Adatbázisok elmélete 3. 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

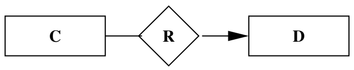
Kapcsolatok típusa

A bináris kapcsolatoknal ugyanúgy van, mint az ODL-nél: van több-több, több-egy és egy-egy kapcsolat. A többágú kapcsolat egy kicsit bonyolultabb.

Bináris kapcsolat

Az "egy" irányt nyíl jelzi, azaz ott van nyíl, amelyik egyedhalmazból csak egy tartozhat a másik egyedhalmaz egy egyedéhez.

Például, ha a C egyedhalmaz egy egyedéhez csak egy D-beli tartozhat az R kapcsolatnál, de egy D-belihez tartozhat több C-beli is, akkor:



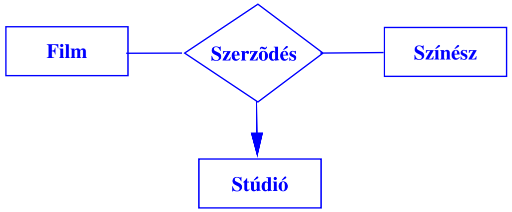
Ha egy-egy kapcsolat van, akkor persze mindkét oldalra kell a nyíl.

Többágú kapcsolat típusa

Az $R(E_1, E_2, \ldots, E_n)$ kapcsolat E_i felé egyirányú, ha igaz az, hogy a maradék $E_1, E_2, \ldots, E_{i-1}, E_{i+1}, \ldots, E_n$ egyedhalmazokból bárhogy válsztva ki egy-egy egyedet (e_k-t az E_k-ból), maximum egy olyan E_i-beli e_i egyed van, amivel az $(e_1, e_2, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n)$ egyedvektorra az R kapcsolat fennáll.

(Természetesen lehet egy többágú kapcsolat egynél több irányban is "egy" jellegű.)

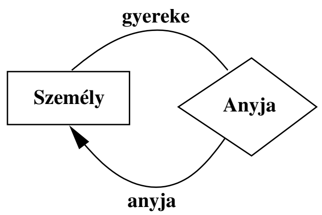
Példa:



Egy rögzített (film, színész) párhoz csak egy stúdió tartozik, de pl. egy rögzített (stúdió, film) párhoz tartozhat több színész.

Nem minden írható le ilyen módon, de nem baj, úgyis az a fontos, hogy majd a relációs megadásnál pontosak tudjunk lenni. A fenti példánál, a rajzon nem tudjuk jelölni azt, hogy már a film egyedül meghatározza a stúdiót, de a relációs modellben lesz erre eszközünk.

Ha egy kapcsolatban egy egyedhalmaz többször is szerepel, akkor a nyilakon/vonalakon jelöljük a különböző szerepeket. pl:



Fontos megcímkézni a vonalakat, mert ahhoz a szerephez tartozik az "egy" nyíl, ami az anyára vonatkozik, a gyerekághoz nem kell.

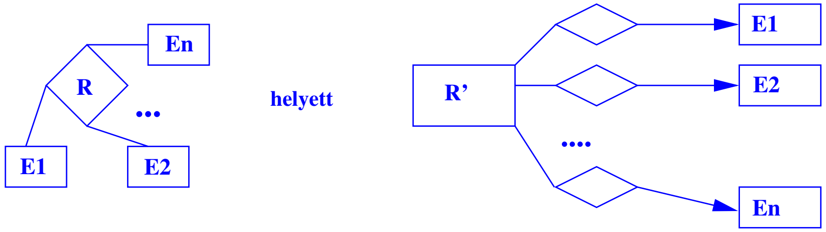
Sokágú kapcsolat átírása binárisra

Miért kell/jó ez?

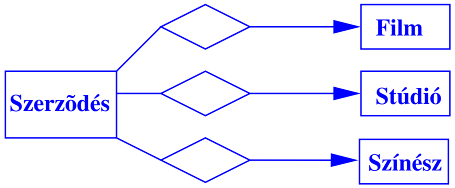
- átláthatóbb a bináris
- a megvalósításban könnyebben kezelhető
- látszik, hogy nem baj, hogy az ODL csak binárist tud, hisz a többágút is át lehet ilyenné írni

Az átíráshoz fő ötlet: egy $R(E_1, E_2, \ldots, E_n)$ kapcsolat egy eleme egy n egyedből álló vektor: $(e_1, e_2, \ldots, e_n)$.
Az átírásnál létrehozunk egy új egyedhalmazt, melynek az ilyen vektorok lesznek az egyedei és az ilyen vektorokat bináris több-egy kapcsolatok (n darab) fogják az $E_1, \ldots, E_n$ egyedhalmazokhoz kötni.

Rajzon:



Konkrét példa (az előbbi háromágú szerződéses példa átírva):



Megjegyzések az átíráshoz

- Ez az átírás azért is jó, mert egy kapcsolatot majd úgyis valahogy így tudunk kezelni a fizikai megvalósításban: a kapcsolat egy eleme egy három mutatóból álló tömb lesz, ahol ez egyes mutatók a kapcsolatot alkotó egyedekre (film, színész, stúdió) mutatnak.
- Az átírásnál persze veszíthetünk infót: az előbbi példánál elveszett az, hogy a többes kapcsolat "egy" volt a Stúdió fele. Ez nem baj, ezt majd a végén a relációs modellben úgyis finomabban le tudjuk írni.
- Mivel minden többágú kapcsolatot át lehet írni binárissá, azért elég volna csak ilyen bináris kapcsolatokat használni.
- Amúgy az ODL-ben lényegében ez történik, ha ott akarnánk ábrázolni azt a hármas kapcsolatot, ami a filmeket, színészeket és stúdiókat összeköti, akkor ehhez fel kellene vennünk egy negyedik osztályt így (és ez lényegében ugyanaz a helyzet, amit az E/K modellben kaptunk az átíráskor):


```
interface Szerződés {
    relationship Stúdió gyártja;
      inverse Stúdió::StúdióSzerződése;
    relationship Film filmje;
      inverse Film::FilmSzerződése;
    relationship Színész szereplője;
      inverse Színész::SzínészSzerződése;
      };
```

Feladat

Javasoljon ODL-sémát egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.

Megoldás (néhány megoldás a sok lehetséges közül):

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```

```
interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```


Variációk

- Lehetett volna struktúra a lakcím típusa:

```
interface Ügyfél {
    attribute string név;
    attribute Struct Cím{string város, string utca, int házszám} lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```


Variációk

- Ha több címe is lehet egy embernek:

```
interface Ügyfél {
    attribute string név;
    attribute Set< Struct Cím{string város, string utca, int házszám}> lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla::tulajdonosai;
};
```


Variációk

- Lehetett volna felsorolástípus a számla típusa:

```
interface Számla {
    attribute int számlaszám;
    attribute enum Típus{ betét, folyó } számlaTípus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

Variációk

- Ha egy embernek csak egy számlája lehet és egy számlának csak egy tulajdonosa:

```
interface Ügyfél {
    attribute string név;
    attribute string lakcím;
    attribute int telefonszám;
    attribute int szemszám;
    relationship Számla számlája;
        inverse Számla::tulajdonosa;
};
```

```
interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Ügyfél tulajdonosa;
        inverse Ügyfél::számlája;
};
```


Variációk

- Ha egy embernek több lakhelye lehet és az egyes lakhelyekhez lehet több telefonszám is:

```
attribute Set< Struct Cím{ string lakcím, Set<int> telszámok }> lakcím;
```

Struct-on belül nem lehet kollekciótípus illetve két kollekciótípus sem lehet egymásba ágyazva egy attribútum típusában.
Az se lenne jó, ha külön nyilvántartunk egy csomó lakcímet és ettől függetlenül az összes telefonszámot, mert akkor nem fogjuk tudni, hogy melyik címhez melyik szám tartozik.


```
interface Ügyfél {
    attribute string név;
    attribute int szemszám;
    relationship Set<Számla> számlái;
        inverse Számla: tulajdonosai;
    relationship Set<Lakhely> ittLakik;
        inverse Lakhely::lakói;
};
```

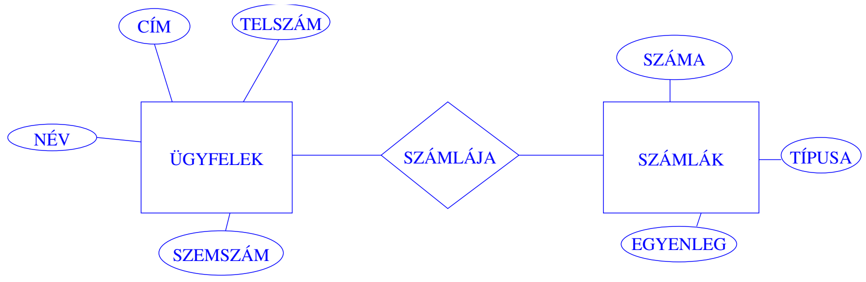
```
interface Számla {
    attribute int számlaszám;
    attribute string típus;
    attribute int egyenleg;
    relationship Set<Ügyfél> tulajdonosai;
        inverse Ügyfél::számlái;
};
```

```
interface Lakhely {
    attribute string város;
    attribute string utca;
    attribute int házszám;
    attribute Set<int> telefonszámok;
    relationship Set<Ügyfél> lakói;
        inverse Ügyfél::ittLakik;
};
```


Feladat

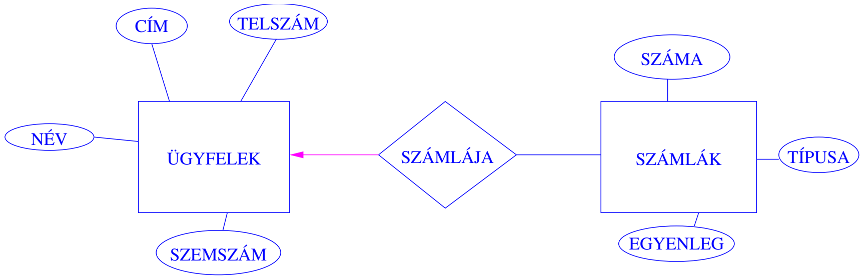
Tervezzen E/K diagrammot egy olyan banki adatbázishoz, amely tartalmazza az ügyfeleket és azok számláit.
Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és személyi számukat. A számláknak legyen számlaszámuk, típusuk (takarékbetét számla, folyószámla pl.) és egyenlegük.

Megoldás (néhány megoldás a sok lehetséges közül):



Variációk

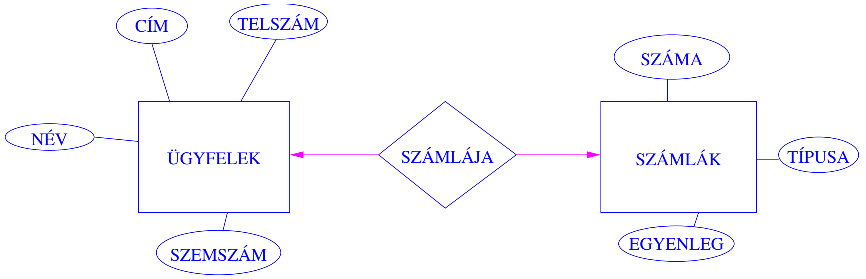
- Ha egy számlának csak egy tulajdonosa lehet:



2

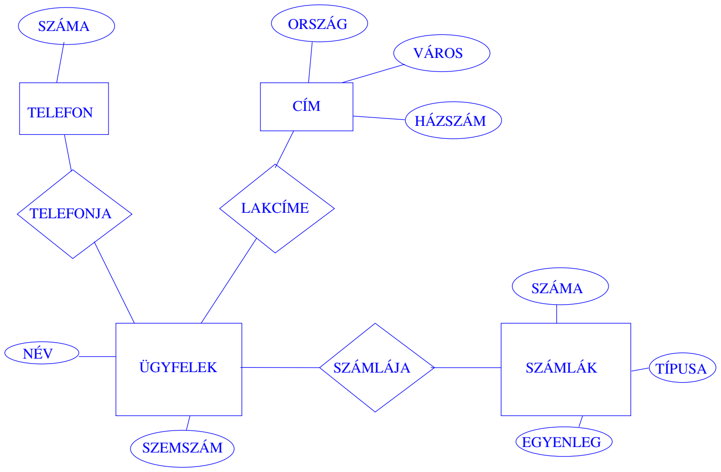
Variációk

- Ha egy számlának csak egy tulajdonosa lehet és egy ügyfélnek csak egy számlája lehet:



Variációk

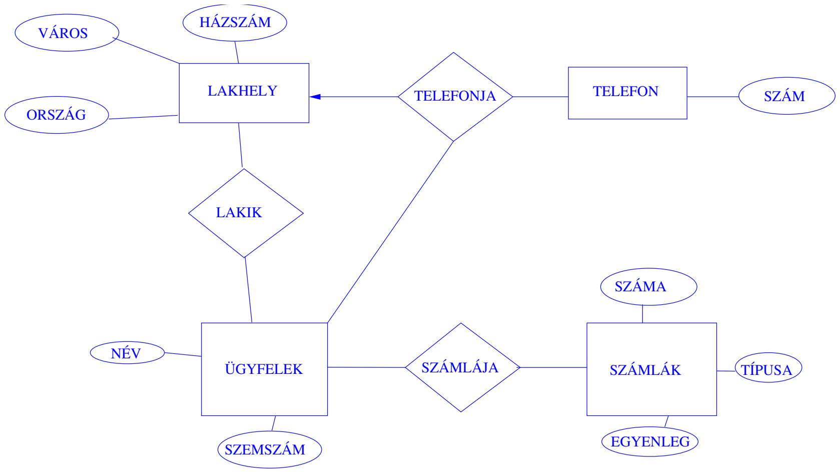
- Ha egy embernek több címe és több telefonszáma is lehet, de azt nem kell nyilvántartani, hogy mik az összetartozó lakcím-telefonszám párok:



Azért nem lehetett az “ügyfelek” egyedhalmaz attribútuma a telefonszám, mert csak egyszerű típusokat használhatunk, kollekciókat nem.

Variációk

- Ha azt is nyilván kell tartani, hogy mik az összetartozó lakcím- telefonszám párok:



Alosztályok

Egy osztály (egyedhalmaz) speciális tulajdonságú, de egymáshoz hasonló objektumai (egyedei) alkotják. Ezeket érdemes együtt kezelni, de úgy, hogy a (nagyobb) osztályba való tartozásuk is megmaradjon.

ODL-ben

Pl. a Film a osztályon belül akarhatunk külön Rajzfilm, Vígjáték, Krimifilm alosztályokat. Ennek megadása az osztályok deklarációjakor:

A Film osztályt megadjuk, úgy, mint eddig, és

```
interface Rajzfilm:Film{
    relationship Set<Színész> hangok;
      inverse Színész::hangItt;
        };
```

Vagyis először megadjuk az alosztály nevét, aztán : után annak az osztálynak a nevét, aminek ő alosztálya lesz. A zárójelen belül már csak azokat az attribútumokat/kapcsolatokat kell megadni, amik az alosztály sajátjai, a többi attribútumot/kapcsolatot örökli a főosztálytól.

Példa még:

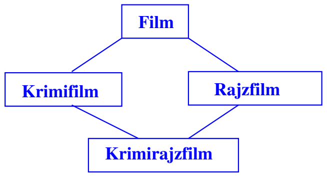
```
interface Krimifilm:Film{
    attribute Set⟨string⟩ bizonyítékok;
        };
```

Sőt, lehet több szintű öröklés, illetve egy alosztály örökölhet két főosztálytól is:

```
interface Krimirajzfilm:Krimifilm, Rajzfilm{
        };
```

Itt a Krimirajzfilm alosztály örökli a Krimifilm és a Rajzfilm (al)osztály összes attribútumát és kapcsolatát (természetesen azokat is, amiket azok is úgy örököltek), neki magának pedig semmi saját attribútuma/kapcsolata nincsen.

Az öröklési kép ilyen:



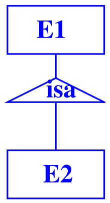
Ha olyan filmeket is nyilván akarunk tartani, amik rajzfilmek (akarunk hangok kapcsolatot a Színészek felé) és krimik is (akarunk bizonyítékok attribútumot), akkor muszáj létrehoznunk a fenti Krimirajzfilm osztályt, mert az objektumos szemléletben minden objektum csak egy (al)osztályhoz tartozhat. Ha nem lenne Krimirajzfilm osztály, akkor vagy csak a hangokat vagy csak a bizonyítékokat tudnánk feljegyezni.

Probléma lehet a többszörös öröklődésnél, ha ugyanolyan nevű attribútumot/kapcsolatot több helyről is örökölne egy alosztály. Ilyenkor valamelyiket át kell nevezni.

Alosztályok az E/K modellben

Sokkal egyszerűbb, mint az ODL-ben, de a cél ugyanaz: egy egyedhalmaz speciális egyedeit együtt kezelni. Ehhez egy speciális (*isa*, magyarul *azegy*) kapcsolat van. (Motiváció: A ló az egy állat = a lovak alosztályát alkotják az állatoknak.)

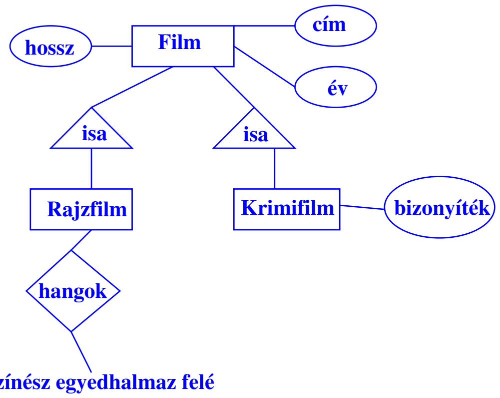
Jelölés, ha E_2 alosztálya E_1-nek:



Az első esetben a nyíl arra mutat, amelyik a Fő osztály, ez most nem a több-egy kapcsolatnál megszokott nyíl, az *isa* ilyen szempontból is speciális kapcsolat. A második esetben az alosztály van alul. Az alárendelt halmaz most is örökli a főosztály attribútumait és kapcsolatait, de persze lehetnek neki sajátjai is.

Különbségek a két modell között

A filmes példánál ez lesz az E/K modellben:



Az ODL-ben minden objektum csak egy osztályhoz tartozhat (ezért ott kellett Krimirajzfilm osztály), az E/K modellnél viszont lehetséges az, hogy egy egyed egyszerre több osztály/alosztály része is, ilyenkor az attribútumait/kapcsolatait úgy szedi össze a felmenőitől (E/K-ban nem kell Krimirajzfilm alosztály). A relációs modellre való átíráskor majd úgyis egységesen fogjuk ezeket a jellemzőket kezelni (lásd majd ott).
Így az E/K modellben egy olyan film, ami krimi is és rajzfilm is (pl. Macskafogó), három helyről szedi össze az attribútumait/kapcsolatait: a címét, hosszát és gyártási évét a Film egyedhalmazból, a hangjait a Rajzfilm alosztályból, a bizonyítékot pedig a Krimifilmből.

Példa

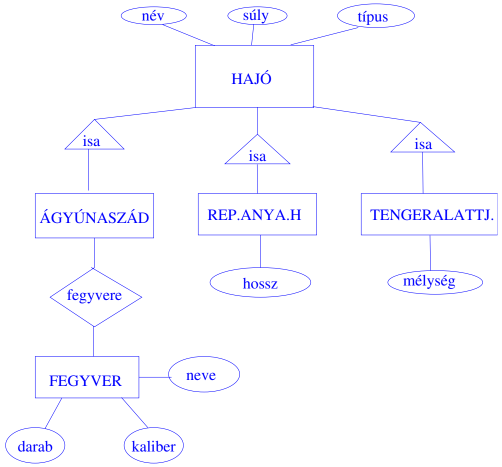
Hadihajók adatbázisát szeretnénk megadni ODL-ben és E/K diagrammal. Minden hadihajóról nyilvántartjuk a nevét, a vízkiszorítását tonnában, típusát. Négyfajta hajót akarunk nyilvántartani:

1. Ágyúnaszád (itt nyilvántartjuk a fegyverek számát és kaliberét)
2. Repülőgép-anyahajó (tároljuk a leszállópálya hosszát)
3. Tengeralattjáró (max. merülési mélység)
4. Csata-repülôgép-anyahajó (olyan ágyúnaszád, ami repülőgépanyahajó is).

Megoldás ODL-ben

```
interface Hajó {
    attribute string név;
    attribute int súly;
    attribute string típus;
};

interface Ágyúnaszád:Hajó {
    attribute Set<Struct Fegyver{string név, int darab, int kaliber}> fegyverek;
};

interface Repülőgép-anyahajó:Hajó {
    attribute int hossz;
};

interface Tengeralattjáró:Hajó {
    attribute int mélység;
};

interface Csatarepülőgép-anyahajó:Ágyúnaszád, Repülőgép-anyahajó { };
```


Megoldás E/K modellel



Megjegyzés: Itt nem kell külön egyedhalmaz a Csatarepülőgép-anyahajóknak, mert nincs olyan attribútum, ami csak ezeknél lenne. Egy ilyen hajót úgy tartunk majd nyilván, hogy lesznek attribútumai mind az ágyúnaszádoktól, mind a repülőgép-anyahajóktól.

Megszorítások

Olyan megszorításokat is ábrázolni akarunk az adathalmazokon, attribútumokon, kapcsolatokon, amik nem fejezhetők ki pusztán az attribútumok és a kapcsolatok felsorolásával.
Ezek további infók, amik

- a séma részei, ezért már a tervezéskor kell rájuk gondolni,
- olyan megkötéseket tartalmaznak, amikre majd mindig figyleni kell.

Nem világos, hogy mik a jó megkötések, miket lehet jól kezelni.

Típusai (tipikus, (néha) jól kezelhető megszorítások):

- *Kulcsok*: olyan attribútum, vagy attribútumhalmaz megadása, ami az egyedet/objektumot már egyértelműen azonosítja. (Pl. személyi szám vagy filmnél gyártási év és cím.) A tervezéskor döntjük el, hogy mik alkossanak kulcsot (persze a valóságot szem előtt tartva). A kulcshoz tartozó attribútumoknak értékeket adva, legfeljebb egy objektum vagy egyed létezhet, amikhez ezek az értékek tartoznak.
  Néha tűnhet úgy az aktuális adatokból, hogy valami kulcs (mert akkor éppen nincs két egyed ugyanolyan értékekkel), de ettől még nem lesz kulcs valami, az csak a deklarációtól függ.

- *Egyértékűségi megszorítások*: előírhatjuk, hogy valami érték vagy értékkombináció legyen egyedi. Pl. ilyen a kulcsok megadása, vagy az, hogy egy kapcsolat nem rendelhet halmazt értékül egy egyedhez/objektumhoz. Ilyenek lesznek majd a funkcionális függések is a relációs modellben.
- *Hivatkozási épség*: a hivatkozott dolognak léteznie kell.
- *Értelmezési tartomány korlátozása*: attribútum lehetséges értékeire megkötés (pl. magasság legyen kisebb 300-nál, film gyártási éve 1800 utáni). *Módszerek erre:* típusok megadása, felsorolás típus, konkrétabban majd az SQL DDL-jénél.
- *Egyéb megszorítások*: minden más, pl. kapcsolat fokának korlátozása (egy filmnek max. 10 szereplőjét akarjuk nyilvántartani).

Általános gond: Mik a jó megszorítások? Miket lehet megvalósítani?

Ez persze majd a konkrét megvalósítástól függ, a konkrét DDL-től, de azért jó lenne már a modellezéskor is annyit leírni, amennyit csak lehet.

A megszorítások haszna

- jobban/valósághoz közelebbi módon le lehet velük írni a világot
- segíthetik a tárolást (elég pl. a kulcsattribútumokat megadni kereséskor)