

# Objektum-relációs adatbázisok – 1. rész

Felhasználói típusok (User-Defined Types)

Objektum ID-k

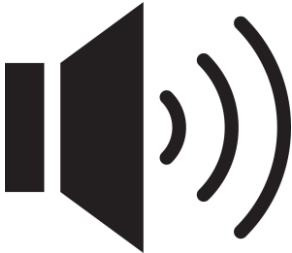





Beágyazott táblák (Nested Tables)

# Relációs és az O-O modell egyesítése

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

# Egy „kutya” a relációs modellben

<http://web.cs.wpi.edu/~cs561/s12/Lectures/2-3/OO.pdf> nyomán

		
	Blöki	
		4

# Egy „kutya” a O-O modellben

<http://web.cs.wpi.edu/~cs561/s12/Lectures/2-3/OO.pdf> nyomán



# 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 (*object-relational extension*) az O-O megközelítés több előnyös tulajdonságát megragadja, mégis megtartja a relációt mint az alapvető absztrakciós mechanizmust és adatszerkezetet

# Relációk → objektumrelációk

- A reláció alapvető fogalom marad, de...
- Attribútumokhoz szerkesztett strukturált típusok
- Metódusok (*methods*)
- Sorok azonosítói (*row IDs*)
- Hivatkozások (*references*)

# Beágyazott relációk (*nested relations*)

- Egy reláció attribútumának típusa nem csak atomi típus lehet, hanem pl. egy relációséma is
- Rekurzív definíció
  - Kiindulás: atomi típus
  - Indukció: reláció típusa séma egy vagy több attribútumnévvel és hozzátartozó típussal, az utóbbi lehet séma is
- Az atomi típusokat általában nem jelöljük külön, de a séma típusú attribútumot annak nevével és zárójelben a sémához tartozó attribútumok zárójelezett listájával adjuk meg
  - Pl.  $\text{Rel1}(\text{Attr1}, \text{Attr2}(\text{Attr2A}, \text{Attr2B}), \text{Attr3})$ , ahol  $\text{Attr2}(\text{Attr2A}, \text{Attr2B})$  egy relációséma

# Beágyazott relációk – példa

- Oktatókhoz tervezzünk egy beágyazott relációsémát!
- Tartalmaz **név** és **TAJ szám** attribútumokat, amelyek atomi típusúak
- Tartalmaz egy **címek** attribútumot, amely relációséma típusú
  - Két attribútuma van: **város** és **közterület**
- Tartalmaz egy **tárgyak** attribútumot is, amely szintén relációséma típusú
  - Két attribútuma van: **tárgynév** és **kód**
- Tehát az **Oktatók** sémája a következő:  
**Oktatók(név, TAJ szám, címek(város, közterület), tárgyak(tárgynév, kód))**



# Beágyazott relációk – példa

Név	TAJ szám	Címek		Tárgyak	
Nagy Morgána	123 456 789	Város	Közterület	Tárgynév	Kód
		Pécs	Petőfi utca 1.	Adatbázisok	EGY-AB001
				Hálózatok	EGY-Halo001
				Programozás	EGY-Prog001
Kovács István	987 654 321	Város	Közterület	Tárgynév	Kód
		Budapest	Rákóczi út 1.	Adatbázisok	EGY-AB001
		Debrecen	Kossuth tér 1.	Hálózatok	EGY-Halo001
				Programozás	EGY-Prog001

# Hivatkozások

- Az előző példában redundancia van (pl. Adatbázisok tárgynál)
- Kellene, hogy egy tárgy csak egyszer szerepeljen az összes **tárgyak** reláció összes sorában
- Megoldás: hivatkozás lehetősége
- Újabb indukciós szabály: egy attribútum típusa egy adott sémájú sorra történő hivatkozás vagy adott sémájú sorokra történő hivatkozási halmaz is
- Jelölés:
  - ha A attribútum típusa egy R relációsémájú sorra történő hivatkozás  $\rightarrow A(*R)$  alakban jelenik meg
  - ha A attribútum típusa egy R séma soraira vonatkozó hivatkozási halmaz  $\rightarrow A(\{*R\})$  alakban jelenik meg

# Hivatkozás – példa

- Oktatóknál látott redundancia helyesbítésére két relációt használunk:
- Külön a Tárgyakra létrehozunk egy relációsémát:

**Tárgyak(tárgynév, kód)**

- Az **Oktatók** sémája pedig a következő lesz:

**Oktatók(név, TAJ szám, címek(város, közterület), tárgyak({\*Tárgyak}))**

# Hivatkozás – példa

Név	TAJ szám	Címek		Tárgyak
Nagy Morgána	123 456 789	Város	Közterület	
		Pécs	Petőfi utca 1.	
Kovács István	987 654 321	Város	Közterület	
		Budapest	Rákóczi út 1.	
		Debrecen	Kossuth tér 1.	

Tárgynév	Kód
Adatbázisok	EGY-AB001
Hálózatok	EGY-Halo001
Programozás	EGY-Prog001

# 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 (*user-defined data type, UDT*), egy O-O osztály definíciója, amely egy adatszerkezet és metódusai.
  - Azonos „típusú” objektumok egy osztályt definiálnak
  - Viselkedés: metódusok halmazával kifejezve, amelyek az osztályhoz tartozó objektumokon hajthatóak végre

# Felhasználó által definiált adattípus

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

# Felhasználó által definiált adattípus

```
CREATE TYPE <típusnév> AS (  
    <attribútum-típus párok listája>  
);
```

- **ORACLE-ben:** CREATE TYPE <típusnév>  
AS OBJECT (<attribútum-típus párok  
listája>);



## Példa: UDT létrehozásra

```
CREATE TYPE TeázóTípus AS (  
    név      CHAR(20),  
    cím      CHAR(20)  
);
```

```
CREATE TYPE TeaTípus AS (  
    név      CHAR(20),  
    gyártó    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 (*pointer*) egy  $T$  típusú objektumra.
- Ezt „objektum azonosítónak” (*object ID*, *OID*) is hívják O-O rendszerekben.
- Gyakorlatilag az *OID* élete végéig azonosít egy objektumot, függetlenül a komponenseinek/mezőinek értékeitől

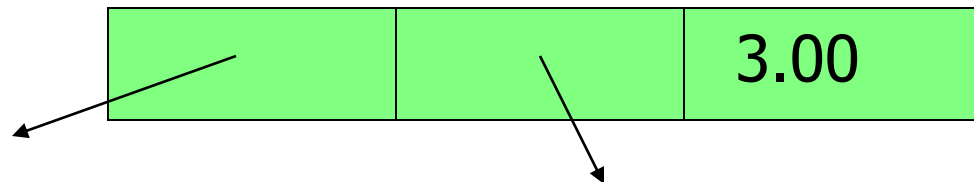
# Hivatkozások

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

# Példa: REF

```
CREATE TYPE FelszoigáltTípus AS (  
    teázó      REF TeázóTípus,  
    tea        REF TeaTípus,  
    ár         FLOAT  
);
```

- FelszoigáltTípus objektum valahogy így néz ki:



Egy *TeázóTípus*  
objektumra hivatkozás

Egy *TeaTípus*  
objektumra hivatkozás

# UDT-k, mint sortípusok

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

- Szintaxis:

```
CREATE TABLE <táblanév> OF  
    <típusnév>;
```

## Példa: Egy reláció készítése

```
CREATE TABLE Teázók OF TeázóTípus;
```

```
CREATE TABLE Teák OF TeaTípus;
```

```
CREATE TABLE Felszolgal OF FelszolgalTípus;
```

# Sortípusú relációk értékei

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



# Példa: típuskonstruktor

- Lekérdezés

```
SELECT * FROM Teázók;
```

- Eredmény sora:

TeázóTípus('Joe''s Teahouse', '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.név, bb.cím  
FROM Teázók 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 (*value access*)

- Az előbbi lekérdezés SQL-99-ben:

```
SELECT bb.név(), bb.cím()  
FROM Teázók 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 Teázók VALUES (  
    TeázóTípus('Joe' 's Teahouse', '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, *újTeázó*.
- A *mutátor* metódusok megváltoztatják a név és cím komponenst.

```
SET újTeázó = TeázóTípus();
```

```
    újTeázó.név('Joe''s Teahouse');
```

```
    újTeázó.cím('Maple St.');
```

```
INSERT INTO Teázók VALUES(újTeázó);
```

## UDT-k, mint oszloptípusok (*column types*)

- 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 TeaTípus AS (  
    név      CHAR(20),  
    gyártó   CHAR(20)  
);
```

```
CREATE TYPE CímTípus AS (  
    utca      CHAR(30),  
    város     CHAR(20),  
    ir.szám   INT  
);  
CREATE TABLE Vendégek (  
    név        CHAR(30),  
    cím        CímTípus,  
    kedvencTea TeaTípus  
);
```

Az *cím* és  
*kedvencTea* attribútumok  
értékei objektumok,  
3 illetve  
2 mezővel

