18. Adatbázisok tervezése és lekérdezése

Relációs adatmodell, egyed-kapcsolat modell és átalakítása relációs adatmodellbe

Relációs adatmodell

Adatmodell: Egy adatmodell a valóság objektumait (egyedeit), azok tulajdonságait, ill. a köztük lévő kapcsolatokat ábrázolja. Az adatmodell a számítógép és felhasználó számára is megadja, hogy hogyan néznek ki az adatok. Az adatok leírására szolgáló jelölés, melynek részei:

- **az adat struktúrája** (*statikus elemek*)
- **az adaton végezhető műveletek** (*dinamikus elemek*)
- **az adatokra tett megszorítások** (*integritási feltételek)*
  Például: egy személyi igazolvány számhoz nem tartozhat két különböző ember

Relációs adatmodell: A relációs adatmodellben egy adatbázis kétdimenziós adattáblák összessége, ahol minden adattábla egy reláció vagyis sorok (rekordok) halmaza.

Reláció: Az adatelemek megnevezett, összetartozó csoportjából kialakított olyan kétdimenziós táblázat, amelyik sorokból (rekordok) és oszlopokból (attribútumok) áll. A relációnak egyedi neve van és oszlopait az attribútumok címzik. A reláció rekordjait tetszőlegesen megcserélhetjük, sorok sorrendje lényegtelen (a halmazszemlélet miatt).

Formálisan: Legyenek $H_1, \ldots, H_n$ alaphalmazok, akkor $r \subset H_1 \times \ldots \times H_n$ részhalmazt relációnak nevezzük.

Attribútumok: A reláció fejrészében találhatóak az attribútumok. Minden attribútumhoz tartozik egy értékkészlet, amelyből felveheti értékeit. Az i-edik attribútum értékkészletét $dom(A_i)$ jelöli.

Adattípus vagy *sortípus*:

$$\big\langle \textit{attr.név}_1 : \text{értéktípus}_1,\ \ldots,\ \textit{attr.név}_n : \text{értéktípus}_n \big\rangle, \text{ ahol attr.név}_i \neq \text{attr.név}_j \quad (\forall i \neq j).$$

A reláció nevét és a reláció attribútumainak halmazát együtt nevezzük ***relációsémának***.
Relációséma: ***Relációnév(sortípus)***, azaz

$$R(\textit{attr.név}_1 : \text{értéktípus}_1, \ldots, \textit{attr.név}_n : \text{értéktípus}_n)$$
$$\Updownarrow$$
$$R(A_1, \ldots, A_n) \Leftrightarrow R(U),\ U = \big\{A_1, \ldots, A_n\big\}.$$

A relációs modellben az adatbázis egy vagy több relációsémát tartalmaz. A relációsémákból álló halmazt az adatbázisban ***relációs adatbázissémának*** vagy röviden ***adatbázissémának*** nevezzük.

Adatbázis séma jelölése: $\mathbb{R}$, ahol $\mathbb{R} = \{R_1, ..., R_k\}$.

Relációs séma feletti reláció előfordulás (példány, instance)

A reláció azon sorait, amelyek különböznek az attribútumokból álló fejléc soraitól, ***sor***oknak (tuple) nevezzük. A reláció minden egyes attribútumához tartozik a sorban egy *komponens*.

Egy reláció előfordulásainak halmaza a sor-típusnak megfelelő véges sok sor (sorok halmaza), azaz

$$t_1, \ldots, t_n \text{ ahol } t_i \text{ (tuple, sor, rekord)} \quad (i = 1, \ldots, n) \quad \text{(véges sok)}$$

Mit jelent egy konkrét sor (rekord)?

$$t_i :< A_1 : \text{érték}_{i,1}, \ldots, A_m : \ \text{érték}_{i,m} >, \ \text{ahol érték}_{i,j} \in dom(A_j) \quad (i \in 1, \ldots, n, \ j \in 1, \ldots, m)$$

- n - számosság (sorok (rekordok) száma)
- m - dimenzió (oszlopok (attribútumok) száma)

Az attribútumok sorrendje nem rögzített a relációsémában. Azonban egy-egy előfordulás ábrázolása esetén viszont rögzítésre kerül.

A relációs modellben követelmény, hogy minden sor minden komponense atomi, azaz elemi típusú legyen (például egész vagy karaktersorozat).

Kulcsok

A $K \subseteq \{A_1, \ldots, A_n\}$ attribútumhalmazt, amelyre az $r \in R(A_1, \ldots, A_n)$ reláció minden sorára különböző ***szuperkulcs***nak nevezzük:

$$\forall t_i, t_j \in r, \ i \neq j : t_1[K] \neq t_2[K]$$

A K attribútumhalmazt ***kulcs***nak nevezzük, ha minimális szuperkulcs.

Egy relációsémában több kulcs is előfordulhat, vagyis több attribútumhalmazt is ki tudunk jelölni kulcsként.

- A kulcs egy attribútumból áll: ***egyszerű kulcs***
- A kulcsot több attribútum alkotja: ***összetett kulcs***.
    - Ha több kulcs is meghatározható a sémában, akkor kijelöljük az egyiket, amelyet ***elsődleges kulcs***nak nevezünk.

Külső vagy idegen kulcs: Ha egy attribútum egy másik séma elsődleges kulcsára hivatkozik.

- $R(A_1, ...A_m)$ reláció, és $X = \{A_{i_1}, ...A_{i_k}\}$ kulcs.
- $S(B_1, ...B_n)$ reláció, és $Y = \{B_{j_1}, ...B_{j_k}\}$ idegen kulcs.
- Az Y az X-re hivatkozik a megadott attribútum sorrendben: $B_{j_1} \to A_{i_1}$-re, és így tovább.

Hivatkozási épség: Megszorítás a két tábla együttes előfordulására.
Ha $s \in R_1$ sor, akkor $\exists t \in R_2$ sor, amelyre

$$s[B_{j_1}, ...B_{j_k}] = t[A_{i_1}, ...A_{i_k}]$$

A relációs adatmodell több szempontból is előnyös, amik miatt elterjedt és kifinomult.

- Az adatmodell egy egyszerű és könnyen megérthető strukturális részt tartalmaz.
- A természetes táblázatos formát nem kell magyarázni, és jobban alkalmazható.
- A relációs modellben a fogalmi-logikai-fizikai szint teljesen szétválik, nagyfokú logikai és fizikai adatfüggetlenség.
- A felhasználó magas szinten, hozzá közel álló fogalmakkal dolgozik (implementáció rejtve).
- Elméleti megalapozottság, több absztrakt kezelő nyelv létezik, például relációs algebra (ezen alapul az SQL automatikus és hatékony lekérdezés optimalizálása).
- Műveleti része egyszerű kezelői felület, szabvány SQL.

Egyed-kapcsolat (E/K) modell

Grafikus ábrázolási mód, amellyel egy-egy adatbázis sémája megtervezhető. Az egyed-kapcsolat modellben az adatok szerkezetét grafikusan, egyedkapcsolat diagramon ábrázoljuk. Az így elkészített ábra később könnyen átalakítható relációs adatmodellé.

A diagram elemei:

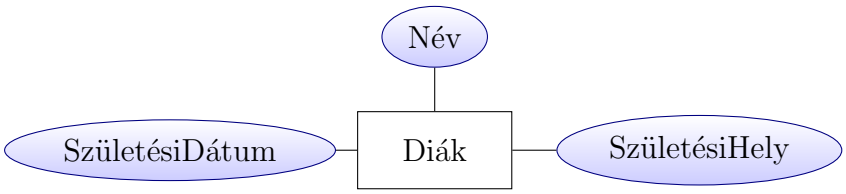
- egyedhalmazok
- attribútumok
- kapcsolatok

Egyed: Az a valami, dolog, amit ismeretekkel akarunk leírni; valami ami van és megkülönbözhető. Az egyedek a valóság azon elemei, melyek számunkra valamilyen lényeges információt hordoznak, *egymástól megkülönböztethetők.*

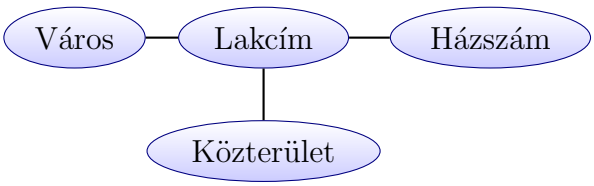
Egyedhalmazok: Hasonló egyedek összessége. Diagrammon: *téglalap.*

Diák

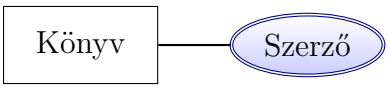
Tulajdonság (*attribútum*): A tulajdonság az, amivel az egyedet leírjuk, ami alapján az egyedhalmaz egyedei megkülönböztethetőek a többi egyedtől. A tulajdonság egy konkrét értéke a tulajdonság előfordulása. A tulajdonság előfordulások összességét tulajdonsághalmaznak nevezzük. Az attribútumok atomiak, tehát nincs belső szerkezetük. Diagrammon: *ovális*.



Összetett tulajdonság (*attribútum*): Olyan tulajdonság, amelynek magának is vannak tulajdonságai.



Többértékű tulajdonság: nem egyetlen adat jellemzi a tulajdonságot, hanem adatok halmaza (sorrendiség nélkül) vagy listája (sorrend számít).



Séma: $E(A_1, ...A_n)$ egyedhalmaz séma ahol:

- E név,
- A_i tulajdonság (attribútumok),
- $dom(A_i)$ a lehetséges értékek halmaza.

Előfordulás: $E(A_1, ...A_n)$ egyedhalmaz séma egy előfordulása $E = \{e_1, \ldots, e_m\}$

- $e_i(k) \in dom(A_k)$ az egyedek halmaza.
- Semelyik két egyed nem egyezik meg minden attribútumán $\rightarrow$ (vagyis az összes tulajdonság szuperkulcsot alkot), minimális szuperkulcs = kulcs.

Kapcsolatok: Kapcsolat két vagy több egyedhalmazt köthet össze egymással. A kapcsolatok tulajdonképpen egyedosztályok előfordulásai közötti relációk. A kapcsolatokat elláthatjuk névvel és a tartozhatnak hozzá attribútumok is. Diagrammon: *rombusz*.

- $K(E_1, \ldots, E_k, A_1, \ldots, A_n)$ egy kapcsolat sémája, ahol:
  - K a kapcsolat neve,
  - E_i az egyedhalmazok sémái,
  - $A_1, \ldots, A_n$ a kapcsolathoz tartozó attribútumok
- Ha $k = 2$, akkor bináris kapcsolatról, $k > 2$ esetén többágú kapcsolatról beszélünk.
- $K(E_1, \ldots, E_p)$ sémájú kapcsolat előfordulása, $K = \{(e_1, \ldots, e_p)\}$ egyed p-esek halmaza, ahol $e_i \in E_i$. A kapcsolat előfordulásaira tett megszorítások határozzák meg a kapcsolat típusát.

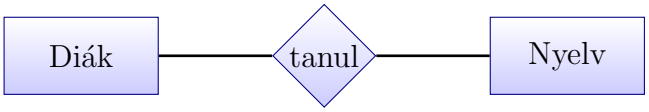
Kapcsolat attribútum: A kapcsolattípusoknak is lehetnek attribútumaik, amelyek hasonlóak az egyedtípusokéihoz. A kapcsolat attribútuma a két egyedhalmaz együttes függvénye, de egyiké sem külön.

Kapcsolatok típusai

- ***egy-egy*** *(1:1)*: Egy E_1-beli egyedhez pontosan egy E_2-beli egyed tartozhat és fordítva.
  Például: Ha a magyar nők és a magyar férfiak közötti jogi értelemben vett érvényes házastársi kapcsolatot szeretnénk modellezni, akkor egy-egy kapcsolatot kell alkalmaznunk.

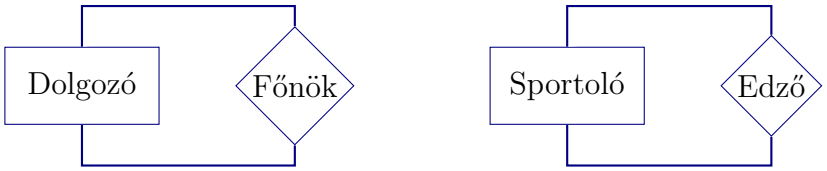
- ***egy-sok*** *(1:n)*: Minden E_2-beli egyedhez legfeljebb E_1-beli egyed kapcsolódik, de egy E_1-beli egyedhez több E_2-beli egyed kapcsolódhat.
  Például: Egy anyának egy vagy több gyermeke is lehet, de egy gyermeknek csak és kizárólag egy vér szerinti anyukája lehet.

- ***sok-sok*** *(n:m)*: Minden E_1-beli egyedhez több E_2-beli egyedek egy halmazát rendelhetjük és fordítva. Ha sok-sok kapcsolat van E_1 és E_2 egyedhalmazok között, akkor E_1-et egyenes vonallal kötjük össze E_2-vel.
  Például: Egy diák több nyelvet is tanulhat, valamint egy nyelvet több diák is tanulhat.



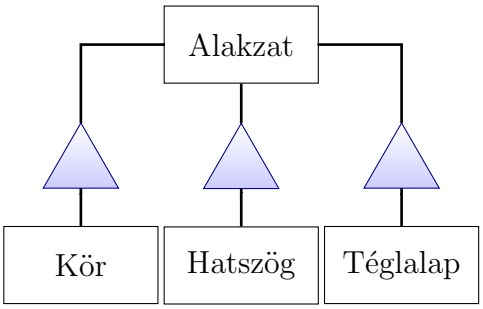
Speciális kapcsolatok

- *Önamagára mutató kapcsolat*: Elképzelhető, hogy valamilyen oknál fogva egy egyed önmagával is kapcsolatban állhat, például: dolgozó és főnöke, hiszen a főnök is egy dolgozó, vagy sportoló és edzője, hiszen az edző is egy sportoló.



- ***Alosztály*** **(„isa”-„az-egy”)**: Gyakran előfordul, hogy egy egyedhalmaz egyedei között egyeseknek olyan speciális tulajdonságaik vannak, amelyekkel nem rendelkezik a halmaz minden egyede. Így célszerű speciális egyedhalmazokat, ún. alosztályokat definiálni, ahol minden alosztálynak vannak speciális attribútumai és/vagy kapcsolatai.

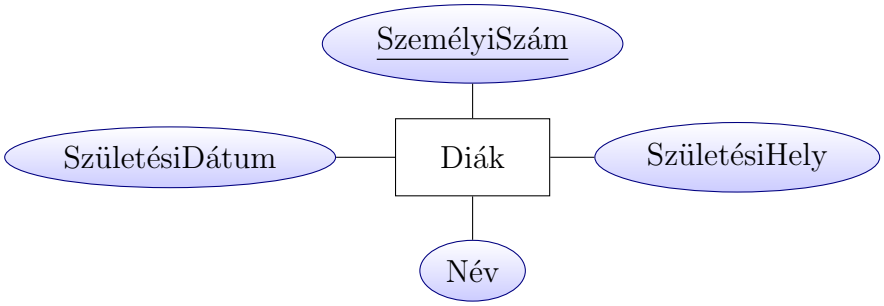
  Ha egy egyedhalmazt alosztályival speciális kapcsolat köt össze, azt ”az-egy” (angolul isa) kapcsolatnak nevezzük (például ”egy A az egy B” állítás kifejezi a speciális ”az-egy” kapcsolatot A-ból B-be). Az öröklési (az-egy) kapcsolatot a hagyományostól eltérően háromszöggel jelöljük, ezzel is kifejezve e kapcsolattípus különlegességét. A háromszög egyik oldalát az alosztállyal kötjük össze, ellenkező oldali csúcsát pedig az ősosztállyal (szuperosztállyal). Minden öröklési kapcsolat egy-egy kapcsolat, de az ezt kifejező nyilakat nem tüntetjük fel külön a diagramon.



Kulcsok

Szuperkulcs: Az egyedhalmaz szuperkulcsa egy azonosító, vagyis olyan tulajdonság-halmaz, amelyről feltehető, hogy az egyedhalmaz előfordulásaiban nem szerepel két különböző egyed, amelyek ezeken a tulajdonságokon megegyeznek. Az összes tulajdonság mindig szuperkulcs.

Kulcs: Ha attribútumok egy halmaza kulcsot alkot egy egyedhalmazon belül, akkor nincs két olyan egyed, amely megegyezik a kulcs összes attribútumán. Diagrammon: *attribútumnév aláhúzásával.*



Megjegyzés: Jelölhető több kulcs úgy is, hogy megadunk egy kulcsot, ami elsődleges kulcs lesz (ezt úgy kezeljük, mint ha több kulcs nem is lenne) és a többi kulcsot, vagy ne mis jelöljük vagy megjegyzésben felsoroljuk azokat.

Hivatkozási épség: Azt a követelményt fejezi ki, hogy egy egyedhalmaz egy vagy több attribútumának az értéke elő kell, hogy forduljon egy másik egyedhalmaz adott attribútumának értékeként. Diagrammon: *kerek végű nyíl.*

Példa:

- A *Stúdiók* egyedhalmazhoz megy egy kerek nyíl a *gyártó* kapcsolatban. Ez azt jelenti, hogy annak a stúdiónak, amely gyártott egy filmet, mindig benne kell lennie a *Stúdiók* egyedhalmazban.
- Hasonlóan a *Stúdiók* egyedhalmazhoz megy egy kerek nyíl a *vezető* kapcsolatban. Ez azt jelenti, hogy ha egy elnök vezetője egy stúdiónak, akkor annak a stúdiónak léteznie kell a *Stúdiók* egyedhalmazban.

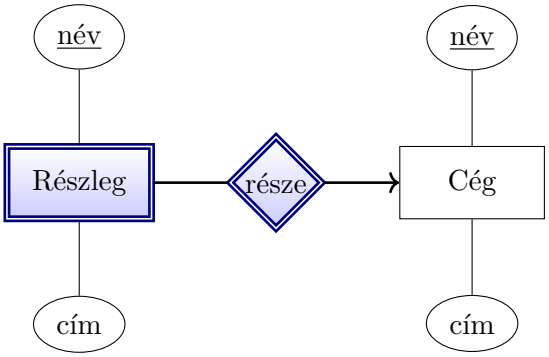
Gyenge egyedhalmaz

Előfordulhat, hogy egy-egy egyedhalmazt csak más egyedhalmazok attribútumainak ismeretében azonosíthatunk egyértelműen. Ezeket gyenge egyedhalmazoknak nevezzük. A gyenge egyedhalmaz az „azonosító” egyedhalmazokhoz egy-sok kapcsolattal kapcsolódhat.

Diagrammon:

- *dupla téglalap* az egyedhalmaznak és
- *dupla rombusz* azoknak a kapcsolatoknak, amiken keresztül megy az azonosítás.

Példa gyenge egyedhalmazra:



A részleg neve nem kulcs, mert sok cégnél lehet azonos részlet, ami még a címmel sem azonosítható egyértelműen. Például egy többszintes épületben két különböző cégen belül is lehet IT vagy HR, stb. Amennyiben azonban a céget is bevesszük az azonosításba egy kapcsolaton keresztül, úgy már egyértelművé válik.

Tervezési alapelvek

- *Valósághű modellezés*: Megfelelő tulajdonságok tartozzanak az egyedosztályokhoz, például a tanár neve ne a diák tulajdonságai közé tartozzon.
- *Redundancia elkerülése*: Az *index(etr-kód, lakcím, tárgy, dátum, jegy)* rossz séma, mert a lakcím annyiszor ismétlődik, ahány vizsgajegye van a diáknak, helyette 2 sémát érdemes felvenni: *hallgató(etr-kód, lakcím)*, *vizsga(etr-kód, tárgy, dátum, jegy)*.
- *Egyszerűség*: Fölöslegesen ne vegyünk fel egyedosztályokat, például a *naptár(év,hónap,nap)* helyett a megfelelő helyen inkább dátum tulajdonságot használjunk.
- *Tulajdonság vagy egyedosztály*: Például a vizsgajegy osztály helyett jegy tulajdonságot. használjunk.

Egyed-kapcsolat modell átalakítása relációs adatmodellbe

Átalakítás E/K modell → relációs adatmodell:

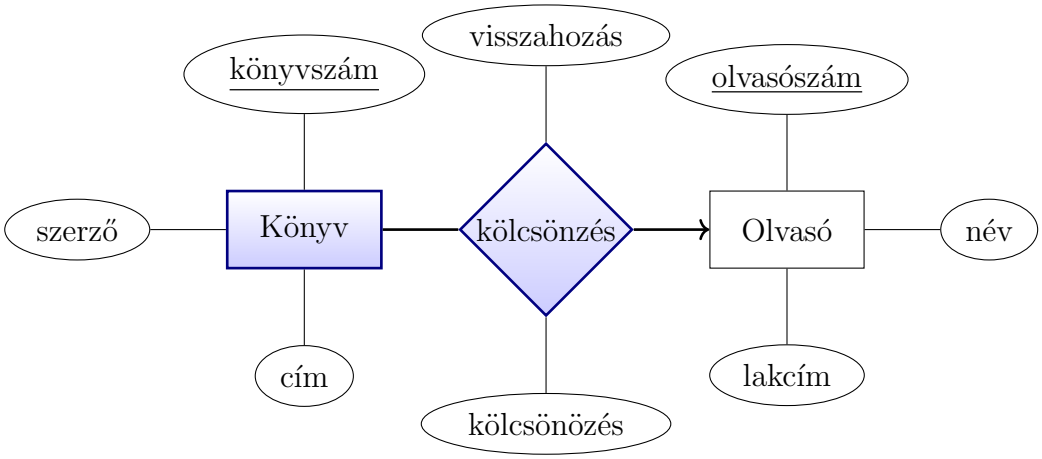
- egyedhalmaz séma ⇒ relációséma
- tulajdonságok ⇒ attribútumok
- (szuper)kulcs ⇒ (szuper)kulcs
- egyedhalmaz előfordulása ⇒ reláció
- egyed ⇒ $e(A_1) \dots e(A_n)$ sor
- $R(E_1, \dots, E_p, A_1, \dots, A_q)$ kapcsolati séma (E_i egyedhalmaz, A_j tulajdonság)

$$\Downarrow$$

$R(K_1, \dots, K_p, A_1, \dots, A_q)$ relációséma (K_i az E_i (szuper)kulcsa)

Átalakítás

- Minden nem gyenge egyedhalmazhoz létrehozunk egy relációt ugyanezzel a névvel és attribútum halmazzal.
- Kapcsolatokat nem tulajdonságként, hanem külön relációban ábrázoljuk. Itt az attribútumok a kapcsolatban résztvevő egyedhalmazok kulcsai lesznek. Valamint a kapcsolat attribútumai (ha vannak).
    - Amennyiben két attribútum neve megegyezne, az egyiket értelemszerűen át kell neveznünk.
- Gyenge egyedhalmazok esetén a relációnak tartalmaznia kell a gyenge egyedhalmaz attribútumait, valamint azokat a más egyedhalmazhoz tartozó attribútumokat, amelyek segítettek kialakítani a kulcsot.
- Gyenge egyedhalmaz kapcsolatait is relációkká alakítjuk, úgy hogy a kapcsolat mindkét oldalán lévő egyedhalmazok kulcsait attribútumként kezeljük.



- KÖNYV(könyvszám, szerző, cím)
- OLVASÓ(olvasószám, név, lakcím)
- KÖLCSÖNZÉS(könyvszám, olvasószám, kivétel, visszahozás)

Összetett attribútumok leképezése: Például ha a lakcímet (helység, utca, házszám) struktúrában akarjuk kezelni, akkor fel kell venni a sémába mindet attribútumként.

- OLVASÓ(olvasószám, név, helység, utca, házszám)

Többértékű attribútumok leképezése:

- Megadás egyértékűként:
  Például egy több szerzős könyvnél egy mezőben soroljuk fel az összeset.
  Nem túl jó megoldás, mert nem lehet a szerzőket külön kezelni, és esetleg nem is fér el mind a mezőben.
  KÖNYV(9635451903, Adatbázis rendszerek - Alapvetés, **{Jeffrey D. Ullman, Jennifer Widom}**)
- Megadás többértékűként:
    - *Sorok többszörözése*: Felveszünk annyi sort, ahány szerző van. (⇒ redundancia)
      KÖNYV(9635451903, Adatbázis rendszerek - Alapvetés, Jeffrey D. Ullman)
      KÖNYV(9635451903, Adatbázis rendszerek - Alapvetés, Jennifer Widom)

- *Új tábla hozzáadása*: A KÖNYV(könyvszám, szerző, cím) sémát az alábbi két sémával helyettesítjük:
  - KÖNYV(könyvszám, cím),
  - SZERZŐ(könyvszám, szerző)
- *Sorszámozás*: Ha nem mindegy a szerzők sorrendje, akkor az előző megoldásban (új tábla) ki kell egészíteni a szerző táblát egy sorszám mezővel.
  - KÖNYV(könyvszám, cím),
  - SZERZŐ(könyvszám, **sorszám**, szerző)

Kapcsolatok leképezése

- **egy-egy**: Az egyik sémát (tetszőleges, hogy melyiket) bővítjük a másik kulcsával és a kapcsolat attribútumaival.

  Például: Ha egy olvasónak egyszerre csak egy könyvet adnak ki, akkor a kölcsönzés 1:1 kapcsolatot jelent. Ilyenkor a KÖLCSÖN sémában a könyvszám és az olvasószám egyaránt kulcs. Továbbá, a visszahozás attribútumra nincs szükségünk, mivel a könyv visszahozásával a könyv-olvasó kapcsolat megszűnik.

  Tehát, a

  - KÖLCSÖN (könyvszám, olvasószám, kivétel) vagy a
  - KÖLCSÖN (könyvszám, olvasószám, kivétel) sémát vehetjük fel a kapcsolathoz.

  A KÖLCSÖN sémát az azonos kulcsú sémába olvasztva a

  - KÖNYV(könyvszám, szerző, cím, **olvasószám, kivétel**)
  - OLVASÓ(olvasószám, név, lakcím)

  vagy a

  - KÖNYV(könyvszám, szerző, cím)
  - OLVASÓ(olvasószám, név, lakcím, **könyvszám, kivétel**)

  adatbázissémákat kapjuk.

- **egy-sok**: az N oldali egyedhez tartozó sémát bővítjük az 1 oldali egyed kulcsával.

  Például: Ha egy olvasó több könyvet is kikölcsönözhet, akkor az olvasó-könyv kapcsolat 1:N típusú. Ekkor a KÖLCSÖN sémában csak a könyvszám lehet kulcs, ezért a KÖLCSÖN sémát csak a KÖNYV sémába olvaszthatjuk:

  - KÖNYV(könyvszám, szerző, cím, olvasószám, kivétel)
  - OLVASÓ(olvasószám, név, lakcím)

- **sok-sok**: új sémát veszünk fel (benne: egyedek kulcsai, kapcsolat attribútumai).

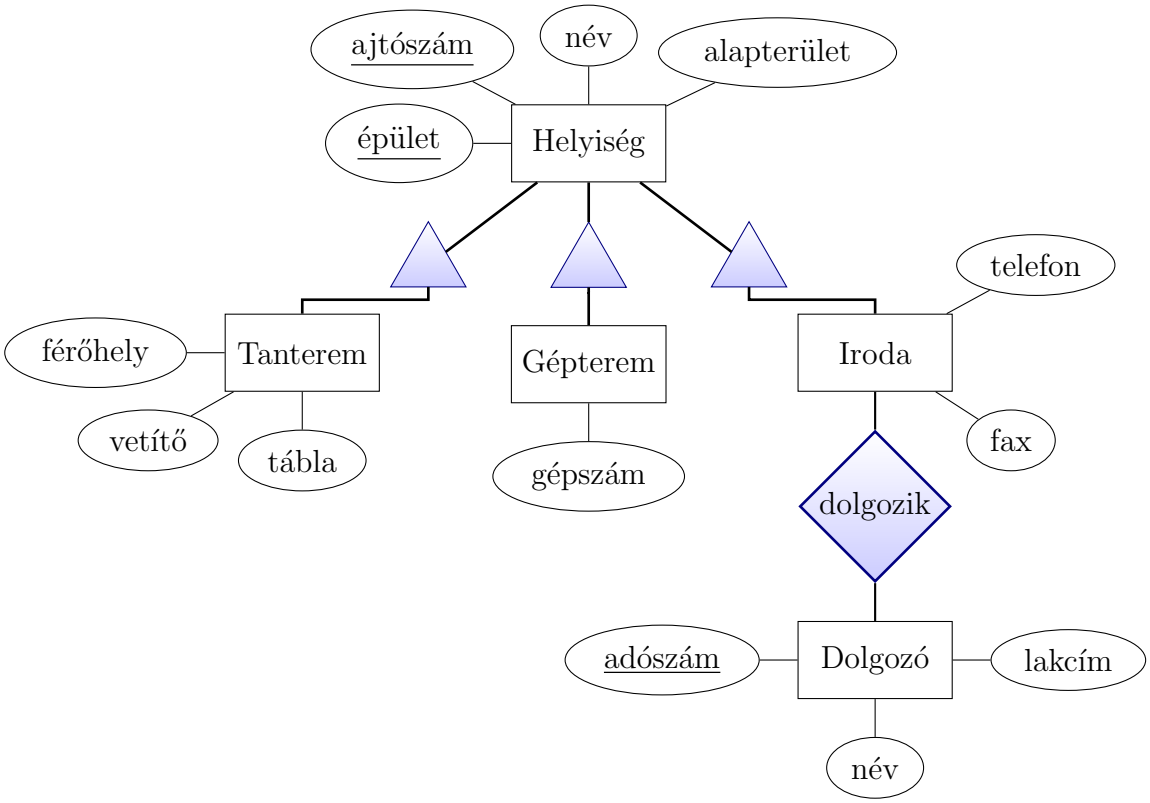
  Például: Ha az egyes könyvek korábbi kölcsönzéseit is nyilvántartjuk, akkor nem csak egy olvasóhoz tartozhat több könyv, hanem egy könyvhöz is több olvasó (N:M kapcsolat), sőt adott olvasó adott könyvet egymás után többször is kikölcsönözhet.

Ezért a KÖLCSÖN sémában könyvszám, kivétel vagy könyvszám, visszahozás a kulcs, a KÖLCSÖN táblát most sem a KÖNYV, sem az OLVASÓ táblába nem tudjuk beolvasztani.

Az adatbázisséma ezért a következő:

- KÖNYV(<u>könyvszám</u>, szerző, cím)
- OLVASÓ(<u>olvasószám</u>, név, lakcím)
- KÖLCSÖN(<u>könyvszám</u>, olvasószám, kivétel, visszahozás)

Specializáló kapcsolatok átírása



Osztályhierarchia átírása

(1) Minden altípushoz külön tábla felvétele, egy egyed csak egy táblában szerepel. Az altípusok örökölik a főtípus attribútumait.

(*Objektumorientált stílusú reprezentálás*)

- HELYISÉG(<u>épület</u>, <u>ajtószám</u>, *név, alapterület*)
- TANTEREM(<u>épület</u>, <u>ajtószám</u>, *név, alapterület*, férőhely, tábla, vetítő)
- GÉPTEREM(<u>épület</u>, <u>ajtószám</u>, *név, alapterület*, gépszám)
- IRODA(<u>épület</u>, <u>ajtószám</u>, *név, alapterület*, telefon, fax)
- DOLGOZÓ(<u>adószám</u>, név, lakcím, épület, ajtószám)

Hátrányai:

- Kereséskor gyakran több táblát kell vizsgálni (ha például a D épület 803. sz. terem alapterületét keressük).
- Kombinált altípus (például számítógépes tanterem) csak új altípus felvételével kezelhető.

(2) Minden altípushoz külön tábla felvétele, egy egyed több táblában is szerepelhet. A főtípus táblájában minden egyed szerepel, és annyi altípusében ahánynak megfelel. Az altípusok a főtípustól csak a kulcs-attribútumokat öröklik.

(*E/K stílusú reprezentálás*)

- HELYISÉG(épület, ajtószám, név, alapterület)
- TANTEREM(épület, ajtószám, férőhely, tábla, vetítő)
- GÉPTEREM(épület, ajtószám, gépszám)
- IRODA(épület, ajtószám, telefon, fax)
- DOLGOZÓ(adószám, név, lakcím, épület, ajtószám)

Hátrányai:

- Előfordulhat, hogy több táblában kell keresni (például: ha a tantermek nevére és férőhelyére vagyunk kíváncsiak).

(3) Egy közös tábla felvétele, az attribútumok uniójával. Az aktuálisan értékkel nem rendelkező attribútumok NULL értékűek.

(*Reprezentálás nullértékekkel*)

- HELYISÉG(épület, ajtószám, név, alapterület, férőhely, tábla, vetítő, gépszám, telefon, fax)
- DOLGOZÓ(adószám, név, lakcím, épület, ajtószám)

Hátrányai:

- Az ilyen egyesített táblában általában sok NULL attribútumérték szerepel.
- Elveszíthetjük a típusinformációt (például: ha a gépteremnél a gépszám nem ismert és ezért NULL, akkor a gépterem lényegében az egyéb helyiségek kategóriájába kerül).

Relációs algebra, SQL

Relációs algebra: Az algebra szó a matematikában azt a diszciplínát jelöli, amely egy halmazon értelmezett műveletek tulajdonságait vizsgálja. A mi esetünkben a műveletek a relációkon értelmezettek, így innen származik a relációs algebra kifejezése. A műveletek tehát relációkon értelmezettek és ami nagyon fontos és lényeges, hogy relációkat is adnak eredményül. Tehát egy lekérdezés eredménye egy újabb relációt szolgáltat, vagyis a relációs algebrai műveletek nem vezetnek ki a relációk halmazából, a kapott eredmény szintén az adatbázis részének tekinthető.

Relációs algebrai műveletek

Halmazműveletek

Legyen $R(A_1, \ldots, A_n)$ és $S(B_1, \ldots, B_m)$ tetszőleges relációséma és $n = m$, valamint

$$dom(A_i) = dom(B_i) \qquad (\forall i)$$

$R \bigcup S$ (R és S ***uniója***): $R \bigcup S = \mathbf{F}(C_1, \ldots, C_n)$, ahol

$$F := \left\{ f_i \;\middle|\; f_i \in R \vee f_i \in S \quad \wedge \quad (f_i \neq f_j \quad \forall i, j \in n,\ i \neq j) \right\}$$

- R, S és $\mathbf{F}$ azonos sémájú.
- $\mathbf{F}$ nem tartalmaz azonos sorokat. (Ez gyakorlatban eltérhet)
- $\left|R \bigcup S\right| \leq \left|R\right| + \left|S\right|$
- Nem feltétlenül örököl típusneveket vagy attribútumneveket.
- *SQL*: SELECT * FROM R **UNION** SELECT * FROM S;

Példa:

$R :=$

A	**B**
0	0
0	1

$S :=$

A	**B**
0	0
1	0

$R \bigcup S :=$

A	**B**
0	0
0	1
1	0

$R \setminus S$ (R és S ***különbsége***): $R \setminus S = \mathbf{F}(C_1, \ldots, C_n)$, ahol

$$F := \left\{ f_i \;\middle|\; f_i \in R \wedge f_i \notin S \right\}$$

- R azon sorait tartalmazza, amelyeket S nem tartalmazza.
- $\left|R \setminus S\right| \leq \left|R\right|$
- Fontos: $R \setminus S \neq S \setminus R$ (nem kommutatív).
- Örökli a típusneveket és az attribútum neveket, mert $R \setminus S \subseteq R$.
- *SQL*: SELECT * FROM R **MINUS** SELECT * FROM S;

Példa:

$R :=$

A	**B**
0	0
0	1

$S :=$

A	**B**
0	0
1	0

$R \setminus S :=$

A	**B**
0	1

$R \times S$ (R és S ***Descartes-szorzata***): Az R és S minden sora párban összefűződik, az első reláció minden sorához hozzáfűzzük a második reláció minden sorát.

- R minden sorát tartalmazza S minden sorával az összes kombinációban.
- $R \times S$ sémája R és S sémájának egyesítése.
- Ha R, S sémáiban nincs közös attribútum, akkor $R \bowtie S = R \times S$
- $\left|R \times S\right| = \left|R\right| * \left|S\right|$
- *SQL*: SELECT * FROM R **CROSS JOIN** S vagy SELECT * FROM R**,** S;

Példa:

$R :=$

A	**B**
0	0
0	1

$S :=$

B	**C**
0	0
1	0

$R \times S :=$

A	**R.B**	**S.B**	**C**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0

Az alapműveletekhez az *unió* és *különbség* és a Descartes-szorzat tartozik.
A *metszet* műveletet származtatjuk:

$$R \bigcap S = R \setminus (R \setminus S)$$

$R \bigcap S$ (R és S ***metszete***): $R \bigcap S = \mathbf{F}(C_1, \ldots, C_n)$, ahol

$$F := \left\{ f_i \;\middle|\; f_i \in R \wedge f_i \in S \quad \wedge \quad (f_i \neq f_j \quad \forall i, j \in n,\ i \neq j) \right\}$$

- R azon sorait tartalmazza, amelyeket S is tartalmazza.
- R, S és $\mathbf{F}$ azonos sémájú.
- $\mathbf{F}$ nem tartalmaz azonos sorokat.
- $\left|R \cap S\right| = \left|R\right| + \left|S\right|$
- *SQL*: SELECT * FROM R **INTERSECT** SELECT * FROM S;

Példa:

$R :=$

A	**B**
0	0
0	1

$S :=$

A	**B**
0	0
1	0

$R \bigcap S :=$

A	**B**
0	0

Vetítés (projekció)

$\mathbf{\Pi}_{A_{i_1},\ldots,A_{i_k}}(R)$ (R ***vetítése***): R relációból olyan új relációt hoz létre, amelyik csak R bizonyos attribútumait tartalmazza:

$$\Pi_{A_{i_1},\ldots,A_{i_k}}(R) := \left\{ t.A_{i_1}, t.A_{i_2},\ \ldots,\ t.A_{i_k} \middle|\ t \in R \right\}$$

- Számít az attribútumok sorrendje. (rendezett lista)
- $\left|\Pi_{A_{i_1},\ldots,A_{i_k}}(R)\right| < \left|R\right|$
- Örökli a típusneveket és az attribútum neveket.

Példa:

$R :=$

ID	Név	Kor
1	Péter	3
2	László	37

$\Pi_{\text{ID,Név}}(R) :=$

ID	Név
1	Péter
2	László

Kiválasztás (szelekció)

$\boldsymbol{\sigma}_C(R)$ (R ***szelekciója***): R reláció azon sorai, amelyekre az C feltétel igaz.

- C feltétel lehet:
  - *Elemi*: $A_i\ \theta\ A_j$, $A_i\ \theta\ d$, ahol d konstans, $\theta \in \left\{ =, <, >, \neq, \leq, \geq \right\}$.
  - *Összetett*: ha B_1, B_2 feltétel, akkor $\neg B_1, B_1 \bigcap B_2, B_1 \bigcup B_2$ és a zárójelezések is feltétel.
- C feltétel ***nem tartalmazhat függvényeket***: $C = A + B < 5$
- Összetett feltételek átírhatók elemi feltételeket használó kifejezésekké
  - $\sigma_{F_1 \wedge F_2}(R) \cong \sigma_{F_1}(\sigma_{F_2}(R)) \cong \sigma_{F_2}(\sigma_{F_1}(R))$
  - $\sigma_{F_1 \vee F_2}(R) \cong \sigma_{F_1}(R) \bigcup \sigma_{F_2}(R)$
  - $\neg(F_1 \wedge F_2) \Rightarrow (\neg F_1) \vee (\neg F_2)$
  - $\neg(F_1 \vee F_2) \Rightarrow (\neg F_1) \wedge (\neg F_2)$
  - $\neg(A < B) \Rightarrow (A \geq B)$
- Örökli a típusneveket és az attribútum neveket, mivel $\sigma_C(R) \subseteq R$.
- $\left|\sigma_C(R)\right| < \left|R\right|$
- *SQL*: SELECT * FROM R **WHERE** <**FELTÉTEL**>;

Példa:

$R :=$

ID	Név	Kor
1	Péter	3
2	László	37

$\sigma_{\text{Kor}>10}(R) :=$

ID	Név	Kor
2	László	37

Természetes összekapcsolás

Szorzás jellegű műveletek közül csak ez alapművelet. Nő az attribútumok száma. A közös attribútumnevekre épül: $R \bowtie S$ azon sorpárokat tartalmazza R-ből illetve S-ből, amelyek R és S azonos attribútumain megegyeznek. $R \bowtie S$ típusa a két attribútumhalmaz uniója.

- Ha R, S sémái megegyeznek, akkor $R \bowtie S = R \bigcap S$
- *SQL*:
  - SELECT * FROM R NATURAL JOIN S vagy
  - SELECT DISTINCT A, R.B, C FROM R**,** S WHERE R.B = S.B;

Példa:

$R :=$

A	**B**
a	a
c	b
b	c

$S :=$

B	**C**
a	a
a	c
b	d
e	d

$R \bowtie S :=$

A	**R.B**	**C**
a	a	a
a	a	c
c	b	d

Szorzásjellegű műveletnél tekinthetjük a *direkt-szorzatot* alapműveletnek, de a természetes összekapcsolást használják. A direkt-szorzat (vagy szorzat, Descartes-szorzat) esetén természetesen nem fontos az attribútumok egyenlősége. A két vagy több reláció azonos nevű attribútumait azonban meg kell különböztetni egymástól (átnevezéssel).

Átnevezés

$\rho_{B(D_1,\dots,D_k)}(R)$ (R ***átnevezése***): R reláció sémájának átnevezése B-re, valamint R attribútumainak átnevezése $A_1,\dots,A_n$-ről $D_1,\dots,D_n$-re.

- $\rho_{B(D_1,\dots,D_n)}(R)$ sémája $B(D_1,\dots,D_n)$
- $\left|\rho_{B(D_1,\dots,D_n)}(R)\right| = \left|R\right|$
- SQL:
  - SELECT A, B, C **D** FROM R **B** vagy
  - SELECT A, B, C **AS D** FROM R **AS B**;

Példa:

$R :=$

ID	Név	Fizetés
1	Péter	300000
2	László	507000

$\rho_{B(\text{ID,Név,Jövedelem})}(R) :=$

ID	Név	Jövedelem
1	Péter	300000
2	László	507000

$:= B$

A relációs algebrában a $\bigcup, \setminus, \times(\bowtie), \Pi, \sigma, \rho$ alapműveletek találhatóak. Ez egy *minimális készlet*, vagyis bármelyiket elhagyva az a többivel nem fejezhető ki. A kivonás egy kivétel, mert nem fejezhető ki a többi alapművelettel.

$R \div S$ (***osztás***): maradékos osztás mintájára. R és S sémája $R(A_1,...,A_n,B_1,...,B_m)$, illetve $S(B_1,...,B_m)$, $Q = R \div S$ sémája $Q(A_1,...,A_n)$. $R \div S$ a legnagyobb (legtöbb sort tartalmazó) reláció, amelyre $(R \div S) \times S \subseteq R$.

Relációs algebrában: $R(A,B) \div S(B) = \Pi_{A_1,\dots,A_n}(R) – \Pi_{A_1,\dots,A_n}(\Pi_{A_1,\dots,A_n}(R) \times S \setminus R)$

Példa:

$R :=$

A	**B**
F	A
F	B
F	C
M	A
M	B
K	A
K	D
N	E

$S :=$

B
A
B

$R \div S :=$

A
F
M

Relációs algebra kiterjesztése

A relációkra vonatkozó műveleteket kiterjesztjük halmazokról multihalmazokra (sorok ismétlődése megengedett).

$\boldsymbol{\delta}(R)$ (***ismétlődések megszüntetése***): Multihalmazból halmazt csinál.

- Örökli a típusneveket és az attribútum neveket, mivel $\delta(R) \subseteq R$.
- $\left|\delta(R)\right| \leq \left|R\right|$
- SQL: SELECT **DISTINCT** * FROM R;

Példa:

$R :=$

Név	Jövedelem
Péter	300000
László	507000
László	507000

$\delta(R) :=$

Név	Jövedelem
Péter	300000
László	507000

$\boldsymbol{\Pi}_L(R)$ (***vetítési művelet kiterjesztése***): A normál vetítési művelet kiterjesztett változata.

A L tartalmazhatja:

- R egy attribútumát,
- E $\rightarrow$ z, ahol E az R reláció attribútumaira vonatkozó (konstansokat, aritmetikai műveleteket, függvényeket tartalmazó kifejezés), z pedig az E kifejezés által számolt az eredményekhez tartozó új attribútum nevét jelöli.

Példa:

$R :=$

A	B
1	2
2	3
4	5

$\Pi_{A+B \rightarrow C}(R) :=$

C
3
5
9

$\boldsymbol{\gamma}_L(R)$ (***Összesítő műveletek és csoportosítás***): A csoportosítást, a csoportokon végezhető összesítő függvényeket (AVG, SUM, COUNT, MIN. MAX, stb.) reprezentálja a művelet. Itt az L valamennyi eleme a következők egyike:

- R olyan attribútuma, amely szerepel a GROUP BY záradékban, egyike a csoportosító attribútumoknak.
- R egyik attribútumára alkalmazott összesítő operátor.
    - Ha az összesítés eredményére névvel szeretnénk hivatkozni, akkor nyilat és új nevet használunk.

Értelmezése, kiértékelés: R sorait csoportokba osztjuk. Egy csoport azokat a sorokat tartalmazza, amelyek a listán szereplő csoportosítási attribútumokhoz tartozó értékei megegyeznek.

- Vagyis ezen attribútumok minden egyes különböző értéke egy csoportot alkot.

Minden egyes csoporthoz számoljuk ki a lista összesítési attribútumaira vonatkozó összesítéseket. Az eredmény minden egyes csoportra egy sor:

1. a csoportosítási attribútumok és
2. az összesítési attribútumra vonatkozó összesítése (az adott csoport összes sorára)

$\boldsymbol{\tau}_{A,\ldots,A_n}(R)$ (***rendezés***): Először A_1 attribútum szerint rendezzük R sorait. Majd azokat a sorokat, amelyek értéke megegyezik az A_1 attribútumon, A_2 szerint, és így tovább.
Ez az egyetlen olyan művelet, amelynek az eredménye se nem halmaz, se nem multihalmaz, hanem egy rendezett lista.

- $\tau(R)| = \left|R\right|$
- *SQL*: SELECT A, B FROM R **ORDER BY** B;

Példa:

$R :=$

A	**B**
1	2
3	4
5	2

$\tau(R) :=$

A	**B**
1	2
5	2
3	4

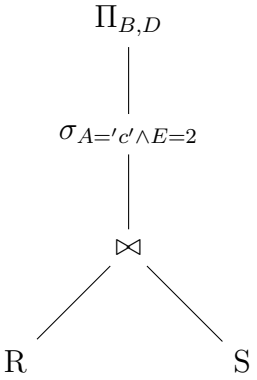
Algebrai kifejezés kiértékelése

Összetett kifejezést kívülről befelé haladva átírjuk kiértékelő fává, ahol a levelek elemi kifejezések.

Legyen R, S az $R(A, B, C), S(C, D, E)$ séma feletti reláció

$$\Pi_{B,D}\sigma_{A='c' \wedge E=2}(R \bowtie S)$$

A kifejezéshez tartozó kifejezésfa (kiértékelése alulról felfelé történik)



SQL

Az SQL kifejezés a Structured Query Language rövidítése, ami strukturált lekérdező nyelvet jelent. A relációs adatbázis-kezelő rendszerekben tárolt adatok kezelésére fejlesztették ki.

Az SQL széles körben népszerű, mert a következő előnyöket kínálja. Lehetővé teszi a felhasználók számára:

- az adatok leírását.
- a relációs adatbázis-kezelő rendszerek adatainak elérését.
- adatbázisokat és táblázatokat hozzanak létre vagy dobjanak el.
- az adatbázisban lévő adatok definiálását és az adatok manipulálását.
- nézetek létrehozását egy adatbázisban.
- táblák, eljárások és nézetek engedélyeinek beállítását.

Lehetővé teszi más nyelveken belüli beágyazást SQL modulok, könyvtárak és előfordítók használatával.

Az első verzió alapjait az IBM fektette le a 70-es években. A nyelv elvi alapjait a relációs adatmodell fogalma, az Edgar F. Codd által leírt 12 szabálya adta. 80-as években az ANSI és ISO is szabványként fogadta el az SQL nyelvet.

Az SQL nyelvi elemeket négyféleképp csoportosíthatjuk:

- ***Adatdefiníciós, DDL*** (Data Definition Language)
- ***Adatmanipulációs, DML*** (Data Manipulation Language)
- ***Adatvezérlő, DCL*** (Data Control Language)
- ***Lekérdező, DQL*** (Data Query Language)

Alap tulajdonságok:

- A nyelvben az egyes utasításokat pontosvessző válassza el egymástól.
- Az SQL case insensitive, tehát a kis és nagy betűk nincsenek megkülönböztetve egymástól.
- A parancsok tetszőlegesen tagolhatók (szóköz, tabulátor) és több sorba írhatók.
- A szöveg konstansokat szimpla aposztrófok jelölik pl.: 'SQL'

SQL-ben a relációt táblának nevezik, amiből alapvetően három féle létezik:

- TABLE (alaptábla, permanens),
- VIEW (nézettábla),
- WITH utasítással (átmeneti munkatábla).

DDL (Data Definition Language)

Adatleíró rész az adatbázis objektumainak a leírására és megváltoztatására.

- CREATE: Adatbázis-objektum létrehozása.

```
CREATE DATABASE Nyilvantartas;

CREATE TABLE alkalmazottak(
    id INT(11) NOT NULL,
    nev VARCHAR(70) NOT NULL,
    szuletesi_datum DATE  NOT NULL,
    irsz NUMBER(7) NOT NULL,
    varos VARCHAR(40) NOT NULL,
    cim VARCHAR(60) NOT NULL
);

CREATE VIEW alk_varosok AS
    SELECT id, nev, varos
    FROM alkalmazottak;
```

- ALTER: Adatbázis-objektum módosítása.

```
ALTER TABLE alkalmazottak ADD cim VARCHAR(80);
```

- DROP: Adatbázis-objektum megszüntetése. Minden, ami ehhez az elemhez kapcsolódott, elérhetetlen lesz.

```
DROP TABLE alkalmazottak;
DROP VIEW alk_varosok;
```

- TRUNCATE TABLE: Tábla összes sorának törlése a tábla szerkezetének megtartásával.

Megszorítások

A *megszorítás* adatelemek közötti kapcsolat, amelyet az adatbázis-rendszernek fent kell tartania. Lehet kulcs megszorítás, értékekre, sorokra vonatkozó. Itt alapvetően az adatbázis bármely módosítása előtt ellenőrizni kell. Egy okos rendszer felismeri, hogy mely változtatások, mely megszorításokat érinthetnek.

Megszorítás megadása CONSTRAINTS kulcsszóval történik. Saját megszorítás megadása: CREATE ASSERTION <név> CHECK (<feltétel>).

- UNIQUE: Az adott oszlopban minden érték egyedi. Hatására index állomány jön létre. Felvehet NULL értéket.

```
CREATE TABLE Persons (
    ...
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

- PRIMARY KEY: Az adott oszlop a tábla elsődleges kulcsa lesz, megköveteli az adott oszlopban az értékek egyediségét. Oszlop-megszorításokban csak akkor tudunk elsődleges kulcsot definiálni, ha az elsődleges kulcs csak egy oszlopból áll. Ilyen oszlop csak egy lehet a táblában. Hatására mindig létrejön egy index állomány. Nem lehet NULL az értéke.

```
CREATE TABLE Persons (
    ...
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

- FOREIGN KEY - Idegen kulcs egy másik tábla megfelelő oszlopára.
    - REFERENCES kapcsolttabla(kapcsoltoszlop) [ON-feltételek]: Az adott oszlop idegen kulcs-hivatkozást tartalmaz a *kapcsolttabla kapcsoltoszlop*ára. A *kapcsolttabla kapcsoltoszlop*a csak elsődleges, vagy egyedi kulcs lehet.

  Idegen kulcs (R-ről S-re) megszorítást meg kell őrizni, ez kétféleképpen sérülhet:

  1. Egy R-be történő beszúrásnál vagy R-ben történő módosításnál S-ben nem szereplő értéket adunk meg.
  2. Egy S-beli törlés vagy módosítás ,,lógó" sorokat eredményez R-ben. Védeni többféleképpen lehet: alapértelmezetten nem hajtja végre, tovább gyűrűzésnél igazítjuk a tábla értékeit a változáshoz, SET NULL-nál pedig az érintett sorokat NULL-ra állítjuk.

```
CREATE TABLE Orders (
    ...
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
        REFERENCES Persons(PersonID)
);
```

- DEFAULT: Alapértelmezett érték, mely akkor kerül az új sor adott oszlopába, ha nem adunk meg a beszúrás során implicit értéket.

```
CREATE TABLE Orders (
    ...
    OrderDate date DEFAULT GETDATE()
);
```

- CHECK: Az oszlopra vonatkozó logikai kifejezést adhatunk meg, a feltételben csak az oszlop szerepelhet. A beszúrás, módosítás csak akkor történik meg, ha a kifejezés értéke igaz lesz.

```
CREATE TABLE Persons (
    ...
    Age int,
    CONSTRAINT CHK_Person CHECK (Age>=18)
);
```

- NOT NULL — NULL: Az adott oszlopban szerepelhet-e NULL érték.

```
CREATE TABLE Persons (
    ...
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    ...
);
```

DQL (Data Query Language)

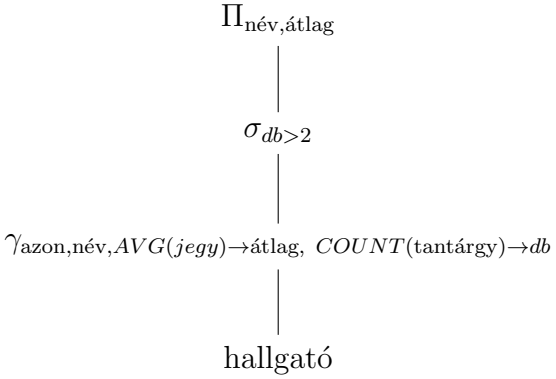
Relációs algebrai kifejezések felírása SELECT-tel:

- SELECT lista FROM táblák szorzata WHERE felt. $= \Pi_{lista}\big(\sigma_{felt.}(\text{táblák szorzata})\big)$
- Halmazműveletek: **UNION**, **EXCEPT**/**MINUS**, **INTERSECT**. Multihalmazból halmaz lesz. ALL kulcsszóval megmarad a multihalmaz.
- Átnevezés: oszlopnév után szóközzel odaírjuk az új nevet vagy AS új név.

Kiterjesztett műveletek:

- Rendezés: **ORDER BY**, minden más záradék után következik, csökkenő, növekvő sorrend.
- Ismétlődések megszüntetése: SELECT **DISTINCT** ...
- Összesítések (aggregálás): **SUM**, **COUNT**, **MIN**, **MAX**, **AVG** a SELECT záradékban. **COUNT**(*) az eredmény sorainak száma. Ha összesítés is szerepel a lekérdezésben, a SELECT-ben felsorolt attribútumok vagy egy összesítő függvény paramétereként szerepelnek, vagy a *GROUP BY* attribútumlistájában is megjelennek.
- Csoportosítás: **GROUP BY**.
- Csoportok szűrése: **HAVING**. Csoportokat szűr, nem egy-egy sort. Az alkérdésre nincs megszorítás. Viszont az alkérdésen kívül csak olyan attribútumok szerepelhetnek, amelyek: vagy csoportosító attribútumok, vagy összesített attribútumok. (Azaz ugyanazok a szabályok érvényesek, mint a SELECT záradéknál).

```
SELECT név, AVG(jegy) AS átlag
FROM hallgató
GROUP BY azon, név
HAVING COUNT(tantárgy) > 2;
```



WHERE záradéknál SQL-es specialitások

- Átírható relációs algebrába: **BETWEEN** ... **AND** ..., **IN**(értékhalmaz)
- Nem írható át relációs algebrába: **LIKE** karakterláncok összehasonlításánál, **IS NULL** (ismeretlen vagy nem definiált érték) [emiatt 3-értékű logika SQL-ben: true, false, unknown]

SELECT utasítás több részből áll, a részeket záradékoknak nevezzük.

Három alapvető záradék van:

- SELECT lista (3.) – milyen típusú sort szeretnénk az eredményben látni? * jelentése: minden attribútum.
- FROM Relációnév (1.) – a tábla neve vagy relációk (táblák) összekapcsolása, illetve szorzata
- WHERE feltétel (2.) – milyen feltételeknek eleget tevő sorokat kiválasztani?

Több séma összekapcsolása esetén, ha egy attribútumnév több sémában is előfordul, akkor kell hozzá a séma is a hivatkozásnál: R.A (R reláció A attribútuma).

Alkérdéseket is használhatunk SQL-ben. Ekkor az alkérdést zárójelbe tesszük.

- FROM záradékban ilyen módon ideiglenes táblát is létrehozhatunk, ekkor többnyire a sorváltozó nevét is meg kell adni hozzá.
- WHERE záradékban az alkérdés eredménye lehet:
  - egy skalárérték, vagyis mintha egy konstans lenne
  - skalár értékekből álló multihalmaz, logikai kifejezésekben használható: EXISTS, [NOT] IN, ANY/ALL (pl x > ANY (alkérdés)).
  - teljes többdimenziós tábla, használható: EXISTS, [NOT] IN

Az alkérdés nem korrelált, ha önállóan kiértékelhető, külső kérdés közben nem változik. Korrelált, ha többször kerül kiértékelésre, alkérdésen kívüli sorváltozóból származó értékadással.

Teljes SELECT utasítás (záradékok sorrendje nem cserélhető fel)

```
SELECT [DISTINCT] ...
FROM ...
[WHERE ...]
[GROUP BY ...
    [HAVING ...]]
[ORDER BY ...]
```

Összekapcsolások

- Descartes-szorzat: R CROSS JOIN S, vagy ,,R,S”
- Természetes összekapcsolás: R NATURAL JOIN S
- Théta-összekapcsolás: R INNER JOIN S ON <feltétel>
- Külső összekapcsolás: R {LEFT | RIGHT | FULL} OUTER JOIN S. LEFT az R lógó sorait őrzi meg, RIGHT az S-ét, FULL az összeset.

DCL (Data Control Language)

GRANT - a felhasználó hozzáférési jogosultságokat ad az adatbázishoz.
REVOKE - visszavonja a felhasználó hozzáférési jogosultságait.

DML (Data Manipulation Language)

A módosító utasítások nem adnak vissza eredményt, mint a lekérdezések, hanem az adatbázis tartalmát változtatják meg. Visszatérési értéke amennyiben van, akkor az a módosításban érintett sorok száma.

Három féle módosító utasítás létezik:

- INSERT - sorok beillesztése, beszúrása.
  - Egyetlen sor:
    - INSERT INTO R [(<attr. lista>)] VALUES (<konkrét értékek listája>);

    A tábla létrehozásánál, ha megadtunk default értékeket attribútumoknál, akkor ha nem adunk meg értéket hozzá, akkor a default lesz, egyébként NULL.
  - Több sor:
    - INSERT INTO R [(<attr. lista>)] (<alkérdés>);
- DELETE – sorok törlése:
  - DELETE [FROM] R [WHERE <feltétel>];
- UPDATE – sorok komponensei értékeinek módosítása:
  - UPDATE R SET <attribútum értékadások listája> WHERE <sorokra vonatkozó feltétel>

Tranzakciók

A tranzakció nem más, mint DML-utasítások sorozata, amelyek a munka egyik logikai egységét alkotják. A tranzakció utasításainak hatása együtt jelentkezik. A tranzakció sikeres végrehajtása esetén a módosított adatok véglegesítődnek, a tranzakcióhoz tartozó visszagörgetési szegmensek újra felhasználhatóvá válnak. Ha viszont valamilyen hiba folytán a tranzakció sikertelen (bármelyik utasítása nem hajtható végre), akkor visszagörgetődik, és az adatbázis tranzakció előtti állapota nem változik meg. Az Oracle lehetőséget biztosít egy tranzakció részleges visszagörgetésére is.

Minden SQL utasítás egy tranzakció része. A tranzakciót az első SQL utasítás indítja el. Ha egy tranzakció befejeződött, a következő SQL utasítás új tranzakciót indít el.

A tranzakció explicit véglegesítésére a COMMIT utasítás szolgál.
A COMMIT a tranzakció által okozott módosításokat átvezeti az adatbázisba és láthatóvá teszi azokat más munkamenetek számára, felold minden – a tranzakció működése közben elhelyezett – zárat és törli a mentési pontokat.

A SAVEPOINT utasítással egy tranzakcióban mentési pontokat helyezhetünk el. Ezek a tranzakció részleges visszagörgetését szolgálják. Az utasítás alakja:

- SAVEPOINT mentési_pont;

A név nemdeklarált azonosító, amely a tranzakció adott pontját jelöli meg. A név egy másik SAVEPOINT utasításban felhasználható. Ekkor a később kiadott utasítás hatása lesz érvényes.

A visszagörgetést a ROLLBACK utasítás végzi, alakja:

- ROLLBACK [TO [SAVEPOINT] mentési_pont];

Az egyszerű ROLLBACK utasítás érvényteleníti a teljes tranzakció hatását (az adatbázis változatlan marad), oldja a zárakat és törli a mentési pontokat. A tranzakció befejeződik.

A TO utasításrésszel rendelkező ROLLBACK a megadott mentési pontig görgeti vissza a tranzakciót, a megadott mentési pont érvényben marad, az azt követők törlődnek, a mentési pont után elhelyezett zárak feloldásra kerülnek és a tranzakció a megadott mentési ponttól folytatódik.

A COMMIT utasítás végrehajtása után a tranzakció véglegesnek tekinthető. A tranzakció módosításai véglegesítődnek.
A ROLLBACK utasítás esetén a tranzakció abortál, azaz az összes utasítás visszagörgetésre kerül.

- Alapvető hibák, mint például 0-val való osztás esetén a ROLLBACK automatikus függetlenül, hogy volt-e rá explicit utasítás.

Az SQL négy elkülönítési szintet definiál, amelyek megmondják, hogy milyen interakciók engedélyezettek az egy időben végrehajtódó tranzakciók közt.

1. SERIALIZABLE: A tranzakciók ütemezése konfliktus ekvivalens egy soros ütemezéssel. Azaz a tranzakciók úgy ütemeződnek, mintha egymás után futnának le.
2. READ COMMITTED: Olvasáskor mindig a már rögzített (commitált) eredményt kapjuk. Ha fut egy tranzakció ami már változtatott az általunk kívánatos sorokon, akkor mi a régi eredményeket kapjuk. Ha sokan írják az adatbázist, itt lassulás következhet be, más folyamatok arra várnak, hogy az egész tábla frissítése befejeződjön.
3. REPEATABLE READ: Csak rögzített rekordokat olvasunk. Vagyis nem várunk a teljes tábla frissítésére, ami már rögzített rekord az olvasható is.
4. READ UNCOMMITTED: Olyan adatokat is olvashatunk, amelyek még nincsenek rögzítve. Piszkos olvasás. A tranzakció vége előtt már olvashatók a nem rögzített adatok.

Az SQL procedurális kiterjesztése (PL/SQL vagy PSM)

Amikor az SQL utasításokat egy alkalmazás részeként, programban használjuk, a következő problémák léphetnek fel:

- Osztott változók használata: közös változók a nyelv és az SQL utasítás között (ott használható SQL utasításban, ahol kifejezés használható).
- A típuseltérés problémája: Az SQL magját a relációs adatmodell képezi. Tábla – gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel?
  Három esetet különböztetünk meg attól függően, hogy a SELECT FROM [WHERE stb] lekérdezés eredménye skalárértékkel, egyetlen sorral vagy egy listával (multihalmazzal) tér-e vissza.
  1. SELECT eredménye egy skalárértékkel tér vissza, elemi kifejezésként használhatjuk.
  2. SELECT egyetlen sorral tér vissza
     - SELECT $e_1, \ldots, e_n$ INTO vált$_1, \ldots,$ vált$_n$
       A végrehajtásnál visszatérő üzenethez az SQL STATE változóban férhetünk hozzá.
  3. SELECT eredménye több sorból álló tábla, akkor az eredményt soronként bejárhatóvá tesszük, kurzorhasználatával.

Háromféleképpen is megközelíthetjük programozási szempontból:

1. SQL kiterjesztése procedurális eszközökkel, az adatbázis séma részeként tárolt kódrészekkel, tárolt modulokkal (pl. PSM = Persistent Stored Modules, Oracle PL/SQL).

2. Beágyazott SQL (sajátos előzetes beágyazás EXEC SQL. - Előfordító alakítja át a befogadó gazdanyelvre/host language, pl. C)

3. Hívásszintű felület: hagyományos nyelvben programozunk, függvénykönyvtárat használunk az adatbázishoz való hozzáféréshez (pl. CLI = call-level interface, JDBC, PHP/DB)

PSM: Persistent Stored Procedures. SQL utasítások és konvencionális elemek (if, while stb) keverékéből áll. Olyan dolgokat is meg lehet csinálni, amit önmagában az SQL-ben nem.

PL/SQL

A PL/SQL (Procedural Language/Structured Query Language) az Oracle által az SQL kiterjesztéseként kifejlesztett procedurális programozási nyelv (Ada alapokon).

A PL/SQL nem tartalmazza az SQL teljes utasításkészletét, csak azon utasításokat, melyek az adatkezelő eljárások során nagyobb szereppel rendelkeznek: SQL SELECT, INSERT, DELETE, UPDATE illetve OPEN, FETCH, CLOSE utasításait. Így a PL/SQL nyelvben lehetőség van az SQL adatkezelő utasításainak, a kurzor szerkezetnek és a tranzakciókezelő utasításoknak a használatára, de hiányoznak belőle például az adatdefiníciós és a védelmet szabályzó utasítások.

A PL/SQL nyelv viszont tartalmazza az alapvető vezérlési elemeket, így a WHILE ciklust és az IF elágazást is. A PL/SQL-ben lehetőség van saját memóriaváltozók létrehozására, melyekkel közbenső számítási eredmények tárolhatók. Igen erős a nyelvhez kapcsolódó hibakezelési komponens, s számos függvény segíti a rugalmas, hatékony programfejlesztést.

Mivel a PL/SQL alkalmazásával az adatkezelő utasítások nem egyesével végrehajtott SQL utasítások formájában kerül végrehajtásra az adatbáziskezelőhöz, ezért a végrehajtási sebesség is jelentősen javítható a PL/SQL segítségével. A PL/SQL eljárások feldolgozása ugyanis programegységekben, úgynevezett blokkokban történik. A blokkban helyet foglaló SQL utasítások együttesét hatékonyabban lehet optimalizálni, mint az egyenként végrehajtott SQL utasításokat.

A PL/SQL alkalmazható többek között az SQLPlus, SQLForms és más komponensekben is. E nyelv előnye, hogy független az alkalmazott konfigurációtól, operációs rendszertől, s csak a futó adatbáziskezelőtől függ a konkrét felépítése, formátuma.

Összefoglalóan a PL/SQL előnyei az alábbi pontokban adhatók meg:

- procedurális elemek és SQL ötvözése
- hatékonyság
- jobb integráció az adatbáziskezelő rendszerrel
- rugalmasság, hordozhatóság.

A következő konstrukciókat adja az SQL-hez:

- változók és típusok,
- vezérlési szerkezet,
- kurzorok, kurzorváltozók
- alprogramok, tárolt eljárások és függvények,
- kivételkezelés,
- triggerek
- objektumorientált eszközök.

A PL/SQL nyelv alapvető strukturális egysége a PL/SQL blokk. A legfőbb hasonlósága az eljárással az, hogy a PL/SQL blokk is adatdefiníciós, majd azt követő műveleti részből áll, s a végrehajtás egységét jelenti. A különbség legfontosabb elemi, hogy itt szintaktikailag külön szerepel egy hibakezelő rész, s a blokkok egymásba is ágyazhatók, ahol a beágyazott blokk a műveleti vagy hibakezelő részben szerepelhet.
A PL/SQL blokk szerkezete:

```
[címke]
[DECLARE deklarációs utasítások ]
BEGIN
    végrehajtandó utasítások
    [ EXCEPTION
        kivételkezelés ]
END [név];
```

A deklarációs rész külön válik a törzstől és opcionális, és a DECLARE kulcsszó csak egyszer szerepel a rész elején, nincs minden változó előtt. Értékadás := jellel.

Több új típus is van, pl NUMBER lehet INT van REAL. Egy attribútum típusára lehet hivatkozni is: R.x%TYPE. Létezik R%ROWTYPE is, ami egy tuple-t ad vissza. Az x tuple komponensének értékét így kapjuk meg: x.a. ELSEIF (PSM-ben) helyett ELSIF. LEAVE ciklus helyett EXIT WHEN <feltétel>.

Deklarációs rész tartalma lehet:

- Típus definíció
- Változó deklaráció
- Nevesített konstans deklaráció
- Kivétel deklaráció
- Kurzor definíció
- Alprogram definíció

A PL/SQL nem tartalmaz I/O utasításokat.

A DBMS_OUTPUT csomag segítségével üzenetet helyezhetünk el egy belső pufferbe. PUT_LINE eljárás üzenetet ír a pufferbe. A puffer tartalmát a SET SERVEROUTPUT ON utasítással jeleníthetjük meg a képernyőn.

Példa:

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

Tárolt eljárások (Stored Procedure)

A tárolt eljárások egy olyan PL/SQL blokkot jelentenek, amelyek paraméterezhetők, saját egyedi azonosító nevük van és az adatbázisban lefordított formában letárolásra kerülnek.
Ennek megoldásnak az előnye, hogy a PL/SQL blokk több helyről is elérhető, elég csak egyszer definiálni, s gyorsabb végrehajtást tesz lehetővé, mint az egyedileg elküldött PL/SQL blokk.

A eljárás deklarációja

```
CREATE PROCEDURE eljárás-név (
    paraméter-lista)
[DECLARE ... deklarációk]
BEGIN
    az eljárás utasításai;
END;
```

ahol a PL/SQL_blokk az eljárás törzse, s megfelel egy szabályos PL/SQL blokknak.
A paraméterlista elemei vesszővel vannak elválasztva egymástól, s minden elem

```
paraméternév jelleg adattipus
```

hármasból áll, melyben a *jelleg* arra utal, hogy kimenő vagy bejövő paraméterről van-e szó.
Ennek megfelelően a jelleg lehetséges értékei:

- IN bementi paraméter
- OUT kimeneti paraméter
- IN OUT mindkét irányba mutató adatforgalmat lebonyolító paraméter

Az adattípus a szokásos PL/SQL adattípusok valamelyike lehet. A törzsben szereplő PL/SQL blokkban a paraméterek ugyanúgy használhatók, mint a normál PL/SQL változók, így nem kell eléjük kettőspontot sem tenni a hivatkozáskor.
A PL/SQL blokk jellemzője, hogy nem kell benne DECLARE kulcsszót megadni a blokk kezdetének kijelölésére.

Tárolt függvény (Stored Function)

A tárolt függvények definíciója hasonló az eljárások definíciójához, azzal a különbséggel, hogy itt visszatérési érték is értelmezett.

```
CREATE FUNCTION függvény-név (
    paraméter-lista) RETURN értéktípus
BEGIN
    utasítások;
END;
```

A visszatérési érték típusát a paraméterlistát követően, a zárójel után megadott taggal jelöljük. A visszatérési értéket a RETURN utasítással határozzuk meg, mint az alábbi példa is mutatja.

Példa: adott típusú autók átlagárát határozza meg.

```
CREATE FUNCTION atlag (tip IN CHAR(20)) RETURN NUMBER
    IS
        ertek NUMBER;
BEGIN
    SELECT AVG(ar) INTO ertek FROM
    autok WHERE tipus LIKE tip;
    RETURN (ertek);
END;
```

Triggerek

A trigger koncepció az aktív integritási feltételek közé tartozik. A trigger két komponensből áll, egy *feltétel* és egy *választevékenység* részből.
A trigger működési elve igen egyszerű: ha a feltétel bekövetkezik, akkor végrehajtódik a választevékenység. A feltételt valamilyen adatkezelő művelet formájában adhatjuk meg. Vagyis a triggerrel az figyelhető, hogy végrehajtásra kerül-e valamilyen kijelölt adatkezelő művelet. A trigger alkalmazásával számos aktív integritási szabály definiálása az alkalmazásból átkerülhet az adatbázisba.

Ezáltal könnyebbé es gyorsabbá válhat az alkalmazások fejlesztése is, nem is beszélve a nagyobb biztonságról, hiszen ebben az esetben az alkalmazói program hibájából, a programozó feledékenysége miatt nem sérülhet meg az adatbázis integritása, hiszen az ellenőrzés mindig végrehajtódik az adatbáziskezelő szoftver által.

A triggereknél a választevékenységet az Oracle RDBMS esetén PL/SQL blokk formájában lehet megadni. Így választevékenység mindazon tevékenységi körre kiterjedhet, amik a PL/SQL nyelv keretében leírhatók, sőt a rendszer még bizonyos extra elemeket is bevezet a trigger koncepcióhoz történő jobb csatlakozáshoz.

A trigger definiálásának általános formátuma következőképpen írható le:

- CREATE TRIGGER triggernév kiváltó_ok PL/SQL_blokk;

ahol a kiváltó_ok a feltételt, míg a PL/SQL_blokk a választevékenységet adja meg. A kiváltó_ok rész több komponensből áll a feltétel pontos megadására. A fontosabb komponensek jelentése és formátuma a következő:

- előtag művelet ON tábla [FOR EACH ROW[WHEN feltétel]]

Az előtag azt jelöli ki, hogy a választevékenységet a figyelt művelet elvégzése előtt vagy után kell-e végrehajtani.

Az előtagban megadható kulcs-szavak:

- BEFORE művelet előtti végrehajtás
- AFTER művelet utáni végrehajtás

A művelet a figyelt adatkezelő műveletek körét jelöli ki, így az INSERT, UPDATE és DELETE utasításokra terjed ki. Egy triggerben egyidejűleg több tevékenység is figyelhető. A módosításnál a figyelés leszűkíthető az egyes mezőkre.

Az alábbi kulcsszavak szerepelhetnek a művelet részben:

- DELETE rekord törlés
- INSERT rekord bővítés
- UPDATE [OF mezőlista] rekord módosítás

Ha több műveletet is figyelni kívánunk, akkor az egyes műveleteket az OR kapcsolóval lehet összekötni.
A tábla azt a táblát jelöli ki, melyre a műveletek vonatkoznak. Egy trigger csak egy táblára vonatkozhat.
Az opcionális FOR EACH ROW tag akkor használatos, ha a megadott választevékenységet a műveletben érintett minden egyes rekordra külön-külön végrehajtásra kerülne. Ha ezt a tagok elhagyjuk, akkor a választevékenység az utasítás szinten hajtódik végre, utasításonként egyszer fut le a PL/SQL blokk.

Így például a

```
CREATE TRIGGER t1 AFTER DELETE ON auto
BEGIN
    INSERT INTO naplo VALUES ('torles', SYSDATE);
END;
```

triggernél a

```
DELETE FROM auto WHERE tip LIKE 'Fiat126%';
```

utasítás hatására egyetlen egyszer hívódik meg a PL/SQL blokk, azaz egyetlen egy új rekord fűződik be a napló állományba. Viszont a

```
CREATE TRIGGER t2 AFTER DELETE ON auto FOR EACH ROW
BEGIN
    INSERT INTO naplo VALUES ('torles', SYSDATE);
END;
```

trigger esetén ugyanazon törlési műveletnél többször is meghívódik a PL/SQL blokk, mégpedig annyiszor, ahány rekord kerül kitörlésre. Ekkor a napló táblába annyi új rekord kerül be, ahány rekordot kitöröltek az auto táblából.
A rekordszintű triggerek esetén lehetőség van arra, hogy ne minden érintett rekordra hívódjon meg a választevékenység, hanem csak azokra, amelyek egy megadott feltételnek eleget tesznek. E szelekciós feltételt a következő opcionális taggal adhatjuk meg.

```
WHEN feltétel
```

A létrehozott triggerek később módosíthatók, illetve megszüntethetők az ALTER TRIGGER és DROP TRIGGER utasításokkal.

A triggerek definiálásánál arra figyelni kell, hogy a triggerek nem izoláltak egymástól teljesen, mivel egy az egyik triggerhez kötött választevékenység kiválthat egy másik triggert, így kialakulhat egy trigger meghívási láncolat. A triggerek tervezésénél ügyelni kell e láncolatok kialakulására és hatására is.

A rendszer azt is megengedi, hogy több triggert is definiáljunk ugyanazon műveletek figyelésérére. Egy művelethez ugyanis létezhet BEFORE művelet szintű, BEFORE rekord szintű, AFTER művelet szintű és AFTER rekord szintű trigger.

Az egyes trigger típusok végrehajtási sorrendje:

1. BEFORE művelet szintű trigger
2. ciklus az érintett rekordokra
   (a) BEFORE rekord szintű trigger a rekordra
   (b) rekord zárolása és módosítása, integritási feltételek ellenőrzése
   (c) AFTER rekord szintű trigger a rekordra
3. késeltett ellenőrzésű integritási feltételek ellenőrzése
4. AFTER műveleti szintű trigger

Több kiváltott tigger esetén a választevékenység minden tagjának sikeresen végre kell hajtódnia, hogy a művelet, és minden általa kiváltott választevékenység megőrződjön.

A rekord szintű triggerek esetén lehetőség van arra, hogy az éppen érintett rekord adataihoz hozzáférjünk a PL/SQL blokkon belül. A mezők értékeit két rendszer rekordváltozón keresztül érhetjük el. A rendszer kétféle rekordváltozót is tartalmaz, az egyik a rekord régi, módosítás előtti, míg a másik a rekord új, módosítás utáni értékeit tartalmazza. A két rekordváltozó alapértelmezés szerinti azonosítói:

- OLD régi rekordérték
- NEW új rekordérték

A PL/SQL blokkon belül e változók, mint külső, nem a PL/SQL blokkban deklarált változók szerepelnek, ezért hivatkozáskor nevük elé egy kettőspontot kell tenni, hasonlóan ahogy a beágyazott SQL-ben a gazdanyelvi változókat használhatjuk. A rekordon belüli régi mezőkértékekre a

- :OLD.mezőnév

míg az új értékekre a

- :NEW.mezőnév

szimbólumokkal hivatkozhatunk. E rekordváltozók azonban csak a rekord szintű triggereknél élnek, és bizonyos műveleteknél csak az egyik formátuma él. Így az INSERT esetén nincs értelme az OLD hivatkozásnak, míg a DELETE esetén a NEW hivatkozásnak.

Ha egy triggert több tevékenységhez kapcsoltunk, pl. beszúráshoz és módosításhoz is, akkor a PL/SQL blokkon belül a

- INSERTING beszúrás jelző
- UPDATING módosítás jelző
- DELETING törlés jelző

rendszer által definiált konstansok segítségével eldönthető, hogy mely művelet volt az éppen futó választevékenység kiváltója. E változók igaz értéket vesznek fel, ha a hozzájuk tartozó művelet volt a kiváltó tevékenység.

Az alábbi példában a dolgozók tábla módosítása esetén az osztályok táblát is aktualizálja. A kapcsolat a két tábla között abban áll, hogy minden dolgozónak van egy osztálya, ahol dolgozik, és az osztályok táblában van egy olyan mező, amely az ott dolgozók összfizetését tartalmazza. Amikor egy dolgozó elmegy az osztályról, vagy egy új dolgozó jön az osztályra, vagy csak a dolgozó fizetése változik, akkor a osztályok tábla megfelelő rekordját is módosítani kell:

```
CREATE TRIGGER ossz_fiz AFTER DELETE OR
INSERT OR UPDATE OF oszt, fiz ON dolgozok
FOR EACH ROW
// PL/SQL blokk kezdete
BEGIN
    // ha törlés van vagy dolgozó áthelyezés
    IF DELETING OR (UPDATING AND
        :OLD.oszt != :NEW.oszt) THEN
        // összfizetés csökkentése a régi osztálynál
        UPDATE osztalyok SET osszfiz = osszfiz - :OLD.fiz
        WHERE oszt = :OLD.oszt;
    END IF;
    // ha új dolgozó vagy dolgozó áthelyezés
    IF INSERTING OR (UPDATING AND
    :OLD.oszt != :NEW.oszt) THEN
        // összfizetés növelése az új osztályon
        UPDATE osztalyok SET osszfiz = osszfiz + :NEW.fiz
            WHERE oszt = :OLD.oszt;
        END IF;
        // ha fizetés módosítás
        IF UPDATING AND :NEW.oszt = :OLD.oszt AND
        :NEW.fiz != :OLD.fiz) THEN
            // összfizetés módosítás az osztályon
            UPDATE osztalyok SET osszfiz = osszfiz + :NEW.fiz
            - :OLD.fiz WHERE oszt = :NEW.oszt;
        END IF;
    // blokk vége
END;
```

A triggerek alkalmazásának rugalmasságát fokozza, hogy ideiglenesen le is lehet tiltani őket, majd egy későbbi időpontban újra lehet engedélyezni a működését. A triggerek engedélyezése és letiltása az ALTER TRIGGER utasítással lehetséges.

A triggerek segítségével igen hatékony integritás ellenőrző, naplózó eszközt kaptunk a kezünkbe, melyet célszerű alaposan elsajátítani, mivel a jövő adatbáziskezelő rendszerei egyre nagyobb mértékben fognak ezen mechanizmusra támaszkodni.

A trigger definiálása a többi adatbázis objektum definiálásához hasonlóan az SQL nyelven keresztül történik. A fenti minta trigger teljes szövege egyetlen egy SQL utasításnak fog megfelelni, amelyet pl. az SQLPlus segítségével interaktívan is kiadhatunk.

Kurzorok

A PL/SQL kurzorszerkezet használata ugyanazon elvi lépésekre épül, mint a beágyazott SQL esetén, vagyis a

- kurzor deklaráció
- kurzor megnyitás
- rekord beolvasások ciklusa
- kurzor lezárás

A kurzor létrehozása a PL/SQL nyelvben a deklarációs részben, s nem a műveleti részben történik, mint az a beágyazott SQL esetén történt. A kurzor deklarációjának formátuma:

```
DECLARE
 ...
 CURSOR kurzornév (paraméterlista) IS SELECT_utasítás;
```

A következő példában egy kurzort deklarálunk a paraméterként megadott dátum előtt született személyek nevének és lakcímének a lekérdezésére:

```
DECLARE
    ...
    CURSOR lista (datum DATE) IS SELECT nev, lakcim
    FROM szemelyek WHERE szuldat < datum;
```

A deklarációs utasításban egy azonosító nevet rendelünk a kurzor szerkezethez, s e név segítségével hajtjuk végre a lekérdezést, s e név segítségével férhetünk hozzá az eredményhez is. A kurzor a PL/SQL nyelvben paraméteresen is deklarálható. A paramétereket a kapcsolódó SELECT utasításban kerülnek felhasználásra. A paraméterlista a paraméter azonosító neve mellett a paraméter típusát is tartalmazza. A listaelemek vesszővel vannak elválasztva egymástól. A PL/SQL szabályai szerint a minden felsorolt paraméternek meg kell jelennie a kapcsolódó SELECT utasításban is.

A kurzor megnyitása után a kapott adatokat egyenként lekérdezhetjük, s módosíthatjuk is. Módosítás esetén azonban a kurzor deklarációjánál jelezni kell, hogy nemcsak olvasásra hozzuk létre a kurzort. A módosítási igényt a SELECT utasítás végén álló, már ismert

- FOR UPDATE OF mezőlista

opcióval jelezzük. A kurzor által visszaadott rekordot a

- CURRENT OF kurzornév

feltétellel jelölhetjük a megfelelő UPDATE utasításban.
A kurzor megnyitása, vagyis kijelölt lekérdezés elvégzése az

- OPEN kurzornév (paraméterlista);

utasítással lehetséges, ahol a zárójelek között megadott paraméterlista opcionális elem. A paraméterlista most konkrét értékeket tartalmaz, melyek sorra behelyettesítődnek a deklarációkor kijelölt formális paraméterekbe, s ezen értékekkel fog a lekérdezés végrehajtódni. Ugyanaz a kurzor többször is végrehajtható különböző aktuális paraméterértékek mellett.
Az eredményrekordok egyenként történő lekérdezésére a

- FETCH kurzornév INTO változólista;

utasítás szolgál. A változólistának vagy annyi elemi változót kell tartalmaznia, ahány elemű az eredménytáblázat, vagy olyan rekordváltozót ad meg, mely struktúrája megegyezik az eredménytábla struktúrájával. A vizsgált PL/SQL változatban a kurzorpointer csak előre léptethető, mégpedig egy rekorddal.

A kurzor felhasználása után célszerű a kurzor által lefoglalt erőforrásokat felszabadítani. Ehhez ki kell adni a

- CLOSE kurzornév;

utasítást.

Az eredménytábla több rekordot is tartalmazhat, ezért a FETCH utasítást többször egymás után is ki kell adni a teljes választábla feldolgozásához. Az eredménytábla végének figyelését, vagyis annak ellenőrzését, hogy az összes rekordot érintettük-e már, egy kurzor attribútum segítségével végezhetjük el.

Az attribútum, melynek alakja

- kurzornév%NOTFOUND

akkor tartalmaz igaz értéket, ha elértük az eredménytábla végét, s nincs már további feldolgozásra váró rekord a kurzornál.

A lekérdező ciklus, melynek magjában a megfelelő FETCH utasítás áll, egy alap LOOP ciklus segítségével is megoldható, melybe a kilépéshez beteszünk egy

- EXIT WHEN kurzornév%NOTFOUND;

utasítást is.

Hibakezelés

A hibakezelő rutinokat a PL/SQL blokk harmadik komponense, az EXCEPTION kulcsszóval kezdődő rész tartalmazza. A hibakezelő részben minden egyes felismert hibatípushoz egyedi választevékenység definiálható. A hibakód és a választevékenység összerendelése a

```
WHEN hibakód THEN
    utasítások;
```

szerkezettel lehetséges. A hibakód utalhat általános, a rendszer által felismert hibákra, és saját, egyedi hibatípusokra is. A definiált tucatnyi rendszer hibakódból néhányat mutat be az alábbi felsorolás:

- NO_DATA_FOUND A SELECT utasítás vagy a FETCH nem tud eredményrekordot visszaadni

- ZERO_DIVIDE nullával való osztás
- VALUE_ERROR adatkonverziós hiba

A PL/SQL hibakezelési mechanizmusának számos előnye van a hagyományos módszerrel szemben. A hagyományos eljárásokban a hibakezelés szorosan összefonódott az utasításokkal: minden művelet után egyedileg kellett megadni a hibaellenőrző és lekezelő utasításokat. A PL/SQL ezzel szemben egy központi helyen tárolja a hibakezelő utasításokat. E módszer pozitív vonása, hogy

- csak egyszer kell leírni a hibakezelő kódot, áttekinthetőbb forrásszöveg
- könnyebb módosítási lehetőség, csak egy helyen kell módosítani a hibakezelő rutint.
- nagyobb megbízhatóság, mivel a hibaellenőrzés automatikusan minden műveletre kiterjed, nem lehet kibújni alóla, hiszen automatikusan, mindig érvényesül.

A hiba fellépte esetén a normál vezérlés megszakad, s hibakezelő utasításcsoport kapja meg vezérlést. A hiba feldolgozása után a blokk végrehajtása is befejeződik. Egy adott hibatípus esetén, ha az aktuális blokk nem tartalmazza a megfelelő hibakezelő rutint, akkor a külső blokkok átnézésével a rendszer megpróbálja a megfelelő hibakezelő rutint megtalálni.

Ha a hibakezelő utasításcsoportot nem csak egyetlen egy hibatípushoz szánjuk hozzárendelni, akkor lehetőség van a WHEN után több hibatípust is felsorolni, ahol az egyes típusokat az OR operátorral kötjük össze. Emellett alkalmazhatjuk a WHEN után az OTHER kulcsszót is, mellyel minden, explicit ki nem jelölt hibatípus esetén ide kerül a vezérlés.

A rendszer által észlelt hibák mellet a felhasználó maga is kiválthat végrehajtási hibákat. Itt most nem a véletlen programozási hibákról van szó, hanem arról, hogy a programozó explicit aktivizálhat hibatípusokat. Egy megadott típusú hiba kiváltása a

```
RAISE hibakód;
```

utasítással történik. Ekkor a vezérlés a megfelelő WHEN utasításra ugrik. A saját hibatípusokat, a gyári rendszer hibatípusoktól eltérően deklarálni kell a blokk deklarációs részében. A hibatípus létrehozása a

használalatát

```
hibatipus EXCEPTION;
```

utasítással lehetséges. Az alábbi programrészlet egy saját hibatípus használatát mutatja be:

```
DECLARE
 sajathiba  EXCEPTION;
BEGIN
 ...
 IF x < 16 THEN
 RAISE sajathiba;
 END IF;
 ...
EXCEPTION
 ...
 WHEN sajathiba THEN
 ROLLBACK;
 ...
END;
```

Példa összetett PL/SQL utasításra

A minta PL/SQL blokk, amelyben az auto táblában a FIAT típusú autók átlagáránál drágább OPEL típusú autók árát 12%-kal növeljük, ha színkóduk P betűvel kezdődik és 9%-kal növeljük, ha egyéb színűek.

```
DECLARE
  atlag NUMBER(10); -- átlagár
  szin  CHAR(3);    -- színkód
  CURSOR autok (atg NUMBER(10)) IS -- kurzor
    SELECT szin
    FROM   auto
    WHERE  ar > atg
    AND    tip LIKE 'OPEL%' FOR UPDATE OF ar;
BEGIN
  BEGIN -- alblokk az atlg kiszámításhoz
    SELECT SUM(ar)/Count(ar)
    INTO   minta.atlag
    FROM   auto
    WHERE  tip LIKE 'FIAT%';

  EXCEPTION
  WHEN zero_divide THEN -- ha nincs FIAT autó, akkor 100000 lesz az ár
    minta.atlag := 100000;
  END;

  OPEN autok (minta.atlag); -- kurzor megnyitása

  LOOP -- lekérdező ciklus
    FETCH autok
    INTO  minta.szin; -- rekord beolvasás

    -- kilépés ha nincs több
    EXIT
  WHEN autok%NOTFOUND;
    IF minta.szin LIKE 'P%' THEN
      UPDATE auto               -- módosítás
      SET    ar = ar * 1.12
      WHERE  CURRENT OF autok;

    ELSE
      UPDATE auto
      SET    ar = ar * 1.09
      WHERE  CURRENT OF autok;

    END IF; -- ciklus vége
  END LOOP;
  CLOSE autok; -- kurzor lezárása
  COMMIT; -- eredmények végelegesítése
END minta; -- blokk vége
```

Relációs adatbázis-sémák tervezése, normálformák, dekompozíciók

Relációs adatbázis-sémák tervezése

Az adatbázisok tervezésekor az egyik legfőbb feladat, a redundancia-mentes adatszerkezetet kialakítása.

Redundanciáról akkor beszélünk, ha valamely tényt vagy a többi adatból levezethető mennyiséget ismételten (többszörösen) tárolunk.

Relációk felbontása

A redundancia, a szükségtelen tároló terület lefoglalása mellett, komplikált frissítési és karbantartási műveletekhez vezet, melyek könnyen anomáliákat idézhetnek elő.

Anomáliák

Az anomália az adatbázisban olyan rendellenesség, mely valamely karbantartási műveletnél plusz műveletek beiktatását igényli, ezzel felesleges redundanciát okozva.

Dolgozó					
Név	Adószám	Cím	Osztálykód	Osztálynév	VezAdószám
Kovács	1111	Pécs, Vár u. 5.	2	Tervezési	8888
Tóth	2222	Tata, Tó u. 2.	1	Munkaügyi	3333
Kovács	3333	Vác, Róka u. 1.	1	Munkaügyi	3333
Török	8888	Pécs, Sas u.8.	2	Tervezési	8888
Kiss	4444	Pápa, Kő tér 2.	3	Kutatási	4444
Takács	5555	Győr, Pap u. 7.	1	Munkaügyi	3333
Fekete	6666	Pécs, Hegy u. 5.	3	Kutatási	4444
Nagy	7777	Pécs, Cső u. 25.	3	Kutatási	4444

- *Beszúrási anomália*: Beszúrási anomáliáról beszélünk abban az esetben, amikor egy adatrekord beszúrása egy másik, hozzá logikailag nem kapcsolódó adatcsoport beszúrását kívánja meg.

  Példa: Új dolgozó felvételénél előfordulhat, hogy az osztálynevet máshogy adják meg (például Tervezési helyett tervezési vagy Tervező).
  Ha új osztály létesül, amelynek még nincsenek alkalmazottai, akkor ezt csak úgy tudjuk felvenni, ha a (név, adószám, cím) mezőkhöz 'NULL' értéket veszünk. Később, ha lesznek alkalmazottak, ez a rekord fölöslegessé válik.

- *Módosítási anomália*: Abban az esetben, ha egy relációban egy adat módosítása több helyen történő módosítást igényel, akkor módosítási anomáliáról beszélünk.

  Példa: Ha egy osztály neve vagy vezetője megváltozik, több helyen kell a módosítást elvégezni.

- *Törlési anomália*: Amennyiben egy adat törlésével másik, hozzá logikailag nem kapcsolódó adatcsoportot is elveszítünk, törlési anomáliáról beszélünk.

  Példa: Ha egy osztály valamennyi dolgozóját töröljük, akkor az osztályra vonatkozó információk is elvesznek.

A normalizálás megértéséhez szükségünk van néhány további fogalom ismeretére.

Funkcionális függőségek

Funkcionális függőség

Funkcionális függésről akkor beszélünk, ha egy tábla valamelyik mezőjében lévő érték meghatározza egy másik mező értékét.

Példa: Személyek tábla

Személyek						
SzemIgSzam	Név	Irsz	Város	Utca	Telefonszam	Mobil
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*	(36)30/555 – 4143	*Igen*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*	(36)1/555 – 7891	*Nem*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*	(36)42/555 – 1235	*Nem*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)70/555 – 2935	*Igen*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)1/555 – 7891	*Nem*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*	(36)30/555 – 7777	*Igen*

A *Személyek* táblában egyes "személy egyedek" többször is előfordulnak. Ha egy nevet megemlítünk nem biztos, hogy pontosan ki tudjuk választani a hozzá tartozó személyi igazolvány számot. Ha azonban egy személyi igazolvány számot vizsgálunk meg, biztosan meg tudjuk mondani a hozzá tartozó nevet. Ezért azt mondjuk, hogy a *Nev* mező funkcionálisan függ a *SzemIgSzam* mezőtől. A *SzemIgSzam* mező azonban nem függ a *Nev* mezőtől.

Teljes funkcionális függőség

A funkcionális függés kiterjesztése a teljes funkcionális függés. Amikor egy adatbázist "normalizálunk", arra törekszünk, hogy minden táblában teljes funkcionális függések legyenek.

A teljes funkcionális függésnek három feltétele van:

- (1) egy tábla minden nem kulcs mezője függjön a kulcstól,
- (2) minden, nem kulcs mező csak a kulcstól függjön,
- (3) összetett kulcs esetén, minden nem kulcs mező függjön a kulcs minden elemétől.

Részleges funkcionális függőség

Részleges funkcionális függés a teljes funkcionális függés egyik akadálya. Akkor fordulhat elő egy táblában, ha abban van összetett kulcs és nem teljesül a teljes funkcionális függés (3)-as feltétele.

Példa:

Személyek						
SzemIgSzam	Név	Irsz	Város	Utca	Telefonszam	Mobil
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*	(36)30/555 – 4143	*Igen*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*	(36)1/555 – 7891	*Nem*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*	(36)42/555 – 1235	*Nem*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)70/555 – 2935	*Igen*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)1/555 – 7891	*Nem*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*	(36)30/555 – 7777	*Igen*

- A *Nev*, az *Irsz*, a *Varos*, és az *Utca* mezők a *SzemIgSzam* mezőtől függnek funkcionálisan.
- A *Mobil* mező nem a *SzemIgSzam*, hanem a *Telefonszam* mezőtől függ.
- Egy személyi igazolvány szám ismeretében pontosan meg tudjuk mondani, hogy hívják az illetőt, és hol lakik. Nem tudunk azonban biztos telefonszámot mondani, hiszen egy személyi igazolvány számhoz több telefonszám is tartozik.
- Egy telefonszám ismeretében egyértelműen megmondható, hogy az mobiltelefon-e. Az azonban még nem biztos, hogy egyértelmű nevet tudunk mondani, hiszen van olyan személy (4. rekord, Kis Veronika) aki ugyanazon a számon is elérhető.

Részleges funkcionális függés csak akkor fordulhat elő egy táblában, ha abban összetett kulcs van.

A normalizálás során a részleges funkcionális függést meg kell szüntetni.

Tranzitív függőség

Tranzitív függés esetén minden, nem kulcs mező függ a kulcstól, de van olyan mező, esetleg mezők, amely a kulcson kívül más mezőtől is függnek. A teljes funkcionális függés (2)-es feltétele, hogy "minden nem kulcs mező csak a kulcstól függjön".

Amennyiben ez a feltétel nem teljesül, *tranzitív függésről* beszélünk. Azt a mezőt, amelytől más mezők tranzitíven függnek, *tranzitív kulcsnak* hívjuk.

Példa:

SzemIgSzam	Név	Irsz	Város	Utca
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*

- A *SzemIgSzam* mező a kulcs.
- A kulcstól függ minden mező, azonban a *Varos* a *SzemIgSzam* mezőn kívül az *Irsz* mezőtől is függ.
- A *Varos* mező tehát tranzitív függésben van, az *Irsz* mező a tranzitív kulcs.

A normalizálás során a tranzitív függést meg kell szüntetni.

Funkcionális függőségek formálisan

Legyen $R(U)$ egy relációséma, ahol $U = \{A_1, \ldots, A_n\}$ attribútum halmaz. Valamint X, Y attribútumhalmazok, hogy $X, Y \subseteq U$.

Ekkor Y ***funkcionálisan függ*** X-től (Jelölése: $X \to Y$), ha bármely R séma feletti r reláció esetén, bármely két rekordjára igaz, hogy ha két rekord megegyezik X-en, akkor megegyezik az Y-on is, azaz

$$\forall t_1, t_2 \in r \text{ esetén } t_1[X] = t_2[X] \;\Rightarrow\; t_1[Y] = t_2[Y].$$

Ez lényegében azt jelenti, hogy az X-beli attribútumok értéke egyértelműen meghatározza az Y-beli attribútumok értékét. Azt, hogy az R kielégíti az $X \to Y$ függőséget $R \models X \to Y$-nal jelöljük.

Az $X \to Y$ függést *triviálisnak* nevezzük, ha $Y \subseteq X$, ellenekező esetben *nemtriviális* ($X \cap Y = \emptyset$).

Fontos:

- *Érdemi függés*: Azok az összefüggések, amelyek minden ilyen attribútumokkal rendelkező táblában fenn kell, hogy álljanak, az adatbázis bármely változása esetén is.

  Például: $SzemelyiSzam \to Nev$

- *Eseti függés*: Azok az összefüggések, amelyek csak egy adott időpillanatban állnak csak fent.

  Például: $Nev \to SzemelyiSzam$ (csupán addig igaz, amíg minden név egyedi egy relációban)

- Csak az érdemi függőségekkel foglalkozunk és a séma megadásakor döntjük el, hogy milyen függőségeket akarunk fenntartani.

A fenálló érdemi F függőségek halmazát a továbbiakban hozzávesszük az R relációsémához.
Jelölése: $(R; F)$.

Jobboldalak szétvágása

$X \to A_1A_2...A_n$ akkor és csak akkor teljesül R relációra, ha
$X \to A_1$, $X \to A_2$, ... , $X \to A_n$ is teljesül R-en.
Például: A→BC ekvivalens A→B és A→C függőségek kettősével.

Fontos: A függőségeknek csak a jobboldalát lehet szétbontani, a baloldalra ez természetesen nem igaz.

Kulcs, szuperkulcs

Egy $R(U)$ relációséma esetén $R \models X \to Y$ speciális esete, ha $Y = U$, ez a *kulcsfüggőség*. $R(U)$ relációséma esetén az $K \subseteq U$ attribútumhalmaz akkor és csak akkor szuperkulcs, ha a $R \models K \to U$.

A kulcsot tehát a függőség fogalma alapján is lehet definiálni: olyan K attribútumhalmazt nevezünk kulcsnak, amelytől az összes többi attribútum függ (vagyis szuperkulcs), de K-ból bármely attribútumot elhagyva ez már nem teljesül (vagyis minimális szuperkulcs).
Megjegyzés: A funkcionális függés nem kétirányú kapcsolat.
Például: *Személyek* $\models SzemelyiSzam \to Nev$, de *Személyek* $\not\models Nev \to SzemelyiSzam$

A séma megadása csak a keretet jelenti, beleértve a függéseket is, ha ezt feltöltjük adatokkal, akkor kapunk egy a sémára illeszkedő relációt. Az r reláció akkor illeszkedik az $(R; F)$ sémára ha az attribútumai az R-ben adottak és teljesülnek benne az F függőségek.

Függőségek implikációja

F implikálja $X \to Y$-t, ha minden olyan relációban, amelyben F összes függősége teljesül, $X \to Y$ is teljesül. Jelölés: $F \models X \to Y$, ha F implikálja $X \to Y$–et.

Legyenek $X_1 \to A_1$, $X_1 \to A_2$,..., $X_1 \to A_n$ adott funkcionális függőségek, szeretnénk tudni, hogy $Y \to B$ teljesül-e olyan relációkra, amire az előbbi funkcionális függőségek teljesülnek.

Példa: $A \to B$ és $B \to C$ teljesülése esetén $A \to C$ biztosan teljesül.

$Y \to B$ teljesülésének ellenőrzéséhez vegyünk két sort, amelyek megegyeznek az összes Y-beli attribútumon. Használjuk a megadott funkcionális függőségeket annak igazolására, hogy az előbbi két sor más attribútumokon is meg kell, hogy egyezzen. Ha B egy ilyen attribútum, akkor $Y \to B$ teljesül. Egyébként az előbbi két sor olyan előfordulást ad majd, ami az összes előírt egyenlőséget teljesíti, viszont $Y \to B$ mégsem teljesül, azaz $Y \to B$ nem következménye a megadott funkcionális függőségeknek.

Implikációs probléma eldöntése definíció alapján (minden előfordulásra ellenőrizni) lehetetlen, de van egyszerűbb lehetőség: levezetési szabályok (ún. Armstrong-axiómák) segítségével előállítani.

Armstrong-axiómák: Legyen R(U) relációséma és $X, Y \subseteq U$, és jelölje XY az X és Y attribútumhalmazok egyesítését. F legyen funkcionális függőségek tetszőleges halmaza.

(A1) (reflexivitás): $Y \subseteq X$ esetén $X \to Y$.

(A2) (bővíthetőség): $X \to Y$ és tetszőleges Z esetén $XZ \to YZ$.

(A3) (tranzitivitás): $X \to Y$ és $Y \to Z$ esetén $X \to Z$.

Levezetés: $X \to Y$ levezethető F-ből, ha van olyan $X_1 \to Y_1$, ..., $X_k \to Y_k$,..., $X \to Y$ véges levezetés, hogy $\forall k$-ra $X_k \to Y_k \in F$ vagy $X_k \to Y_k$ az FD1, FD2, FD3 axiómák alapján kapható a levezetésben előtte szereplő függőségekből. Jelölés: $F \vdash X \to Y$, ha $X \to Y$ levezethető F-ből.

További levezethető szabályok:

- Szétvághatósági szabály: $F \vdash X \to Y$ és $Z \subseteq Y$ esetén $F \vdash X \to Z$.
- Összevonhatósági szabály: $F \vdash X \to Y$ és $F \vdash X \to Z$ esetén $F \vdash X \to YZ$.
- Pszeudotranzitivitás: $F \vdash X \to Y$ és $F \vdash WY \to Z$ esetén $F \vdash XW \to Z$.

Az Armstrong-axiómarendszer helyes és teljes, azaz minden levezethető függőség implikálódik is, illetve azok a függőségek, amelyeket F implikál azok le is vezethetők F-ből. $F \vdash X \to Y \iff F \models X \to Y$

Mivel az Armstrong axiómarendszer helyes és teljes, elegendő a levezetési szabályokkal levezetni. De még a levezetési szabályoknál is van egyszerűbb út: kiszámítjuk Y lezártját: $Y+$-t.

Attribútumhalmaz lezártja

Adott R séma és F funkcionális függőségek halmaza mellett, X^+ az összes olyan A attribútum halmaza, amire X $\to$ A következik F-ből. (R;F) séma esetén legyen $X \subseteq R$.

$X^{+(F)} := \{A \mid F \vdash X \to A\}$ az X attribútumhalmaz ***lezárása*** F-re nézve.

Lemma. $F \vdash X \to Y \iff Y \subseteq X^+$.

A lemma következménye: az implikációs probléma megoldásához elég az X^+-t hatékonyan kiszámolni.

Lezárás algoritmus vázlata

- Kiindulás: $X^+ = X$.
- Indukció: Olyan funkcionális függőségeket keresünk, melyeknek a baloldala már benne van X^+-ban. Ha $W \to A$ ilyen, A-t hozzáadjuk X^+-hoz.
- Kimenet: Ha már nem bővül, ez a halmaz az X^+

A konkrét algoritmus:

- Kiindulás: $X^+ = X$.
- Iteráció, amíg X_n változik
  - $X_0 := X$
  - $X_{n+1} := X_n \bigcup \{A \mid W \to Z \in F,\ A \in Z,\ W \subseteq X_n\}$
  - Ha $X_{k+1} = X_k$, akkor Output: $X_v = X^+$
- Kimenet: X^+

Példa:

- $R = ABCDEFG, \quad AB \to C,\ B \to G,\ CD \to EG,\ BG \to E$
  $X = ABF,\ X^+ = ?$
  - $X(0) := ABF$
  - $X(1) := ABF \bigcup \{\mathbf{C,G}\} = ABCFG$
  - $X(2) := ABCFG \bigcup \{\mathbf{C,G,E}\} = ABCEFG$
  - $X(3) := ABCEFG$
  - $X^+ = ABCEFG$

Funkcionális függőségek vetítése

Motiváció: „normalizálás”, melynek során egy reláció sémát több sémára bonthatunk szét.

Példa: $R = ABCD, \quad F = \{AB \to C,\ C \to D,\ D \to A\}$.

- Bontsuk fel ABC és AD-re.
- Milyen funkcionális függőségek teljesülnek ABC–n?
- ABC-n nem csak $AB \to C$, de $C \to A$ is!

Vetület kiszámítása: Induljunk ki a megadott funkcionális függőségekből és keressük meg az összes nem triviális funkcionális függőséget, ami a megadott funkcionális függőségekből következik. (Nem triviális = a jobboldalt nem tartalmazza a bal). Csak azokkal az funkcionális függőségekkel foglalkozzunk, amelyekben a projektált séma attribútumai szerepelnek.

Függőségek vetülete: Adott (R;F), és $R_i \subseteq R$ esetén:

$$\Pi_{R_i}(F) := \left\{X \to Y | F \vdash X \to Y, XY \subseteq R_i\right\}$$

Felbontás (dekompozíció)

$d = \{R_1, \ldots, R_k\}$ az (R, F) ***dekompozíciója***, ha nem marad ki attribútum, azaz

$$R_1 \bigcup \ldots \bigcup R_k = R$$

Az adattábla felbontását projekcióval végezzük.

Elvárások a felbontással szemben:

1. Veszteségmentes legyen a felbontás, vagyis vissza tudjuk állítani az eredeti relációt a dekompozícióval kapott relációk soraiból.
   Azaz ha teljesül a következő összefüggés az előbbi összekapcsolásra azt mondjuk, hogy ***veszteségmentes***.

$$r \supseteq \Pi_{R_1}(r) \bowtie ... \bowtie \Pi_{R_k}(r) \text{ (ahol } r \text{ egy } R \text{ sémájú relációt jelöl),}$$

Chase-teszt a veszteségmentességhez:

(a) Készítünk egy felbontást.

(b) A felbontás eleminek összekapcsolásából veszünk egy sort.

(c) Az algoritmussal bebizonyítjuk, hogy ez a sor az eredeti relációnak is sora.

2. A vetületek legyenek jó tulajdonságúak, és a vetületi függőségi rendszere egyszerű legyen (normálformák: BCNF, 3NF, 4NF)

3. Függőségek megőrzése a vetületekben: A dekompozíciókban érvényes függőségekből következzen az eredeti sémára kirótt összes függőség. Adott (R;F) esetén $d = \{R_1, \ldots, R_k\}$ függőségőrző dekompozíció akkor és csak akkor, ha minden F-beli függőség levezethető a vetületi függőségekből: minden

$$X \to Y \in F \text{ esetén } \Pi_{R_1}(F) \cup ... \cup \Pi_{R_k}(F) \vdash X \to Y.$$

Függőségőrzés $\not\Rightarrow$ Veszteségmentesség
Veszteségmentesség $\not\Rightarrow$ Függőségőrzés

Normálformák

A nem triviális függőségek redundanciát okozhatnak. A redundancia kiküszöbölésének egyik módja a normalizálás.

A normalizálás során egy kezdeti állapotból több fázison keresztül átalakítjuk az adatbázist. Az átalakítás fázisait normálformáknak nevezzük. Megkülönböztetjük a nulladik, az első, második, harmadik, negyedik, ötödik, és hatodik normálformát. A normálformák jelölésére az 0NF, 1NF, 2NF, 3NF ... jelöléseket használjuk. Az adatbázis kialakítása mindig az alacsonyabbtól a magasabb normálformák felé halad. 0NF $\to$ 1NF $\to$ 2NF $\to$ 3NF $\to \ldots$

Minden normálforma kialakításának megvannak a maga szabályai, kialakításának előfeltételei, és a kialakításhoz szükséges műveletei. Ha egy tábla kielégíti az előfeltételeket, akkor elvégezhetjük vele a szükséges műveleteket. Eredményként olyan táblát kapunk, amely teljesíti a normálforma szabályait.

A relációs adatmodell szerint elkészült táblákat legalább harmadik normálformába kell alakítani. Ez általában elegendő ahhoz, hogy hibamentesen kezelhető adatbázisokat kapjunk.

Normalizálás:

- Funkcionális függőségek $\to$ (1,2,)3NF, BCNF
- Többértékű függőségek $\to$ 4NF

Első normál forma (1NF)

Azt mondja ki, hogy az attribútumok tartománya kizárólag atomi (egyszerű, oszthatatlan) értékeket tartalmazhat, és hogy a rekordokban bármely attribútum értéke csak egyetlen érték lehet az adott attribútum tartományából. Az 1NF ezáltal megtiltja, hogy egy rekordon belül az attribútumok értéke egy értékhalmaz, egy érték n-es vagy ezek kombinációja legyen.

Más szóval, az 1NF nem engedi meg a relációkon belüli relációkat, illetve a relációkat mint attribútumértékeket a rekordokon belül. Az 1NF szerint tehát egy attribútum értéke kizárólag egyetlen atomi (vagy oszthatatlan) érték lehet.

Azok relációk, amelyek nem elégítik ki az előző feltételt, nulladik normálformában vannak (0NF).

Példa:

Szakkörök		
Szakkör	**Tanár**	**Diákok**
Számítástechnika	Kiss Elemér	Név: Tóth Pál; Csizmazia Károly Osztály: III. b; II. a
Grafika	Tóth Árpád	Név: Kiss Veronika; Latabár Kálmán Osztály: I. c; II. b

$\Downarrow^{(0NF)}_{(1NF)}$

Szakkörök			
Szakkör	**Tanár**	**Diák**	**Osztály**
Számítástechnika	Kiss Elemér	Tóth Pál	III. b
Számítástechnika	Kiss Elemér	Csizmazia Károly	II. a
Grafika	Tóth Árpád	Kiss Veronika	I. c
Grafika	Tóth Árpád	Latabár Kálmán	II. b

Második normál forma (2NF)

Az R relációséma 2NF-ben van, ha R minden A másodlagos attribútuma teljesen funkcionálisan függ R elsődleges kulcsától.

A 2NF előfeltétele, hogy adatbázisunk minden táblája legalább 1NF-ben legyen! 2NF-ben vagyunk akkor, ha 1NF-ben vagyunk, és a táblákban megszűntetjük az esetleges részleges funkcionális függéseket.

A 2NF kialakításakor azt a táblát, amiben részleges funkcionális függés van, két új táblára bontjuk. Az egyik táblába az összetett kulcs egyik eleme kerül, a tőle függő összes mezővel együtt. A másik táblába a kulcs másik eleme kerül a tőle függő összes mezővel együtt. A két kapott táblában már nem lesz összetett kulcs, tehát nem lesz részleges funkcionális függés sem. Az új táblák azonosítói az eredeti összetett kulcs elemei lesznek.

Személyek						
SzemIgSzam	Név	Irsz	Város	Utca	Telefonszam	Mobil
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*	(36)30/555 – 4143	*Igen*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*	(36)1/555 – 7891	*Nem*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*	(36)42/555 – 1235	*Nem*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)70/555 – 2935	*Igen*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*	(36)1/555 – 7891	*Nem*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*	(36)30/555 – 7777	*Igen*

$\Downarrow^{2NF}$ (szétválasztás)

Személyek				
SzemIgSzam	Név	Irsz	Város	Utca
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*

Telefonok	
Telefonszam	Mobil
(36)30/555 – 4143	*Igen*
(36)1/555 – 7891	*Nem*
(36)42/555 – 1235	*Nem*
(36)70/555 – 2935	*Igen*
(36)30/555 – 7777	*Igen*

Az eredeti tábla "kettévágása" valójában azt eredményezte, hogy különválasztottunk két egyedtípust (Személyek-Telefonok) amelyek eddig egy táblában voltak. A művelet hatására csökken a redundancia.

Ez mindenképpen hasznos, de észre kell vennünk egy súlyos problémát. A két új tábla között még nincs kapcsolat. Nem tudjuk, melyik személy melyik számon hívható, illetve, hogy egy bizonyos készüléken kik érhetők el.

Tudjuk, hogy a relációs adatmodellben idegen kulcsokkal ábrázoljuk a kapcsolatokat. Ha a *Személyek* táblában helyezzük el az *Telefonok* táblából származó idegen kulcsot, jelezve, hogy melyik személyt melyik számon lehet elérni, akkor az idegen kulcs többértékű lesz, hiszen egy személynek több telefonja lehet.

Ha a telefonok táblában helyezzük el az idegen kulcsot, megmutatva, hogy kik hívhatók az adott készüléken, a mező ismét többértékű lesz, mert egy számon több személy is elérhető.

A probléma megoldhatatlannak látszik. Ha jól megvizsgáljuk a táblákat, észrevehetjük, hogy közöttük N:M kapcsolat van, hiszen egy embernek több telefonja is lehet, de lehet olyan telefon, amin keresztül több személy is elérhető (pl. munkahelyi telefon: (36) 1/555-7891.

A kapcsolattípusokról tanulva említettük, hogy a relációs adatmodellben két tábla között közvetlenül nem lehet több-több (N:M) kapcsolat. Ennek éppen az az oka, hogy ilyenkor bárhová is tennénk az idegen kulcsot, az többértékű mező lenne. A probléma úgy oldható meg, hogy a keletkezett két tábla között még egy harmadik, úgynevezett kapcsolótáblát is létrehozunk, amiben mindkét tábla azonosítóját elhelyezzük idegen kulcsként. Így a két tábla nem közvetlenül, hanem egy kapcsolótáblán keresztül kapcsolódik egymáshoz.

Személyek				
SzemIgSzam	Név	Irsz	Város	Utca
102564 BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*
234576 ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*
783402 EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*
982601 PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*
315672 ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*

Telefonok	
Telefonszam	Mobil
(36)30/555 – 4143	*Igen*
(36)1/555 – 7891	*Nem*
(36)42/555 – 1235	*Nem*
(36)70/555 – 2935	*Igen*
(36)30/555 – 7777	*Igen*

$\Downarrow^{2NF}$

SzemelyekTelefonok	
SzemIgSzam	Telefonszam
102564 BL	(36)51/555 – 4143
234576 ZM	(36)1/555 – 7891
783402 EA	(36)42/555 – 1235
982601 PM	(36)70/555 – 2935
982601 PM	(36)1/555 – 7891
315672 ZA	(36)30/555 – 7777

A relációs adatmodellben ,,A” és ,,B” egymással több-több kapcsolatban lévő egyedtípusok táblái nincsenek közvetlen kapcsolatban. A kapcsolat egy harmadik (”A_B”) tábla, a kapcsolótábla közvetítésével valósul meg. A kapcsolótáblában lévő mezők egyike, ,,A_Azon” az ,,A” tábla rekordjait, még a másik mező, ,,B_Azon” a ,,B” tábla rekordjait azonosítja.

A kapcsolótábla rekordjai elárulják, hogy az ,,A” tábla rekordjai mely rekordokhoz kapcsolódnak a ,,B” táblában és fordítva.

Megfigyelhetjük, hogy az ,,A” tábla 1:N kapcsolattal kapcsolódik az ”A_B” kapcsolótáblához, és a ,,B” tábla is 1:N kapcsolattal kapcsolódik a kapcsolótáblához. A kapcsolótábla két 1:N kapcsolattá alakítja az N:M kapcsolatot.

Harmadik normál forma (3NF)

A 3NF előfeltétele, hogy adatbázisunk minden táblája legalább 2NF-ben legyen! 3NF-ben vagyunk akkor, ha 2NF-ben vagyunk, és a táblákban megszűntetjük a tranzitív függéseket.

Minden tranzitív függést tartalmazó táblából két táblát csinálunk. Új táblába kerülnek a tranzitív függésben lévő mezők, azzal a tranzitív kulcs mezővel együtt, amelytől a kulcson kívül függnek. Az új táblában a tranzitív kulcs mező lesz az azonosító.

A 2NF kialakításakor létrejött *Személyek* tábla *Varos* mezője tranzitív függésben van. A *SzemIgSzam* kulcson kívül az *Irsz* mezőtől is függ. A tranzitív függést úgy szüntetjük meg, hogy az *Irsz* mezőt (tranzitív kulcs), és a *Varos* mezőt is új táblába, a *Varosok* táblába tesszük.

Személyek				
SzemIgSzam	Név	Irsz	Város	Utca
102564BL	*Tóth Árpád*	1082	*Budapest*	*Futó utca 11.*
234576ZM	*Szél Tamás*	1083	*Budapest*	*Bokay János 22.*
783402EA	*Egyed Bence*	5000	*Szolnok*	*Háló utca 2.*
982601PM	*Kis Veronika*	3022	*Lőrinci*	*Jegenyesor utca 2.*
315672ZA	*Nagy Bence*	3300	*Eger*	*Stadion utca 7.*

$\Downarrow^{3NF}$

Személyek			
SzemIgSzam	Név	Utca	Irsz
102564BL	*Tóth Árpád*	*Futó utca 11.*	1082
234576ZM	*Szél Tamás*	*Bokay János 22.*	1083
783402EA	*Egyed Bence*	*Háló utca 2.*	5000
982601PM	*Kis Veronika*	*Jegenyesor utca 2.*	3022
315672ZA	*Nagy Bence*	*Stadion utca 7.*	3300

Városok	
Irsz	Város
1082	*Budapest*
5000	*Szolnok*
3022	*Lőrinci*
3300	*Eger*

A 3NF kialakítása közben ismét új táblákat hoztunk létre, amelyek között idegen kulccsal kell biztosítanunk a kapcsolatot. Megfigyelhetjük, hogy a keletkezett *Városok* és *Személyek* táblák között egytöbb, (1:N) kapcsolat van, hiszen egy városban több személy állandó lakhelye van, egy ember állandó lakhelye azonban csak egy városban lehet. Ebben a kapcsolatban a *Városok* tábla oldalát 1, a *Személyek* tábla oldalát több oldalnak nevezzük.

Ha az idegen kulcsot az 1 oldalra, a *Városok* táblába tennénk (jelezve, hogy kik laknak az adott városban), akkor az, többértékű mező lenne, hiszen egy városban többen is laknak. Ha azonban a több oldalra, a *Személyek* táblába tennénk az idegen kulcsot (megmutatva, hogy hol van az adott személy állandó lakhelye), akkor az nem lenne többértékű. Ebből a tapasztalatból kiindulva a következő szabályt alkothatjuk: 1:N kapcsolat esetén az idegen kulcsot mindig a több oldalon lévő táblában helyezzük el.

- 1:N kapcsolat esetén tehát a több oldalon,
- N:M kapcsolat esetén pedig kapcsolótáblában helyezzük el az idegen kulcsot.
- 1:1 kapcsolat van, akkor az idegen kulcs bármelyik táblába kerülhet.

Az új tábla létrehozásának köszönhetően ismét csökkent a redundancia. Azoknak a városoknak a neveit, amelyekben több személy is lakik, most már csak egyszer kell tárolni.

Boyce-Codd normálforma

A normálformák tárgyalása során eddig olyan relációkra mutattunk példákat, melyeknek csak egy reláció kulcsa van. A normálformák definíciója alkalmazható a több kulccsal rendelkező relációkra is.

Ebben az esetben minden attribútum, mely valamely kulcsnak a része, elsődleges attribútum, de ez az attribútum függhet egy másik, ezt nem tartalmazó kulcs részétől. Ha ez a helyzet fennáll, redundanciát tartalmaz a reláció. Ennek a felismerése vezetett a harmadik normálforma egy szigorúbb definíciójához, a Boyce/Codd normálformához.

- A reláció harmadik normál formában van
- Minden elsődleges attribútum teljes funkcionális függőségben van azokkal a kulcsokkal, melyeknek nem része

A Boyce–Codd-féle normálforma látszólag a 3NF egy egyszerűbb alakja, de valójában erősebb, mint a 3NF. Azaz minden BCNF-ben lévő reláció egyúttal 3NF-ben is van, ám egy 3NF-ben lévő reláció nem szükségképpen van BCNF-ben.

Az R reláció BCNF-ben van akkor és csak akkor, ha minden olyan esetben, ha az R-ben érvényes egy $X \to Y$ nem triviális függőség, akkor az X attribútumhalmaz szuperkulcsa R-nek. Azaz minden nem triviális funkcionális függőség bal oldalának szuperkulcsnak kell lennie. (A szuperkulcsnak nem kell minimálisnak lennie.)

Bizonyos FF halmazok esetén a felbontáskor elveszíthetünk függőségeket. *3. normálformában* (3NF) úgy módosul a BCNF feltétel, hogy az előbbi esetben nem kell dekomponálnunk. Egy attribútum elsődleges attribútum (prím), ha legalább egy kulcsnak eleme. X → A megsérti 3NF-t akkor és csak akkor, ha X nem szuperkulcs és A nem prím.

Tantárgyak				
Tanár	**Időpont**	**Tantárgy**	**Félév**	**Diák_szám**
Kiss Pál	93/1	Adatbázis	1	17
Jó Péter	93/1	Unix	1	21
Kiss Pál	93/2	Adatbázis	2	32
Jó Péter	93/1	Unix	2	19
Kiss Pál	93/1	Adatbázis	3	25

$\Downarrow^{BCNF}$

Tantárgyak			
Időpont	**Tantárgy**	**Félév**	**Diák_szám**
93/1	Adatbázis	1	17
93/1	Unix	1	21
93/2	Adatbázis	2	32
93/1	Unix	2	19
93/1	Adatbázis	3	25

Tanárok		
Tanár	**Időpont**	**Tantárgy**
Kiss Pál	93/1	Adatbázis
Jó Péter	93/1	Unix
Kiss Pál	93/2	Adatbázis

Tételezzük fel, hogy minden tanár csak egy tantárgyat, de annak különböző féléveit oktatja. Ezek alapján a következő funkcionális függőségek írhatók fel:

- Tanár, Félév → Tantárgy
- Tantárgy, Félév → Tanár

A relációnak két kulcsa van, a (Tanár, Időpont, Félév) és a (Tantárgy, Időpont, Félév). A relációban csak egy nem elsődleges attribútum található, a Diák_szám. Ez teljes funkcionális függőségben van mindkét reláció kulccsal, az elsődleges attribútumok között nincs függőségi viszony. Ezek alapján a reláció harmadik normál formában van. Azonban tartalmaz redundanciát, mivel ugyanazon tanár mellett többször is tároljuk a tantárgyat azonos időpontokban. A redundanciának az az oka, hogy a tanár attribútum az őt nem tartalmazó reláció kulcs (Tantárgy, Időpont, Félév) csak egy részétől (Tantárgy, Félév) függ.

Többértékű függőségek és 4NF

A többértékű függőség (TÉF): az R reláció fölött $X \to\to Y$ teljesül: ha bármely két sorra, amelyek megegyeznek az X minden attribútumán, az Y attribútumaihoz tartozó értékek felcserélhetők, azaz a keletkező két új sor R-beli lesz.

Példa: Ha a *SörIvók* relációban a *KedveltSörök* között a következő sorok szerepelnek,

Sörívók			
Név	**Irányítószám**	**Telefonszám**	**KedveltSörök**
⋮	⋮	⋮	⋮
Kis Veronika	3022	70/555-2935	Kuchlbauer
Kis Veronika	3022	1/555-7891	Stella
⋮	⋮	⋮	⋮

akkor a következő előfordulásoknak is szerepelnie kell, mivel a sörivók telefonszámai függetlenek az általuk kedvelt söröktől.

Sörívók			
Név	**Irányítószám**	**Telefonszám**	**KedveltSörök**
Kis Veronika	3022	70/555-2935	Kuchlbauer
Kis Veronika	3022	1/555-7891	Stella
⋮	⋮	⋮	⋮
Kis Veronika	**3022**	**70/555-2935**	**Stella**
Kis Veronika	**3022**	**1/555-7891**	**Kuchlbauer**
⋮	⋮	⋮	⋮

Így egy-egy sörivó minden telefonszáma minden általa kedvelt sörrel kombinációban áll.

Egy R relációs sémában teljesül az $X \to\to Y$ többértékű függőség, ha minden R sémához tartozó r relációra igaz, hogy tetszőleges $t_1, t_2 \in r$ sorokra, melyekre $t_1[X] = t_2[X]$ léteznek $t_3, t_4 \in r$

- $t_3[XY] = t_1[XY]$,
- $t_3[R \setminus XY] = t_2[R \setminus XY]$,
- $t_4[XY] = t_2[XY]$,
- $t_4[R \setminus XY] = t_1[R \setminus XY]$.

Állítás: Elég az t_3, t_4 közül csak az egyik létezését megkövetelni.

Axiómák többértékű függőségekre:

(A4) (komplementer) : Ha $X \to\to Y$ és $Z = R \setminus XY$, akkor $X \to\to Z$.

(A5) (tranzitivítás) : Ha $X \to\to Y$ és $Y \to\to S$, akkor $X \to\to S \setminus Y$.

(A6) (bővíthetőség): $X \to\to Y$ és tetszőleges $V \subseteq W$ esetén $XW \to\to YV$

Tétel. $A4, A5, A6$ helyes és teljes a többértékű függőségekre.

Axiómák vegyes függőségekre:

(A7) (funkcionálisból többértékű) $X \to Y$ esetén $X \to\to Y$.

- Ha $X \to Y$ és két sor megegyezik X-en, Y-on is megegyezik, emiatt ha ezeket felcseréljük, az eredeti sorokat kapjuk vissza, azaz: $X \to\to Y$.

(A8) (többértékűből és funkcionálisból funkcionális): $X \to\to Y$ és $W \to S$, ahol $S \subseteq Y$, $W \cap Y = \emptyset$ esetén $X \to S$.

Tétel. $A1, A2, A3, A4, A5, A6, A7, A8$ helyes és teljes a vegyes függőségekre.

Állítás. $X \to\to Y$-ből nem következik, hogy $X \to\to A$, ha $A \in Y$. (A jobb oldalak nem szedhetők szét!)

Állítás. $X \to\to Y$ és $Y \to\to V$ nem következik, hogy $X \to\to V$, ha $A \in Y$.
(A szokásos tranzitivitás nem igaz általában!)

A veszteségmentesség, függőségőrzés definíciójában most F funkcionális függőségi halmaz helyett **D** függőségi halmaz többértékű függőségeket is tartalmazhat.

$d = \{R_1, \ldots, R_k\}$ az $(R; \mathbf{D})$ ***dekompozíciója***, akkor és csak akkor, ha minden D-t kielégítő r reláció esetén

$$r = \Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_k}(r)$$

A következő tétel miatt a veszteségmentesség az implikációs problémára vezethető vissza, így hatékonyan eldönthető.

Tétel. A $d = (R_1, R_2)$ akkor és csak akkor veszteségmentes dekompozíciója R-nek, ha

$$D \vdash R_1 \cap R_2 \to\to R_1 \setminus R_2$$

Megjegyzés:

- $Y \subseteq X$ vagy $XY = R$ esetén $X \to\to Y$ triviális többértékű függőség.
- X szuperkulcsa R-nek D-re nézve, ha $D \vdash X \to R$.

Negyedik normálforma

A Boyce/Codd normál forma is tartalmazhat redundanciát. Mindeddig csak a funkcionális függőségeket vizsgáltuk, a többértékű függőségeket nem. A további két normál forma a többértékű függőségekből adódó redundancia kiszűrését szolgálja. Egy reláció negyedik normál formában van.

A *4. normálforma* hasonlít a BCNF-re, azaz minden nem triviális többértékű függőség bal oldala szuperkulcs, de a többértékű függőségek okozta redundanciát a BCNF nem szünteti meg.

A megoldás: A negyedik normálforma. A negyedik normálformában (4NF), amikor dekomponálunk, a többértékű függőségeket úgy kezeljük, mint az funkcionális függőségeket, a kulcsok megtalálásánál azonban nem számítanak.

Egy R reláció 4NF -ben van, ha: minden $X \to\to Y$ nemtriviális többértékű függőség esetén X szuperkulcs.

R 4NF-ben van D-re nézve, ha $XY \neq R$, $Y \not\subset X$, és

$$D \vdash X \to\to Y \text{ esetén } D \vdash X \to R$$

$d = \{R_1, \ldots, R_k\}$ dekompozíció 4NF-ben van D-re nézve, ha minden R_i 4NF-ben van $\Pi_{R_i}(D)$-re nézve.

Állítás. Ha R 4NF-ben van, akkor BCNF-ben is van.
Következmény: Nincs mindig függőségőrző és veszteségmentes 4NF dekompozíció.

Veszteségmentes 4NF dekompozíciót mindig tudunk készíteni a naiv BCNF dekomponáló algoritmushoz hasonlóan.

Naiv algoritmus veszteségmentes 4NF dekompozíció előállítására:

(1) Ha R 4NF-ben van, akkor megállunk

- Egyébként van olyan nem triviális $X \to\to Y$, amely R-ben teljesül, de megsérti a 4NF-et, azaz X nem szuperkulcs.

(2) Ekkor R helyett vegyük az $(XY, R \setminus Y)$ dekompozíciót és (1).

Minden $X \to Y$ funkcionális függőség $X \to\to Y$ többértékű függőség is, így ha az R 4NF-ben van, akkor BCNF-ben is.
Képzeljük el azt, hogy egy relációban tároljuk a személyek, és barátaik nevét valamint hobbiját. Minden személynek több barátja és több hobbija is lehet.

BaratokHobbik		
Személy	**Barát**	**Hobbi**
Nagy József	Elek Attila	foci
Nagy József	Varga Attila	foci
Kiss Péter	Kiss Pál	sakk
Kiss Péter	Kiss Pál	video

Baratok	
Személy	Barát
Nagy József	Elek Attila
Nagy József	Varga Attila
Kiss Péter	Kiss Pál

Hobbik	
Személy	**Hobbi**
Nagy József	foci
Kiss Péter	sakk
Kiss Péter	video