Objektumorientált adatbázisok

- Az objektumorientált programozás segítségével, könnyebben, természetesebben modellezhetjük a valós világot, jobban szervezhetjük az adatainkat.
- Az adatbázisok világa is elmozdult az objektumorientáltság felé.

Két irányban történt a fejlesztés:

- Objektumorientált adatbázisok (OOABKR), egy objektumorientált programozási nyelvet kibővítették az adatbázis-kezelő rendszerek képességeivel.
- Objektum-relációs rendszerek (ORABKR), létező relációs adatbázis-kezelő rendszereket kibővítették az objektumorientáltság fogalmaival.

Objektumorientált adatbázisok (**OOABKR**), egy objektumorientált programozási nyelvet kibővítették az adatbázis-kezelő rendszerek képességeivel.

– Adatok permamens tárolása (objektumokat perzisztenssé kell tenni) egy sor extra programozási feladatot kíván meg a programozótól.
– Függetlenségi szintek. Az ABKR-ek jellemzője, hogy a felhasználónak nem kell ismernie az adatok fizikai tárolását. A felhasználó egy emberközelibb kezelő nyelv segítségével kommunikálhat az ABKR-rel, amely majd lefordítja a kapott magas szintű utasítást a végrehajtáshoz szükséges fizikai szintre.
– Konkurrens adatkezelés biztosítása, adatvédelem, helyreállítás hiba esetén. Az ABKR egyidejűleg több párhuzamos kérés kiszolgálására alkalmas mechanizmussal rendelkezik.

– Ad-hoc lekérdező nyelv. Az ABKR-ek fontos eleme egy rugalmas lekérdező nyelv, amellyel egyszerűen és hatékonyan lehet információt kinyerni a rendszerből.

Ez a megoldás rendszerint egy speciális, saját könyvtár létrehozását jelenti, amely a permanens adatkezeléshez szükséges adattárolási funkciókat valósítják meg. Ekkor a fejlesztés menete megegyezik a normál programok készítésének menetével, azzal a különbséggel, hogy az adatok tárolását egy csatolt függvény gyűjtemény felhasználásával oldják meg. E rendszerekben sokszor hiányzik a védelem, a tranzakciókezelés, az optimalizálás és a rugalmas lekérdező nyelv is.

Objektum-relációs rendszerek (**ORABKR**), létező relációs adatbázis-kezelő rendszereket kibővítették az objektumorientált fogalmakkal:

- Komplex típusokkal
- Osztály fogalmával
- Örökléssel

- Ezen fogalmak jellemzők úgy az objektumorientált, mint az objektum-relációs rendszerekre is, a továbbiakban részletezni fogjuk
- Ennek a megoldásnak az előnye, hogy az ABKR szolgáltatások már rendelkezésre állnak, s a módosításnál, az új elemek bevonásánál is támaszkodhatnak az ABKR fejlesztők a korábbi szolgáltatásokra.

Az első “tiszta” objektumorientált adatbázis-kezelő rendszerek (OOABKR) a 80-as évek végén jelentek meg. Ma már sok OOABKR van a piacon, de nem olyan sikeresek, mint várható lett volna.

A ma létező OOABKR pozitív tulajdonságai:

– igen népszerű, nagyon sok helyen foglalkoznak objektumorientált adatbázis rendszer fejlesztésével;
– nagyon sok egymástól független, párhuzamosan dolgozó műhely alakult ki az OOABKR kidolgozására és fejlesztésére;
– biztosítja az objektumorientált elvek megvalósítását;
– szoros kapcsolódást biztosít az egyéb objektumorientált fejlesztő eszközökhöz, fejlesztő nyelvekhez.

Az OOABKR **gyenge** pontjai:

– az objektumorientált adatmodell nem rendelkezik olyan elméleti megalapozottsággal, mint a relációs modell.
– az első OOABKR-ek teljesítményben, megbízhatóságban lényegesen alacsonyabb szinten álltak, mint a relációs ABKR rendszerek.
– objektumorientált adatmodellel szemben érezni bizonyos ellenállást.

Chris Date (a relációs modell támogatója) véleménye:

- OOABKR - **rekordorientált** megközelítést, lényeges visszalépést jelent a relációs modellben megszokott halmazorientáltsághoz képest.
- Az OOABKR-ek nincs megfelelő mechanizmusuk az adatok **helyességének ellenőrzésére** vonatkozólag, nem lehet deklaratív integritási feltételeket megadni. A metódusok segítségével kell ezeket a problémákat megoldani.
- A lekérdező nyelvek által igényelt nyíltság (SQL SELECT utasításhoz hasonló) összeilleszthető-e az objektumok zártságának az elvével.

Elméleti kifogások mellett a gyakorlati szakemberek az OOABKR-ek megvalósításának a kérdőjeleit, mint hogy

– megfelelő lesz-e az OOABKR hatékonysága;
– tud-e majd a rendszer nagy adatmennyiséget is kezelni;
– mennyire megbízhatóak az OOABKR-t fejlesztő cégek;
– mennyire szabványosak az egyes OOABKR-ek.

Relációs kontra objektumorientált adatmodell vitában

- nem az fő kérdés, hogy kell-e objektumokat tárolni az adatbázisokban, mivel ennek szükségességében mindenki egyetért
- hogyan célszerű megvalósítani az objektumok tárolását.

Két fő tábor :

- OOABKR
- ORABKR

A típusrendszer

Az objektumorientált nyelvek a típusok széles választékát kínálják. Az *alaptípusok* az egészek, valós számok, logikai értékek és karakterláncok. A típus-konstruktorok segítségével lehetőségünk van az alaptípusokból újabb típusok létrehozására. Segítségükkel például a következő összetett típusok hozhatók létre:

- *Rekordstruktúrák*
- *Kollekciótípusok*
- *Hivatkozástípusok*

– *Rekordstruktúrák*. Ha adottak a T_1, T_1, … T_n típusok és az m_1, m_2,…, m_n mezőnevek, ahol az m_i mező T_i típusú, létrehozzuk a

RECORDOF (m_1, m_2,…, m_n)

típust, mely n komponensből álló rekord és típusa:

[m_1:T_1, m_2 : T_2,…, m_n : T_n]

– *Kollekciótípusok*: Legyen T egy típus, új típusokat hozhatunk létre *kollekcióoperátor* alkalmazásával. Kollekcióoperátorok a **tömbök**, **listák** és **halmazok**.

Hivatkozástípusok: Egy *T* típusra való hivatkozás egy olyan típus, amelynek értékei segítségével megkereshetünk egy *T* típusú értéket.

- A fogalom hasonló a C++-beli mutatóhoz
- Az adatbázisrendszerekben viszont, ahol az adatok jó néhány lemezen, sokszor különböző számítógépeken helyezkednek el a hivatkozás sokkal bonyolultabb: tartalmazhatja a számítógép nevét, a lemezegység számát, azon belül egy blokk azonosítóját és a blokkban egy poziciót, ahol a keresett érték tárolva van.

A rekordstrukturát és kollekcióoperátorokat egymásba ágyazottan többször is alkalmazhatjuk, így összetett típusokat hozhatunk létre.

Osztályok és objektumok

- osztály – absztrakt adattípus, metódusok (függvény vagy eljárás) áll.
- az osztály minden objektuma azonos típusú

Objektumazonosító

- minden objektumnak van egy objektumazonosítója (OID).
- egyedi minden objektumra nézve.
- egy mutató, amelyik az objektumra mutat.
- érvényes kell legyen mindaddig amíg az objektum létezik

Absztrakt adattípusok

- absztrakt adattípus: megakadályozza az objektumaihoz való tetszőleges hozzáférést, azaz csak az osztály metódusain keresztül biztosít hozzáférést.
- csak az osztály metódusai módosíthatják közvetlenül az objektumokat, így az csak olyan módon lehet módosítani, ahogy az megvolt tervezve.

Osztályhierarchiák

Az alosztály örökli a szuperosztály összes tulajdonságát és metódusait, és deklarálhatunk plusz tulajdonságokat is.

Az alosztályoknak is lehet alosztálya, egy alosztályt származtathatunk több szuperosztályból, így az osztályok egy hierarchiájához jutunk.

Tervezés ODL (Object Definition Language) segítségével

- ODL: egy javasolt szabvány, segítségével adatbázisok struktúráját specifikálhatjuk objektumorientált terminológiával.
- **célja**, támogatja az adatbázisok objektumorientált tervezését, és utána ennek átalakítását OOABKR-ek deklarációiba.
- **nem** ad lehetőséget az adatbázis tartalmának megadására, az adatok kezelésére, lekérdezésekre.
- egy adatdefiníciós nyelv, ugyanúgy, ahogy az SQL-ben a DDL.

Objektumorientált tervezés

- a valós világot objektumok segítségével modellezzük.
- Példa objektumok: a személyeket, épületeket, kurzusokat, stb.
- Az objektumokról feltételezzük, hogy rendelkeznek egy sajátos azonosítóval (OID), ennek segítségével különböztetjük meg az objektumokat egymástól.
- Az objektumokat osztályokba csoportosíthatjuk, azzal a megkötéssel, hogy egy osztályon belül minden objektum azonos típusú, s ugyanazok a metódusok vonatkoznak rájuk, mint amelyek az osztály definiálásánál adottak

Amikor specifikálunk az ODL-ben egy osztályt, a következő három jellemzőjét adjuk meg:

- Attribútumok: ezek az objektum tulajdonságai, s típussal rendelkeznek.
- Kapcsolatok: típusa valamilyen osztály egy objektumára való hivatkozás, vagy ilyen hivatkozások gyűjteménye.
- Metódusok: ezek függvények, alkalmazhatjuk őket az osztály objektumaira.

Osztály deklarációja

Az ODL-ben az osztály deklarációja a következőképpen néz ki:

```
interface  <név> {
        < jellemzők listája >
}
```

<név> - az osztály neve
jellemzők: attribútumok, kapcsolatok és metódusok.

Attribútumok az ODL-ben

Az attribútum deklarálásakor az *attribute* kulcsszót használjuk. Például deklaráljuk a Diák osztályt:

```
interface Diák {
    attribute integer beiktatásiSzám;
    attribute string név;
    attribute struct cím
        {string város, string utca} lakcím;
};
```

A lakcím attribútum típusa sortípus, mely egy rekordstruktúra. Ennek a típusnak a neve cím és két karakterlánc típusú mezője van: a város és az utca. ODL-ben a struct kulcsszót használjuk a rekordstruktúra deklarációjára.

Kapcsolatok ODL-ben

Példa: szeretnénk a `Diák` osztályba felvenni egy jellemzőt, ami a tantárgyak egy halmaza.

A `Diák` osztály objektumainak nincsenek közös jellemvonásai a tantárgyakkal, ezért értelmezünk egy `Tantárgy` osztályt:

```
interface Tantárgy {
    attribute string név;
    attribute string leírás;
    attribute integer kreditszám;
};
```

- a tantárgyak nem lehet egy attribútum, ugyanis ennek a típusa nem lehet egy osztály.

A diák tantárgyainak a halmaza egy kapcsolat lesz.

```
interface Diák {
    attribute integer beiktatásiSzám;
    attribute string név;
    attribute struct cím
       {string város, string utca} lakcím;
    relationship set<Tantárgy> tantárgyai;
};
```

kapcsolat: a Diák osztály minden objektumának van egy hivatkozása a Tantárgy osztály objektumainak a halmazára.

`set<Tantárgy>` mutatók egy listája, és mindegyik mutató egy Tantárgy objektumra mutat.

– Más kapcsolat: egy objektumhoz csak egy objektumot rendel hozzá.
Példa: Diák osztályba a kedvenc tantárgyát szeretnénk eltárolni:

```
interface Diák {
    attribute integer beiktatásiSzám;
    attribute string név;
    attribute struct cím
       {string város, string utca} lakcím;
    relationship set<Tantárgy> tantárgyai;
    relationship Tantárgy kedvenctantárgy;
};
```

Inverz kapcsolatok

– ha szeretnénk tudni, hogy a tantárgyakat kik tanulják.

```
interface Tantárgy {
    attribute string név;
    attribute string leírás;
    attribute integer kreditszám;
    relationship set<Diák> tanulja;
};
```

Fontos helyesség szempontjából:

– ha az X diák benne van az Y tantárgyhoz tartozó tanulja halmazban akkor kötelezően benne kell legyen az Y tantárgy a X diák tantárgyai halmazában.

```
interface Diák {
    attribute integer beiktatásiSzám;
    attribute string név;
    attribute struct cím
        {string város, string utca} lakcím;
    relationship set<Tantárgy> tantárgyai
        inverse Tantárgy::tanulja;
};
interface Tantárgy {
    attribute string név;
    attribute string leírás;
    attribute integer kreditszám;
    relationship set<Diák> tanulja
        inverse Diák::tantárgyai;
};
```

– Az ODL megköveteli, hogy a kapcsolatoknak legyen inverze.

Kapcsolattípusok ODL-ben

egyed/kapcsolat adatmodell esetén megismert

- egy-az egy-hez (1:1)
- egy a sokhoz (1:*n*)
- sok a sokhoz (*n*:*m*)

ODL-ben azzal specifikáljuk a **sok**at, hogy használunk kollekció típusú operátorokat, mint a **set**.

```
interface Diák {
   attribute integer beiktatásiSzám;
   attribute string név;
   attribute struct cím
      {string város, string utca} lakcím;
relationship set<Tantárgy> tantárgyai
  inverse Tantárgy::tanulja;
relationship Tantárgy kedvenctantárgy
  inverse Tantárgy::kinekkedvence;
relationship Csoport tagja
   inverse Csoport::diákjai;
relationship Csoport felelős
   inverse Csoport::csoportfelelős;
};
```

```
interface Tantárgy {
    attribute string név;
    attribute string leírás;
    attribute integer kreditszám;
    relationship set<Diák> tanulja
        inverse Diák::tantárgyai;
    relationship set<Diák> kinekkedvence
        inverse Diak::kedvenctantárgy;
};
interface Csoport {
    attribute string csoportKód;
    relationship set<Diák> diákjai
        inverse Diák::tagja;
    relationship Diák csoportfelelős
        inverse Diák::felelős;
};
```

A példa kapcsolattípusai:

- *n*:*m* kapcsolat:
  - Diák és Tantárgy tantárgyai, illetve tanulja.

- 1:*n* kapcsolat:
  - Csoport és Diák osztály közötti diákjai, illetve tagja
  - Tantárgy és Diák: kinekkedvence, illetve kedvenctantárgy

- 1:1 kapcsolat:
  - Csoport és Diák csoportfelelős, illetve felelős

Típusok ODL-ben

- hasonló más programozási nyelvek típusrendszeréhez.
- alaptípusokból épül fel, és ezekből **összetett** típusokat építhetünk.

Az ODL alaptípusai a következők:

1. *Atomi típusok*: integer, float, character, string, boolean és enum. Az enum felsorolás típus.

2. *Interfész típusok*: például a Diák és Tantárgy osztályok, melyek struktúrákat reprezentáló típusok.

Típuskonstruáló utasításokkal az alaptípusok strukturált típusokba kombinálhatóak. Ezek a következők:

1. *Halmaz.* Ha *T* egy típus, akkor Set<T> jelöli azt a típust melynek értéke *T* típusú elemek halmaza.
2. *Multihalmaz.* Ha *T* egy típus, akkor Bag<T> jelöli azt a típust melynek értéke *T* típusú elemek multihalmaza. A multihalmaz megengedi, hogy egy elem többször is előforduljon benne. Például {3, 4, 3} multihalmaz.
3. *Lista*. Ha *T* egy típus akkor List<T> jelöli azt a típust melynek értéke *T* típusú elemek véges listája.
4. *Tömb*. Ha *T* egy típus és *i* egy egész szám, akkor Array<T,i> jelöli azt a típust, amelynek az értéke egy *i* dimenziójú tömb, és az elemek *T* típusúak. Például array<integer,10> egy 10 dimenziójú és elemei integer típusúak.

5. *Struktúra*. Ha T_1, T_2, ..., T_n típusok, F_1, F_2, ..., F_n mezőnevek, akkor

`Struct N {`T_1 F_1`,` T_2 F_2`, ...,` T_n F_n`}`

jelöli azt az *N* nevű típust, amely egy *n* mezőből álló struktúra.

Az első négy típust *kollekciótípusnak* nevezzük.

Attribútum típusa lehet:

- atomi típus
- struktúra, amely mezőinek típusa szintén típus, azaz lehet atomi vagy összetett típus. Az atomi típusra vagy a struktúrára kollekciótípust vagy struktúraképzést alkalmazhatunk.

Kapcsolat típusa lehet:

- interfész típus
- kollekciótípus alkalmazva egy interfész típusra.

Alosztályok az ODL-ben

– hasznos az osztályt alosztályokra bontani, ha az osztály rendelkezik olyan objektumokkal, amelyeknek olyan tulajdonságaik vannak, amelyekkel az osztály más objektumai nem rendelkeznek.

Példa: a diákok között vannak olyan diákok, akik kapnak ösztöndíjat, és vannak olyanok akik külföldről jöttek. Számukra definiálhatunk a Diák osztály egy alosztályát.

```
interface ösztöndíjasdiák: Diák {
 attribute float ösztöndíj;
};
interface külföldidiák: Diák {
 attribute string ország;
};
```

– Az alosztály örökli a szuperosztály összes tulajdonságát, a szuperosztály összes attribútuma, metódusa és kapcsolata automatikusan attribútuma, metódusa és kapcsolata lesz az alosztálynak is.

Többszörös öröklődés az ODL-ben

Példa: ha egy diák külföldi és van ösztöndíja egy újabb osztály:

```
interface külföldiösztöndíjas: külföldidiák,
ösztöndíjasdiák{ };
```

– külföldiösztöndíjas objektumnak megvan minden tulajdonsága amivel a külföldidiák és ösztöndíjasdiák alosztályok rendelkeznek.

– előfordulhat, hogy két szuperosztálynak van egy *ugyanolyan nevű*, de *más típusú attribútuma*.
– az alosztályban ennek az attribútumnak a típusa tisztázatlan.

A következő módszerekkel lehet a típusát tisztázni:

1. Specifikálhatjuk, hogy a két definíció közül melyik kerüljön az alosztályba.

2. Az egyik szuperosztályban megváltoztatjuk az attribútum nevét.

3. Újra definiálhatjuk az alosztályban az attribútumot.

Kulcsok deklarálása ODL-ben

Egy $\boldsymbol{K}$ attribútum halmaz egy O osztály **kulcs**a, ha bármely két O_1 és O_2 különböző objektumok esetén a két objektum értéke nem egyezik meg a K attribútumhalmazán.

```
interface Diák
    (extent Diákok
     key beiktatásiSzám)
{
    attribute integer beiktatásiSzám;
    attribute string név;
    attribute struct cím
       {string város, string utca} lakcím;
    relationship set<Tantárgy> tantárgyai
```

```
    inverse Tantárgy::tanulja;
  relationship Tantárgy kedvenctantárgy
    inverse Tantárgy::kinekkedvence;
   relationship Csoport tagja
      inverse Csoport::diákjai;
   relationship Csoport felelős
      inverse Csoport::csoportfelelős;
};
```

Osztályhoz tartozó objektumkészlet

- **objektumkészlet**: az osztály aktuális objektumainak a halmazát jelöli.
- minden ODL osztályhoz deklarálható egy objektumkészlet az **extent** kulcsszó segítségével
- egy osztály objektumkészlete megfelel a relációs adatmodell esetén a relációnak (**tábl**ának).
- a Diák osztály deklaráció már az objektumkészlet deklarálását is tartalmazza: extent Diákok
- az OQL lekérdezések az objektumkészletre vonatkoznak.

Metódusok deklarálása ODL-ben

- ODL-ben az osztály (interface) deklarálásában metódusokat is megadhatunk, melyek az adott osztály objektumaira alkalmazhatók.

- A paraméterek típusai:
  - in (érték szerinti paraméter átadás)
  - out (hivatkozás szerinti paraméter átadás)
  - inout (hivatkozás szerinti paraméter átadás)

- A metódusok kiválthatnak kivételeket, melyek egy abnormális állapotot jeleznek, például nullával való osztás.

```
interface Diák
    (extent Diákok
     key beiktatásiSzám)
{
    attribute integer beiktatásiSzám;
    attribute string név;
   attribute string születésiDátum;
    attribute struct cím
       {string város, string utca} lakcím;
    relationship set<Tantárgy> tantárgyai
     inverse Tantárgy::tanulja;
   relationship Tantárgy kedvenctantárgy
     inverse Tantárgy::kinekkedvence;
    relationship Csoport tagja
```

```
        inverse Csoport::diákjai;
    relationship Csoport felelős
        inverse Csoport::csoportfelelős;
    integer kora() raises (nincsSzülDatum);
    hasonlóErdeklődésűDiákok(out Set<String>);
};
```

- kiegészítettük a Diák osztály deklarációját a diák születési dátumával, mint attribútum és egy kora() nevű metódussal, mely megadja a diák korát. A metódus kivételt vált ki, ha a születésiDátum attribútumnak nincs értéke.
- hasonlóErdeklődésűDiákok metódus a kimenő paraméter értékekénk megadja azoknak a diákoknak a nevét, akiknek ugyanaz a kedvenc tantárgya, mint annak a Diák objektumnak, mely meghívta a metódust.