Adatbázisok elmélete 2. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/˜kiskat

2004

Adatbáziskezelő rendszerek története

Ősei a file-kezelők; ezek nem teljesítik ugyan azokat az elvárásokat, amiket a DBMS-sel szemben támasztunk, de sok a hasonlóság: sok adat, hosszú élettartam. Viszont primitív a lekérdezés (csak a file-hierarchiában lehet mozogni), nincs sémadefiníció (csak könyvtárszerkezet), nincs védelem rendszerhibák esetére, többfelhasználós működés sincs támogatva.

Első rendszerek

Jellemzők: sok kis adat, gyakori, de kevés adatot érintő lekérdezések, módosítások.

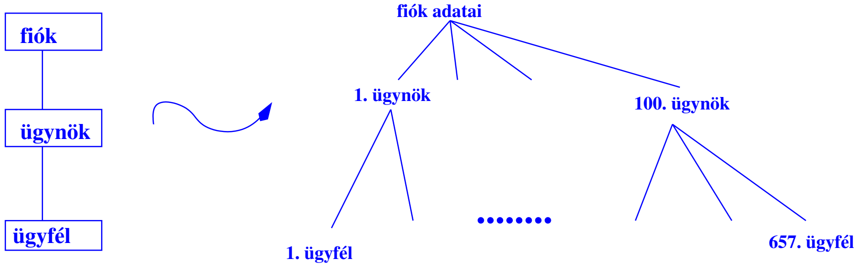
- *Repülőgépes helyfoglalás*
  Adatelemek: indulás, érkezés, honnan indul, hova érkezik, ár, darabszám, utas neve...
  Lekérdezések: van-e még hely, mennyi az ára, mikor indul a gép
  Módosítások: új utas bevitele, helyfoglalás
  Párhuzamosság: egyszerre több jegyeladás és lekérdezés is mehet
  Védelem: helyfoglalás nem veszhet el
- *Banki rendszerek*
  Adatelemek: ügyfelek adatai, szamlák adatai, jogosultságok...
  Lekérdezések: egyenlegek
  Módosítások: pénzmozgások
  Párhuzamosság, biztonság fontos/megoldva valahogy.

- *Vállalati rendszerek*
  Ügyfelek, eladások, szerződések adatai, kimutatások készítése, új szerződések bevitele.

Korai modellek

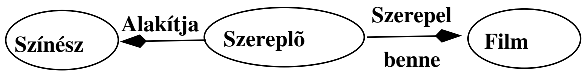
Közös jellemzők: a fogalmi keret tükrözi a tárolást

- Hierarchikus adatmodell
  Jó ott, ahol a reprezentálandó adatokban valódi hierarchia van, például biztosítós példa:



  Adatnyilvántartás: fában, ami a hierarchiát tükrözi, a gyökér szerint rendezetten tárolva
  $\Longrightarrow$ a lekérdezés és módosítás, illetve az adatok elérése csak a fa ismeretében lehetséges
- Hálós modell

Irányított gráffal adjuk meg az adatok közötti logikai összefüggéseket, a csúcsok a rekordtípusok, a nyilak a kapcsolatok.



Mindkét modell hátránya: nincs magas szintű lekérdezés, bármilyen hozzáféréshez a tárolás pontos ismerete szükséges

Relációs adatmodell

- Jelenleg a legelterjedtebb modell
- E.F. Codd 1970-es cikkén alapul
- Fő elv: az adatbázist alkossák táblák (relációk)
- Előnye a hierarchikus és hálós modellel szemben:
  - ⋆ magas szintű lekérdezés, a tárolási struktúra ismerete nélkül
  - ⋆ jól átlátható, mégis pontos, elméleti háttere is van
  - ⋆ a relációk mögött lehet bonyolult adatszerkezet is, de azt nem kell ismerni a működtetéshez

Ízelítő

Táblázat, reláció = fogalmi keret, egy-egy sor = egy-egy tárolandó adategyüttes.

Termelő(név, cím, termék, ár) tábla esetén:

név	**cím**	**termék**	**ár**
X. Kft	Sümeg	Kinder–tojás	127 Ft
•••••	•••••	•••••	•••••

Lekérdezés: egyszerű, de hatékony, nem kell ismerni, hogy mi hogyan tárolódik. Pl. SQL-ben egy lekérdezés:

SELECT ár, név FROM termelő
WHERE termék="Zizi"

Ez megkeresi az öszes olyan (ár, termelő) párt, ami a "Zizi"-hez tartozik.

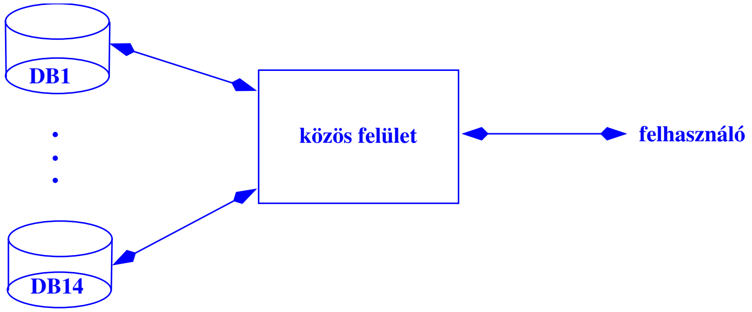
Jelenlegi rendszerek jellemzői

- főleg relációs modell, modellezésre pedig E/K diagram
- egyre kisebb rendszerek (DBMS-ek PC-re)
- nagy adatbázisok (egyre hosszabb idejű tárolás, illetve képek, hangok, multimédiás cuccok) $\Longrightarrow$ harmadlagos tárolás CD-n
- párhuzamos feldolgozás

Jövőbeni technológiák (részben már létezők)

- *objektumos adatbázisrendszerek*: ODL-es tervezés, szokásos objektumos megközelítés, összetett típusok (jól leírják a modellezni kívánt világot)
- *megszorítások, triggerek*: aktív elemek (ha valami feltétel teljesül $\Longrightarrow$ beindul valami folyamat a rendszerben).
  - ⋆ megszorítások: előre megadott feltételeknek mindig teljesülniük kell. Ha valamelyik sérülne: cselekvés,pl. letiltás.
  - ⋆ triggerek: kódrészlet, ha valami adott helyzet bekövetkezik, akkor automatikusan kiváltódik valami esemény.
- *multimédiás adatok*: kép, hang, szöveg
  - ⋆ sokkal nagyobb adatok
  - ⋆ egyszerűbb műveletek is nehezek (pl. összehasonlítás), illetve új műveletek megjelenése
  - ⋆ továbbítás problémája (nem egyszerre, hanem adagokban)
- *adattárházak*: cél az adathalmazok egységesítése. Sokféle adat, sok helyen, ugyanolyan vagy hasonló dolgokról, de különféle tárolási struktúrában. Egységesen akarjuk látni az adatokat (webes katalógus, egységes vállalati nyilvántartás).
  Megoldás az adattárház: átalakított, különböző DB-kból származó adatok "közös nevezőre" hozása.

Nem kell lecserélni a kis adatbázisokat, hanem csak föléjük építünk egy struktúrát:



- *adatbányászat*: adatok között levô érdekes, szokatlan összefüggések keresése. Pl. aki fiatal férfi és ....-t vásárol, az vásárol ....-t is.

Adatmodellezés

- Célja: a modellezendő valóságdarabhoz adatbázisséma létrehozása.
- Elvárás: jól írja le a valóságot, könnyű legyen a gyakori kérdéseket és módosításokat megtenni
- Részei:
  1. Terv készítése (nagyon fontos rész, ha rossz tervet csinálunk, később nehéz módosítani) valamilyen modellező eszköz/nyelv segítségével (E/K diagram, ODL-es megadás).
  2. A terv átalakítása formálisabb leírássá (tipikusan E/K-ból relációs megadás).
  3. Az adatbázisséma formális megadása a rendszer által kívánt DDL-en (ez az átalakítás már viszonylag automatikusan megy, a DDL persze rendszerfüggő).

Mi most az első lépéssel foglalkozunk, a tervezéssel, később lesz majd még arról szó, hogy hogyan kell a tervet átírni relációs sémára, aztán pedig az SQL DDL-jére.

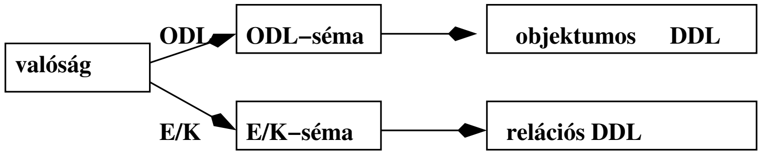
Adatmodellező eszközök

Egy adatmodellező eszköz egy többé-kevésbé formális jelölésrendszer, adatok és a köztük levő kapcsolatok megadására. (ODL inkább formális, E/K kevésbé).

Alapfogalmak:

- *adatok*, pl. pilóta, utas, járat
- *kapcsolatok*, pl. járat utasai, személyzete
- *műveletek,* már ahol van, vannak modellek, amiknek vannak saját műveleteik, amiket könnyű megvalósítani.

Tipikus használat:



Az E/K-relációs séma-relációs DDL út a hagyományosabb.

ODL alapelvei

Cél: objektumos szemléletű DB tervezése, az adatbázis struktúrájának megadása objektumos terminológiával. CORBA része, objektumos programozási nyelvekhez jól passzol. Az ODL-es tervet könnyű objektumos DDL-be transzformálni (relációsra viszont nehézkes).

Alapelvek:

- A világot objektumokkal írjuk le (objektum = megfogható, megkülönböztethető egyed, pl. egy-egy járat, utas, dolgozó).
- Minden objektumnak egyedi azonosítója van (OID), ez automatikusan generálódik neki és minden más OID-től különböző.
- Az objektumokat osztályokba soroljuk, az osztály elemei hasonlóak, ugyanolyan dolgokat tartunk róluk nyilván (pl. egy osztály lehet az összes utas, összes járat). Az egyes értékek persze lehetnek mások (az utasok neve különbözik, de minden utasnak van neve). Egy objektumot általában egy rekorddal adunk meg, az egyes mezők a nyilvántartott tulajdonságoknak felelnek meg.

Osztálydeklaráció

- Meg kell adni az osztály *nevét*.
- Az osztályhoz tartozó *attribútumok*: az osztályba tartozó objektumok jellemzői, lehetőleg egyszerűbb adattípusokkal megadva. (Erről majd később.)
- *Kapcsolatok* az osztályok között, ezeknek is van típusa, aszerint, hogy egy objektum egy másik osztáy egy vagy pedig több objektumával kapcsolódik-e össze (pl. egy járatnak egy kapitánya van, de sok utasa).

Az osztálydeklaráció formája

interface <osztály neve> {<attribútumok listája, kapcsolatok>};

Példa

```
interface Film {
    attribute string cím;
    attribute int hossz;
    attribute int év;
    attribute enum Szalag{színes, fekete-fehér} szalagfajta;
  };
```

Az osztály neve Film, négy attribútuma van. Az attribute kulcsszó után megadjuk az attribútum típusát (a lehetséges típusokról később), majd az attribútum nevét. Az utolsó sorban egy felsorolás jellegű (enum), szalagfajta nevű attribútumot definiálunk, ami a Szalag (kételemű) halmazból veszi az értékét.

Ez persze csak a kezdete egy osztálydeklarációnak, kapcsolatokat még nem is adtunk meg.

Egy objektum egy rekordnak felel meg, pl. a fenti megadaás szerint a Film osztály egy objektuma pl. (Amélie csodálatos élete, 120, 2000, színes).

Még egy példa

```
interface Színész {
    attribute string név;
    attribute Struct Cím{string város, string utca} lakcím;
  };
```

Itt a második attribútum struktúra típusú, ami két mezőből áll, az első mező neve város, típusa string, a másodiké utca, típusa string. Az attribútum neve lakcím.

Kapcsolatok megadása

Az objektumok tulajdonságait az attribútumokkal adjuk meg, az objektumok közötti hivatkozásokat pedig a kapcsolatokkal. Egy objektum kapcsolódhat egy vagy több másik obejktumhoz is.A kapcsolatokat ugyanott írjuk le, ahol az attribútumokat, a megadás módja:

relationship <osztálynév> <kapcsolatnév>;

ha egy objektumhoz vezet a kapcsolat, illetve

relationship <kollekcióoperátor>< <osztálynév> > <kapcsolatnév>;

ha több (a kollekcióoperátor mondja meg, hogy milyen) objektumhoz vezet a kapcsolat. A lehetséges kollekcióoperátorokról (Set, Bag, List, Array) majd később.

Példa kapcsolat megadására

A Film osztályba
relationship Set<Színész> szereplők;

és

relationship Színész főszereplő; kell.

Az első esetben egy filmhez a szinészek egy halmaza kapcsolódik, a második esetben egy filmhez egy darab színész tartozik.

Fontos! A kapcsolatot a másik osztálynál is jelölni kell és meg kell adni, hogy melyik kapcsolat inverzéről van szó.

Így a Színész osztályba is kell

relationship Set<Film> szerepelBenne;
inverse Film::szereplők;

és

relationship Set<Film> főszereplőBenne;
inverse Film::főszereplő;

És persze a Film osztályba is kell a két inverse:

relationship Set<Színész> szereplők;
inverse Színész::szerepelBenne;

és

relationship Színész főszereplő;
inverse Színész::főszereplőBenne;

Inverzek

Itt persze ugyanazon dolog két nézetéről van szó. Fontos konzisztenciatényező az inverzpárok feltüntetése, mert

- Elvárjuk, hogy ha X.Y. szerepel egy filmnél, mint szereplő, akkor az a film szerepljen nála a szereplBenne kapcsolatnál.
- Általában azok a jól megfogott kapcsolatok, amikhez könnyű, természetes inverzet találni.

Igazából egy dolog van csak, egy ilyen fajta megfeleltetés:

Színész	Film
A.Tautou	Amélie csodálatos élete
A.Tautou	Szeretni bolondulásig
M. Kassovitz	Amélie csodálatos élete
M. Kassovitz	Férfiak mélyrepülésben

Ennek kétféle elérése a két kapcsolat.

Kapcsolatok jellege

Egy C és egy D osztály közötti kapcsolat lehet

- több-több (sok-sok, N:N) kapcsolat: egy C-beli objektumhoz több D-beli és egy D-belihez több C-beli is tartozhat (pl. a szereplők/szerepelBenne kapcsolatpár).
- több-egy (sok-egy, N:1) kapcsolat: egy C-belihez csak egy D-beli tartozhat, de egy D-belihez tartozhat több C-beli is (pl. a Film és a Színész osztályok között levő főszereplője/főszereplőBenne pár).
- egy-egy (1:1) kapcsolat: egy C-belihez csak egy D-beli és egy D-belihez csak egy C-beli tartozhat (férj-feleség kapcsolat pl.).

A kapcsolat jellege azt mutatja, mennyire függvényszerű a kapcsolat az objektumok között. A kapcsolat jellege deklarációs kérdés, az osztály megadásakor döntjük el (azzal, hogy használunk-e kollekcióoperátort vagy sem).

(Egy több-több kapcsolat esetén is előfordulhat persze, hogy egy adott objektum csak egy másikhoz csatlakozik.)

Típusok az ODL-ben

Vannak alaptípusok, építkezési lehetőségek és megszorítások, amik szabályozzák az építkezést.

Alaptípusok

- Atomi típusok (elemi típusok): integer, real, float, char, string, boolean, enum
- Interface típusok: mi magunk csináljuk őket, a deklarált osztályok ezek (pl. Film, Színész)

Típuskonstruktorok

- *Halmaz*: ha T egy típus, akkor Set$< T >$ a T típusú elemek halmaza
- *Multihalmaz*: ha T egy típus, akkor Bag$< T >$ a T típusú elemek multihalmaza, azaz egy elem többször is szerepelhet
- *Lista*: ha T egy típus, akkor List$< T >$ a T típusú elemek listája, pl. string=List$< char >$
- *Tömb*: ha T egy típus, akkor Array$< T, i >$ a T típusú elemek i hosszú tömbje,
  pl. Array$<$ char$, 12 >= 12$ hosszú karakterlánc

- *Struktúra*: ha $T_1, T_2, \ldots, T_n$ típusok, $f_1, f_2, \ldots, f_n$ pedig mezőnevek, akkor Struct $<$ Név $>$ $\{T_1 f_1, T_2 f_2, \ldots, T_n f_n\}$ n mezőből álló $<$ Név $>$ nevű struktúra, ahol a mezők nevei $f_1, f_2, \ldots, f_n$, típusai pedig $T_1, T_2, \ldots, T_n$.
  Például: Struct Cím{string város, string utca}

Az első négy (Set, Bag, List, Array) típuskonstruktort **kollekcióoperátornak** hivjuk.

Megkötések

- *Attribútum típusa*: lehet atomi típus, struktúra atomi típusú mezőkkel, illetve ezekre lehet még egy kollekció operátort vagy egy struktúrát rakni (de csak egyszer!!!!)
  (Ezzel elég bonyolult típusokat lehet csinálni, de önmérséklet, mert nehéz lesz megvalósítani, ha túl bonyolult).
- *Kapcsolat típusa*: interface típus vagy interface típusra egyszer alkalmazott kollekcióoperátor (struktúra nem lehet!!!)

Megjegyzések

- Ugyanaz a típus nem lehet attribútum és kapcsolat típusa is.
- Kollekcióoperátort mindkét helyen lehet használni, de amire alkalmazom az más (elemi típus, illetve interface).
- Példa:
  Array$<$ Struct $N\{$string $m_1,$ string $m_2\}, 10 >$ lehet egy attribútum típusa

Példa még: `http://www.cs.bme.hu/~kiskat/adatbazis/211.ps`

E/K diagram

Eddig azt néztük meg, hogy ODL-ben hogyan lehet osztályokat, kapcsolatokat megadni és ezzel a DB fogalmi keretét kialakítani.

Most egy másik módszer jön, az E/K diagram, ezt könnyen át lehet majd írni relációs sémára.

E/K= egyed-kapcsolat vagy entitás-relációs (E/R, entity-relationship) modell

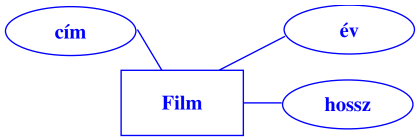
Szemléletes, könnyű vele dolgozni. Egy rajzot készítünk, ez ábrázolja az adatelemeket és a köztük levő kapcsolatot is.

Alapfogalmak

Hasonlítanak az alapelemek az ODL-hez:

- *Egyedhalmaz* (kb. mint az osztály az ODL-ben): elemei az egyedek (ODL-es objektumok), de itt nincs egyedi azonosító, az egyedek az attribútumaikkal és a kapcsolataikkal azonosítódnak.
  Rajzon:

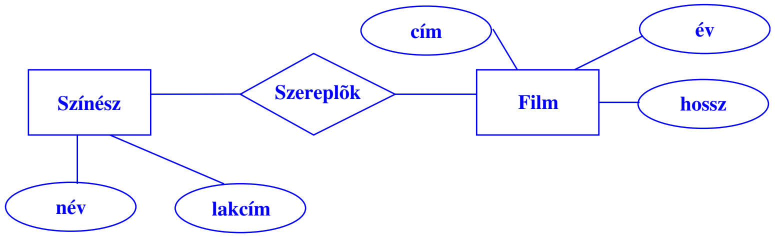
- *Attribútumok*: értékeik egy egyed tulajdonságait adják meg, mint az ODL-nél, de itt nincs formális előírás a típusokra, csak annyi, hogy legyenek egyszerűek, hogy könnyű legyen relációsra átírni.
  Szöveges jelölés: Film(Cím, Hossz, ...), rajzon:



- *Kapcsolatok*: egyedhalmazok közötti viszony, máshogy van, mint ODL-ben.
  - ★ ODL-ben minden kapcsolatot mindkét irányban reprezentálunk, itt egy kapcsolat = egy vonal

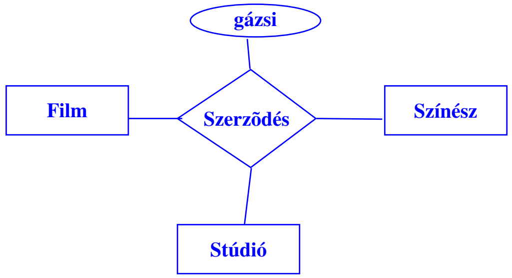
- ★ ODL-ben minden kapcsolat bináris (két osztály között megy), E/K-ban lehetnek többágú kapcsolatok is

Jelölés szövegesen: Szereplők(Film, Színész), illetve rajzon:



Ha az $R(E_1, E_2, \ldots, E_{10})$ kapcsolat 10 egyedhalmazt köt össze, akkor az R kapcsolat egy példánya egy 10 hosszú vektor $(e_1, e_2, \ldots, e_{10})$, ahol az e_i egy egyed az E_i egyedhalmazból.

Fontos különbség még az ODL-hez képest, hogy az E/K modellben a kapcsolatnak is lehet attribútuma:



Itt a gázsi a szerződéshez tartozik, ami a filmet, a színészt és a stúdiót köti össze. Lehetne úgy is csinálni, hogy a Szerződés kapcsolatnak lenne egy negyedik egyedhalmaza is, a Gázsi, egyetlen attribútummal, az összeggel, de felesleges olyan egyedhalmazt létrehozni, aminek csak egy attribútuma van.