...) AS <type>)` utasítást a reláción azért, hogy a helyes adattípussal rendelkező értékeivel egy beágyazott táblázattá alakítsuk.

Példa: CAST --- 1

- Tegyük fel, hogy a **Beers(beer, manf)** olyan reláció, hogy a **sör (beer)** egy **BeerType** típusú objektum és *manf* pedig egy string – a sör gyártója.
- Egy új sort akarunk beilleszteni a Manfs –ba , „Pete’s Brewing Co.” -t, mint (gyártó) nevet, és Pete által gyártott sörök halmazát.

Példa: CAST --- 2

```
INSERT INTO Manfs VALUES (
```

```
  'Pete''s', 'Palo Alto',
```

```
  CAST(
```

```
    MULTISET(
```

```
      SELECT bb.beer
```

```
      FROM Beers bb
```

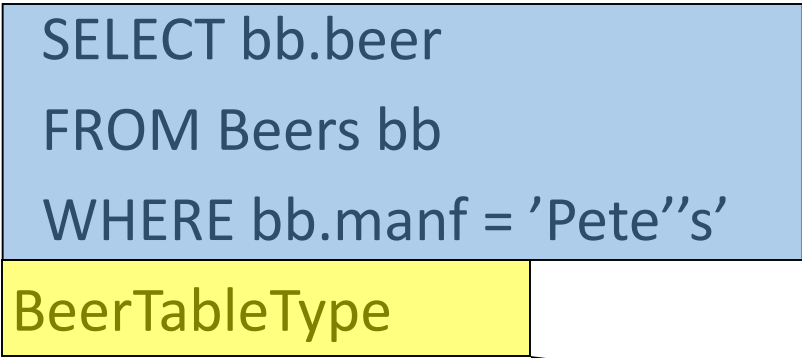
```
      WHERE bb.manf = 'Pete''s'
```

```
    ) AS BeerTableType
```

```
  )
```

```
);
```

Pete söreire
BeerType
objektumok
halmaza



Az objektumok halmazát
beágyazott relációvá alakítjuk