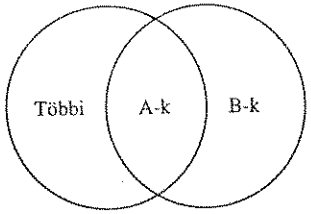


3.33. ábra. *BCNF megsértésén alapuló relációséma-felbontás*

1. Ezek a részhalmazok BCNF-ben levő relációsémák.

2. Az eredeti relációban tárolt adatokat hűségesen reprezentálják a felbontás eredményeként nyert relációk adatai. A 3.7.6. részben pontosítjuk, hogy ezt hogyan értjük. Durván megfogalmazva arra van szükségünk, hogy a felbontott relációkból képesek legyünk pontosan visszaállítani az eredeti relációt.

A 3.35. példa azt sugallja, hogy talán csak annyit kell tennünk, hogy a relációsémát kétattribútumos részhalmazokra bontjuk fel, és ennek az eredménye feltétlenül BCNF-ben van. Azonban ezek a felbontások nem mindig teljesítik a 2. feltételt, ahogyan később a 3.7.6. részben látni fogjuk. Valójában óvatosabban kell haladnunk, és a BCNF-et megszegő funkcionális függőségeket használjuk majd a felbontás vezérfonalaként.

Azt a felbontási stratégiát követjük, hogy kiválasztunk egy $A_1A_2...A_n \rightarrow B_1B_2...B_m$ nem triviális funkcionális függőséget, amely megsérti a BCNF-et, azaz $\{A_1, A_2, ..., A_n\}$ nem szuperkulcs. Heurisztikusan általában hozzávesszük a jobb oldalhoz mindazokat az attribútumokat, amelyeket

$\{A_1, A_2, ..., A_n\}$

funkcionálisan meghatároz. A 3.33. ábrán szemléltetjük hogyan bontjuk fel az attribútumokat két átfedő relációsémára. Az egyik az összes olyan attribútum, amelyet a megsértő függőség tartalmaz, a másik pedig a bal oldal plusz minden olyan attribútum, amelyet nem tartalmaz ez a függőség, azaz az összes attribútum kivéve a B-k közül azokat, amelyek nem A-k.

3.36. példa: Vegyük újból a 3.30. ábrán található `Film` relációt. Beláttuk korábban a 3.33. példában, hogy a

```
cím év → hossz szalagFajta stúdióNév
```

megsérti a BCNF feltételt. Ebben az esetben a jobb oldal már tartalmazza az összes olyan attribútumot, amelyet a `cím` és `év` funkcionálisan meghatároz, emiatt ezt a BCNF megsértést fogjuk alkalmazni a `Film` felbontásához az alábbi sémákra:

1. A függőségben szereplő összes attribútumból álló séma, azaz

   `{cím, év, hossz, szalagFajta, stúdióNév}`

2. A `Film` összes attribútumából álló séma, kivéve azt a hármat, amelyek a függőség jobb oldalán szerepelnek. Azaz töröljük a `hossz`, `szalagFajta` és `stúdióNév` attribútumokat. Az így megmaradó második séma a:

   `{cím, év, színészNév}`

Megjegyezzük, hogy ezek a sémák éppen azok, amelyeket a 3.32. példában a `Film1` és `Film2` relációkhoz választottunk. A 3.34. példában megmutattuk, hogy ezek mindegyike BCNF-ben van. □

3.37. példa: Vegyük a `FilmStúdió` relációt, amelyet a 3.30. példában vezettünk be. Ebben a relációban tároljuk az információt a filmekről, a gyártó stúdiókról és ezeknek a stúdióknak a címeiről. Az ehhez a relációhoz tartozó sémát és néhány jellemző sorát a 3.34. ábrán mutatjuk be.

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*	*stúdióCím*
Csillagok háborúja	1977	124	színes	Fox	Hollywood
Rút kiskacsa	1991	104	színes	Disney	Buena Vista
Wayne világa	1992	95	színes	Paramount	Hollywood
Addamék családja	1991	102	színes	Paramount	Hollywood

3.34. ábra. *A `FilmStúdió` reláció*

Megjegyezzük, hogy a `FilmStúdió` reláció redundáns információt tartalmaz. Minthogy a szokásos adatainkhoz hozzávettünk egy a Paramount által gyártott második filmet is, a Paramount címe kétszer szerepel. Jóllehet itt a probléma forrása más, mint ami a 3.36. példában volt. A korábbi példában az volt a probléma, hogy egy többértékű kapcsolatot (adott filmben szereplő színészeket) a film többi információjával együtt tároltuk. A mostani példában minden egyértékű: a film `hossz` attribútuma, a `gyártó` kapcsolat, amely a filmekhez kapcsolja egyértelműen a gyártó stúdiót, és a stúdiók `cím` attribútuma.

Ebben az esetben az a probléma, hogy van egy „tranzitív függés". Amint korábban a 3.30. példában már megjegyeztük a `FilmStúdió` reláció függőségei a

```
cím év → stúdióNév
stúdióNév → stúdióCím
```

Alkalmazzuk ezekre a tranzitív szabályt, amivel egy új függőséget kapunk:

```
cím év → stúdióCím
```

Azaz a cím és az év (azaz a filmek kulcsa) funkcionálisan meghatározza a stúdió címét – annak a stúdiónak a címét, amelyik gyártotta a filmet. Mivel

```
cím év → hossz szalagFajta
```

egy másik nyilvánvaló függőség, ebből következik, hogy a {cím, év} a FilmStúdió kulcsa, sőt valójában az egyetlen kulcsa. Másrészt a

```
stúdióNév → stúdióCím
```

függőség, amelyet a fenti tranzitív szabály alkalmazásában használtunk, nem triviális, és a bal oldala nem szuperkulcs. Ezek szerint a FilmStúdió nincs BCNF-ben. Megoldhatjuk a problémát, ha a fenti függőség felhasználásával követjük a felbontási szabályt. A felbontás első sémája magában a függőségben szereplő attribútumokból áll, azaz:

```
{stúdióNév, stúdióCím}
```

A második séma a stúdióCím kivételével a FilmStúdió összes attribútumát tartalmazza, mert ez az egy attribútum szerepel a felbontásban alkalmazott függőség jobb oldalán. Azaz a másik séma:

```
{cím, év, hossz, szalagFajta, stúdióNév}
```

A 3.34. ábrán szereplő reláció projekciói ezekre a sémákra megadják a FilmStúdió1 és a FilmStúdió2 relációkat, amelyeket a 3.35., illetve a 3.36. ábrákon láthatunk. Mind a kettő BCNF-ben van. A FilmStúdió1 egyedüli kulcsa a {cím, év}, és a FilmStúdió2 egyedüli kulcsa pedig a {stúdióNév}. Egyik esetben sincs olyan nem triviális függőség, amely ne tartalmazná ezeket a kulcsokat a bal oldalon. □

cím	*év*	*hossz*	*szalagFajta*	*stúdióNév*
Csillagok háborúja	1977	124	színes	Fox
Rút kiskacsa	1991	104	színes	Disney
Wayne világa	1992	95	színes	Paramount
Addamék családja	1991	102	színes	Paramount

3.35. ábra. *A FilmStúdió1 reláció*

stúdióNév	*stúdióCím*
Fox	Hollywood
Disney	Buena Vista
Paramount	Hollywood

3.36. ábra. *A FilmStúdió2 reláció*

Az előző példák mindegyikében a felbontási szabály egyetlen jól átgondolt alkalmazása elég volt, hogy előállítsuk a BCNF-ben levő relációkat. Általában nem ez a helyzet.

3.38. példa: A 3.37. példát tudjuk úgy általánosítani, hogy a funkcionális függőségeknek kettőnél hosszabb lánca legyen. Vegyük a következő relációsémát

```
{cím, év, stúdióNév, elnök, elnökCím}
```

Azaz ennek a relációnak minden egyes sora megadja egy film adatait, a stúdióját, a stúdió elnökét, és a stúdió elnökének a címét. Tegyük fel, hogy az alábbi három függőség teljesül ebben a relációban:

```
cím év → stúdióNév
stúdióNév → elnök
elnök → elnökCím
```

Ehhez a relációhoz az egyedüli kulcs a {cím, év}. Azaz az utolsó két függőség megsérti a BCNF-et. A felbontást kezdhetjük például a

```
stúdióNév → elnök
```

függőséggel. Először ennek a funkcionális függőségnek a jobb oldalához vegyük hozzá a stúdióNév lezárásában előforduló attribútumokat. A tranzitivitási szabályt alkalmazva stúdióNév → elnök és elnök → elnökCím kapjuk, hogy

```
stúdióNév → elnökCím
```

Azt a két függőséget kombinálva, amelyiknek a bal oldalán stúdióNév áll, kapjuk:

```
stúdióNév → elnök elnökCím
```

Ennek a funkcionális függőségnek már maximálisra kibővítettük a jobb oldalát, emiatt a felbontásban a következő két relációsémához jutunk

```
{cím, év, stúdióNév}
{stúdióNév, elnök, elnökCím}
```

Ezek közül az első BCNF-ben van. A másodiknak a {stúdióNév} az egyetlen kulcsa, és érvényes benne az

```
elnök → elnökCím
```

függőség, amely megsérti a BCNF-et. Emiatt folytatnunk kell a felbontást ezzel a függőséggel, kibővítve a jobb oldalt az `elnök` attribútummal. Az eredményül kapott alábbi három relációséma mind BCNF-ben van:

```
{cím, év, stúdióNév}
{stúdióNév, elnök}
{elnök, elnökCím}
```
□

Általában addig kell folytatnunk a felbontási szabály alkalmazását, ameddig az összes relációnk BCNF-ben nem lesz. Biztosak lehetünk abban, hogy ez sikeresen befejeződik, hiszen minden alkalommal, amikor a felbontási szabályt alkalmazzuk az R relációra az eredményül kapott mindkét sémában kevesebb attribútum van, mint az R-ben volt. Amikor pedig lejutunk két attribútumig, az a reláció biztos, hogy BCNF-ben van, amint ezt korábban a 3.35. példában láttuk.

Ahhoz, hogy belássuk miért eredményez a felbontás mindig kisebb sémákat, tegyük fel, hogy van olyan $A_1A_2...A_n \rightarrow B_1B_2...B_m$, amely megsérti a BCNF-et. Feltehetjük, hogy ezt a függőséget már kibővítettük úgy, hogy tartalmazza a B-k között az összes olyan attribútumot, amelyeket az A-k funkcionálisan meghatároznak, valamint, hogy a B-k közül azokat, amelyek az A-kban is szerepelnek eltávolítottuk a B-kből.

A felbontás egyik sémája a B-k kivételével tartalmazza az R összes attribútumát. Mivel legalább egy B-nek kell lennie, ezért ez a séma nem tartalmazza az összes attribútumot.

A másik séma az összes A-kból és B-kből áll. Ez a halmaz nem állhat az R minden attribútumából, mivel ha így volna, akkor $\{A_1A_2...A_n\}$ szuperkulcs lenne R-ben (azaz az A-k funkcionálisan meghatároznák az R minden attribútumát), viszont nem lehet BCNF-et sértő függőség bal oldala szuperkulcs.

Arra jutottunk, hogy a felbontás mindkét sémája kisebb, mint az R sémája. Emiatt a felbontási eljárás ismétlésével szükségszerűen megkapjuk a BCNF-ben levő relációkat.

3.7.5. Funkcionális függőségek vetítése

Amikor felbontunk egy relációsémát, ellenőriznünk kell, hogy az eredménysémák BCNF-ben vannak-e. A 3.38. példában láttuk, hogy előfordulhat, hogy az új sémák közül az egyik vagy mindkettő megsérti a BCNF-et. Hogyan tudjuk megmondani, hogy egy reláció BCNF-ben van, ha nem tudjuk meghatározni, hogy abban a relációban mely funkcionális függőségek érvényesek. A 3.38. példában ad hoc módon indokoltuk csak, hogy az új relációkban mely függőségek állnak fenn. Szerencsére van elméleti út, hogy megkeressük a felbontás eredményeihez tartozó funkcionális függőségeket.

Vegyünk egy R relációt, amelyet felbontottunk az S-re és egy másik relációra. Legyen F az R-ben érvényes funkcionális függőségek halmaza. Számoljuk ki az S-ben fennálló funkcionális függőségeket a következőképpen:

Vegyük az S attribútumhalmaz által tartalmazott minden egyes X attribútumhalmazt. Számoljuk ki X^+-t. Ekkor minden B attribútumra, amelyikre

1. B az S-nek egy attribútuma,

2. B benne van X^+-ban,

3. B nincs benne X-ben,

az $X \rightarrow B$ funkcionális függőség fennáll S-ben.

3.39. példa: Legyen az R sémája $R(A, B, C)$, és tegyük fel, hogy $A \rightarrow B$ és $B \rightarrow C$ funkcionális függőségek adottak az R-hez. Legyen R felbontásának egyik relációja $S(A, C)$. Kiszámoljuk az S-ben érvényes függőségeket.

Elméletben ki kell számolnunk az S attribútumhalmazának, azaz $\{A, C\}$-nek minden egyes részhalmazhoz a lezárását. Kezdjük $\{A\}^+$-szal. Könnyű belátni, hogy ez a halmaz $\{A, B, C\}$. Mivel B nincs benne az S sémájában, nem állíthatjuk, hogy $A \rightarrow B$ S-ben érvényes függőség lenne. Mivel C benne van az S sémájában, emiatt $A \rightarrow C$ az S-ben érvényes függőség.

Most nézzük meg $\{C\}^+$-t. Mivel C nem szerepel egyik adott függőség bal oldalán sem, semmi újat nem kapunk a lezárásban, tehát $\{C\}^+ = \{C\}$. Általában azok a halmazok, amelyeket az adott függőségek közül egyiknek a bal oldala sem tartalmaz, nem jelenthetnek az S-re semmilyen függőséget.

Nézzük meg $\{A, C\}^+$-t is, amely $\{A, B, C\}$. Ezzel sem kaptunk az $\{A\}^+$-hoz viszonyítva új függőséget. Ebből arra következtethetünk, hogy $A \rightarrow C$ az egyetlen függőség, amelyet S-re fel kell tennünk. Természetesen az S-re érvényesek további más függőségek, amelyek ebből az egyből levezethetők, mint az $AC \rightarrow C$ vagy a triviális függőségek, mint $A \rightarrow A$. Ezeket megkaphatjuk a 3.6. részben megadott szabályok segítségével, és nem kell külön feltenni, amikor az S funkcionális függőségeit adjuk meg. □

3.40. példa: Vegyük az $R(A, B, C, D, E)$ felbontását $S(A, B, C)$-re és egy másik relációra. Legyenek az R funkcionális függőségei $A \rightarrow D$, $B \rightarrow E$ és $DE \rightarrow C$.

Először nézzük meg $\{A\}^+ = \{A, D\}$. Mivel D nincs benne az S sémájában, ebben az esetben nem kaptunk érvényes függőséget. Hasonlóan a $\{B\}^+ = \{B, E\}$ és $\{C\}^+ = \{C\}$ sem szolgáltat érvényes függőséget az S-re.

Most nézzük meg a párokat. $\{A, B\}^+ = \{A, B, C, D, E\}$. Emiatt az $AB \rightarrow C$ függőség fennáll S-re. A többi párok közül egyik sem szolgáltat érvényes függőséget az S-re. Természetesen az S mindhárom attribútumából álló halmaz, az $\{A, B, C\}$ nem eredményezhet semmilyen nem triviális függőséget az S-re. Így az egyetlen függőség, amelyet S-re fel kell tennünk az $AB \rightarrow C$. □

3.7.6. Információ visszanyerése a felbontásból

Fordítsuk a figyelmünket arra a kérdésre, hogy a 3.7.4. részben közölt felbontási algoritmus miért őrzi meg az eredeti relációban tárolt információt. Az ötlet, hogy ha ezt az algoritmust követjük, az, hogy az eredeti sorok vetítéseiből újra vissza tudjuk kapni természetes összekapcsolással az eredeti sorokat, mégpedig az összeset és csakis ezeket.*

Az egyszerűség kedvéért vegyük az $\{A, B, C\}$ sémájú R relációt, és a $B \rightarrow C$ funkcionális függőséget, amelyről feltesszük, hogy megsérti a BCNF-et. Előfordulhat például, hogy van egy másik $A \rightarrow B$ funkcionális függőség, mint ahogyan korábban a 3.37. példában láttuk, ahol létezett egy tranzitivitási függőségi lánc. Ebben az esetben $\{A\}$ az egyetlen kulcs, és $B \rightarrow C$ bal oldala nyilvánvalóan nem szuperkulcs. A másik lehetőség, hogy $B \rightarrow C$ az egyetlen nem triviális függőség, ebben az esetben az egyedüli kulcs $\{A, B\}$. Ekkor $B \rightarrow C$ bal oldala szintén nem szuperkulcs. Mindkét esetben a $B \rightarrow C$ függőségen alapuló felbontás az attribútumokat két sémába osztja szét $\{A, B\}$ és $\{B, C\}$ sémákra.

Legyen t az R reláció egy sora. Legyen $t = (a, b, c)$, ahol a, b és c az A, B és C-nek megfelelő t-beli komponenseket jelölik. A t sor vetítése az $\{A, B\}$ sémára (a, b), és a $\{B, C\}$ sémára pedig (b, c).

Természetes összekapcsolással hozzá tudjuk kapcsolni az $\{A, B\}$-ből származó sorokhoz a $\{B, C\}$-ből származó sorokat, feltéve ha megegyeznek a B komponensen. Speciálisan (a, b)-hez kapcsolva (b, c)-t visszakapjuk az eredeti $t = (a, b, c)$ sort. Független attól, hogy melyik t sorból indulunk ki, ennek a vetítéseiből mindig vissza tudjuk kapni az eredeti t-t.

A függőségek keresésének egyszerűsítése

Amikor a funkcionális függőségeket származtatjuk az R-beliekből az S relációra a 3.7.5. rész algoritmusának a segítségével, akkor gyakran korlátozhatjuk a keresést azzal, hogy nem számoljuk ki az S attribútumainak összes részhalmazára a lezárást. Megadunk néhány szabályt, amelyek segítségével csökkenthetjük a munkánkat.

1. Figyelmen kívül hagyhatjuk az S összes attribútumából álló halmaz lezárását.
2. Figyelmen kívül hagyhatunk egy attribútumhalmazt, ha nem tartalmazza egyik függőség bal oldalát sem.
3. Figyelmen kívül hagyhatunk egy halmazt, ha tartalmaz egy olyan attribútumot, amely egyik függőség bal oldalán sem fordul elő.

* Szerkesztői megjegyzés: az ilyen tulajdonságú felbontásokat veszteségmentes összekapcsolású felbontásnak nevezik.

Az, hogy visszakapjuk azokat a sorokat, amelyekből kiindultunk, még nem biztosítja elegendően, hogy az eredeti R relációnkat valóban reprezentálja a felbontás. Mi történhet, ha lenne az R-nek két sora, mondjuk $t = (a, b, c)$ és $v = (d, b, e)$? Amikor t sort vetítjük az $\{A, B\}$ sémára kapjuk az $u = (a, b)$-t, és amikor a v-t vetítjük a $\{B, C\}$-re a $w = (b, e)$-t kapjuk, amint ezt a 3.37. ábra mutatja.

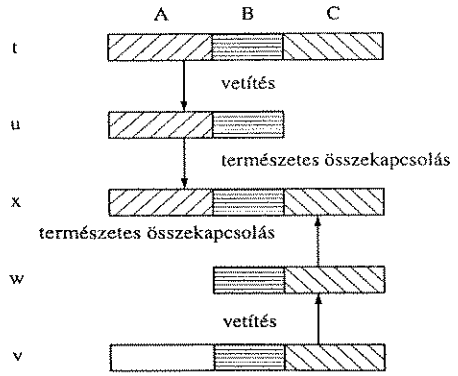
Az u és w sorokat összekapcsolhatjuk, mivel ezek megegyeznek a B komponenseken. Az eredmény sor $x = (a, b, e)$. Lehetséges, hogy az x hamis sor? Azaz lehetséges, hogy (a, b, e) az R-nek nem sora?

Mivel feltettük az R relációra $B \rightarrow C$ funkcionális függőséget, a válasz „nem". Emlékezzünk vissza, hogy ez a függőség azt mondja, hogy az R-nek bármely két sora, amelyik megegyezik a B komponensen, meg kell hogy egyezzen a C komponensen is. Mivel t és v megegyeznek a B komponenseiken (mind a kettő b), ezeknek meg kell egyezniük a C komponenseiken is. Ez azt jelenti, hogy $c = e$, azaz a két érték, amiről feltételeztük, hogy különböznek, valójában megegyeznek. Így (a, b, e) valójában (a, b, c), azaz $x = t$.

Mivel t az R-ben van, emiatt x-nek is benne kell lennie R-ben. Más szavakkal, amíg a $B \rightarrow C$ funkcionális függőség érvényes, a két vetített sor összekapcsolásával nem állíthatunk elő hamis sorokat, azaz az összekapcsolással nyert minden sor biztosan az R-ben van.

Általánosan is így van. Tegyük fel, hogy A, B és C egyszerű attribútumok, de ugyanezt az érvelést alkalmazhatjuk attribútumhalmazokra is. Vegyünk egy tetszőleges függőséget, amely megsérti a BCNF-et, legyen B a bal oldali attribútumok, C a jobb oldalon szereplő olyan attribútumok, amelyek nem fordulnak elő a bal oldalon, és A pedig olyan attribútumok, amelyek egyik oldalon sem szerepelnek. Arra lehet következtetnünk, hogy:

- Ha egy relációt felbontunk a 3.7.4. rész módszere szerint, akkor az eredeti relációt pontosan vissza tudjuk állítani az új relációk sorainak az összes lehetséges módon vett természetes összekapcsolásával.



3.37. ábra. *A vetített relációk két sorának természetes összekapcsolása*

Ha a relációk felbontása úgy történik, hogy nem funkcionális függőségen alapul, akkor nem mindig tudjuk visszaállítani az eredeti relációt. Mutatunk erre egy példát.

3.41. példa: Vegyük az $\{A, B, C\}$ sémájú R relációt, mint az előbb, csak a $B \to C$ funkcionális függőség most nem teljesül. Ekkor R állhat a következő két sorból

A	B	C
1	2	3
4	2	5

Az R vetítései az $\{A, B\}$ és $\{B, C\}$ sémájú relációkra

A	B
1	2
4	2

és

B	C
2	3
2	5

Mivel mind a négy sor ugyanaz a B érték, a 2, így az egyik reláció mindkét sorához hozzákapcsolhatjuk a másik reláció mindkét sorát. Így amikor megpróbáljuk az R-et természetes összekapcsolással visszaállítani, azt kapjuk, hogy

A	B	C
1	2	3
1	2	5
4	2	3
4	2	5

Így „túl sokat" kaptunk, van két hamis sorunk is, az (1, 2, 5) és (4, 2, 3) nem szerepeltek az eredeti R relációban. □

3.7.7. Harmadik normálforma

Esetenként találkozhatunk olyan relációsémával és hozzá kapcsolódó függőségekkel, amely nincs BCNF-ben, mégsem kívánjuk tovább felbontani. A következő példa jellemző erre.

3.42. példa: Legyen egy `Foglalások` relációnk az alábbi attribútumokkal:

1. `cím`, a filmcím.
2. `mozi`, a mozi neve, ahol a filmet bemutatják.
3. `város`, az a város, ahol a mozi van.

A (c, m, v) sor azt fejezi ki, hogy a c című filmet jelenleg melyik m mozi játssza a v városban.

Ésszerűen feltehetjük az alábbi függőségeket:

```
mozi → város
cím város → mozi
```

Az első szerint minden egyes mozi csak egy városban van. A második nem nyilvánvaló, de azon a feltevésen alapul, hogy ugyanazt a filmet nem játsszák egy városban két moziban. Csak a példa kedvéért tegyük fel azonban ezt a függőséget is.

Először keressük meg a kulcsokat. Nincs egy attribútumból álló kulcs. Például a `cím` nem kulcs, hiszen egy filmet több mozi is vetíthet egy időben több városban.[9] A `mozi` sem kulcs, bár a `mozi` funkcionálisan meghatározza a `várost`, mégis lehetnek többtermes mozik, ahol több filmet is bemutatnak egyszerre. Így a `mozi` nem határozza meg a `címet`. Végül a `város` sem kulcs, hiszen általában a városokban több mozi van és több filmet játszanak.

A három kétattribútumos halmaz közül kettő kulcs. Valóban a `{cím, város}` kulcs, mert az adott függőség szerint ezek az attribútumok funkcionálisan meghatározzák a `mozit`.

Igaz az is, hogy a `{mozi, cím}` kulcs. Ahhoz, hogy bebizonyítsuk, induljunk ki az adott `mozi → város` függőségből. A 3.6.3.a) feladat bővítési szabálya szerint ebből a `mozi cím → város` következik. Ez nem csoda, hiszen ha a `mozi` önmagában funkcionálisan meghatározza a `várost`, akkor a `mozi` és a `cím` együtt is meghatározzák.

A maradék attribútumpár a `város` és `mozi` nem határozza meg funkcionálisan a `címet`, és így nem kulcs. Azt kaptuk, hogy csak két kulcs van, amelyek

```
{cím, város}
{mozi, cím}
```

Most azonnal látható a BCNF megsértése. Adott volt a `mozi → város` funkcionális függőség, aminek a bal oldala a `mozi` nem szuperkulcs. Emiatt megpróbáljuk felbontani két relációsémára ennek a BCNF-et megsértő függőségnek a segítségével:

[9] Ebben a példában feltesszük, hogy nincs két különböző, de azonos nevű aktuális film, az-az szemben egy korábbi észrevételünkkel, most nem engedjük meg, hogy ugyanannak a filmnek legyen két különböző évben készült változata.

Más normálformák

Ha van „harmadik normálforma" mi történt az első két „normálformával"? Ezeket valóban definiálták, de ma már keveset használjuk őket. Az *első normálforma* egyszerűen az a feltétel, hogy minden sor minden komponense atomi értékű legyen. A *második normálforma* kevésbé szigorú feltétel, mint a 3NF. Megengedjük a tranzitív függőségeket egy relációban, de megtiltjuk azokat a nem triviális függőségeket, amelyeknek bal oldala egy kulcs valódi részhalmaza. Van „negyedik normálforma" is, amivel később a 3.8. részben találkozunk.

{mozi, város}
{mozi, cím}

Van egy probléma ezzel a felbontással a `cím város → mozi` funkcionális függőséggel kapcsolatban. A felbontott sémákhoz tartozhatnak olyan aktuális relációk, amelyek kielégítik a `mozi → város` funkcionális függőséget (ezt a {mozi, város} relációban tudjuk ellenőrizni), mégis, amikor összekapcsoljuk őket, az eredményül kapott reláció nem tesz eleget a `cím város → mozi` függőségnek. Például legyen a két reláció

mozi	*város*
Corvin	Budapest
Kert	Budapest

és

mozi	*cím*
Corvin	A hálózat
Kert	A hálózat

Mindkettő reláció megengedett, ha a funkcionális függőségeket nézzük. Mégis, amikor összekapcsoljuk őket, kapunk két sort,

mozi	*város*	*cím*
Corvin	Budapest	A hálózat
Kert	Budapest	A hálózat

ami megsérti a `cím város → mozi` függőséget.* □

* Szerkesztői megjegyzés: az adott példában a BCNF felbontás nem őrzi meg az eredeti függőségi rendszert, azaz a vetített függőségek nem alkotják az eredeti bázisát. Abban az esetben, ha a vetített függőségek az eredeti bázisát adják, a felbontást függőségőrző felbontásnak nevezzük. A példa mutatja, hogy nem mindig létezik függőségőrző BCNF felbontás.

A fenti probléma megoldása, hogy kicsit enyhítünk a BCNF követelményünkön, és megengedünk esetenként olyan relációsémát, mint a 3.42. példában, amelyet nem tudunk felbontani BCNF-ben levő relációkra úgy, hogy minden funkcionális függőséget tudnánk ellenőrizni egy reláción belül. Ezt az enyhébb feltételt nevezzük *harmadik normálforma* feltételnek:

- Az R reláció *harmadik normálformában* (3NF) van akkor és csak akkor, ha valahányszor létezik az R-ben egy $A_1A_2...A_n \to B$ nem triviális függőség, akkor vagy az $\{A_1, A_2, ..., A_n\}$ halmaz az R szuperkulcsa, vagy a B attribútum valamelyik kulcsnak az egyik eleme.

Megjegyezzük, hogy a 3NF feltétel és a BCNF feltétel közötti különbség „vagy a B attribútum valamelyik kulcsnak az egyik eleme" mondatrészben van, ami „felmenti" a `mozi → város`-hoz hasonló függőséget a 3.42. példában, mivel a jobb oldali `város` egy kulcsnak eleme.

Túllépi a könyv kereteit, hogy bebizonyítsuk, hogy a 3NF tulajdonképpen megfelel a céljainknak. Azaz egy relációsémát mindig fel tudunk bontani 3NF-ben levő sémákra úgy, hogy ne veszítsünk információt és minden funkcionális függőséget ellenőrizni tudjunk.* Ilyenkor ezek a relációk nincsenek BCNF-ben, ezért egy kevés redundancia megmaradhat a sémában.

Érdemes megjegyeznünk, hogy az a példa, amelyben 3NF-ben levő, de nem BCNF relációsémát kaptunk, nagyban különbözik a korábbi nem BCNF példáktól. Az egyik lényeges funkcionális függőség, a `mozi → város` azon alapul, hogy a mozi egy város egyértelmű objektuma. A másik függőség, a

`cím város → mozi`

abból a megfigyelésből származik, ahogyan a filmstúdiók a filmek elosztási politikáját működtetik a gyakorlatban. Általában a funkcionális függőségek két kategóriába esnek: azok, amelyek azon alapulnak, hogy egyértelmű objektumokat reprezentálunk, mint például filmek és stúdiók, és azok, amelyek a valós világ gyakorlatán alapulnak, mint például a filmek előjegyzése legfeljebb egy mozira történhet egy városon belül.

3.7.8. Feladatok

3.7.1. feladat: A következő relációsémákra és funkcionális függőségi halmazokra:

* a) $R(A, B, C, D)$ sémára és $AB \to C$, $C \to D$ és $D \to A$ funkcionális függőségekre.

* Szerkesztői megjegyzés: a megfelelő 3NF felbontást a 3.6.5. szakaszban bemutatott szigorú minimális bázis függőségeire való vetítéssel és esetleg még egy kulcsra való vetítéssel kapjuk.

* b) $R(A, B, C, D)$ sémára és $B \to C$, és $B \to D$ funkcionális függőségekre.

c) $R(A, B, C, D)$ sémára és $AB \to C$, $BC \to D$, $CD \to A$ és $AD \to B$ funkcionális függőségekre.

d) $R(A, B, C, D)$ sémára és $A \to B$, $B \to C$, $C \to D$ és $D \to A$ funkcionális függőségekre.

e) $R(A, B, C, D, E)$ sémára és $AB \to C$, $DE \to C$ és $B \to D$ funkcionális függőségekre.

f) $R(A, B, C, D, E)$ sémára és $AB \to C$, $C \to D$, $D \to B$ és $D \to E$ funkcionális függőségekre.

Végezzük el az alábbiakat:

i) Jelezzünk minden BCNF megsértést. Ne feledkezzünk el azokról a függőségekről sem, amelyek nincsenek az adott halmazban, de következnek belőle. Nem szükséges megadnunk azokat a megsértéseket, amelyek jobb oldalán egynél több attribútum áll.

ii) Bontsuk fel a relációkat, amelyiket szükséges, BCNF-ben levő relációkra.

iii) Jelezzünk minden 3NF megsértést.

iv) Bontsuk fel a relációkat, amelyiket szükséges, 3NF-ben levő relációkra.

3.7.2. feladat: Megemlítettük a 3.7.4. részben, hogy a BCNF-et megsértő funkcionális függőség jobb oldalát ki kell bővítenünk, ha lehetséges. Jóllehet ezt szabadon választott lépésnek tekintettük. Vegyünk egy R relációt, amelynek sémája $\{A, B, C, D\}$ attribútumhalmaz, és érvényesek az $A \to B$ és $A \to C$ funkcionális függőségek. Ezek megsértik a BCNF-et, mivel R egyetlen kulcsa $\{A, D\}$. Tegyük fel, hogy R felbontását az $A \to B$ szerint kezdjük. Ez ugyanarra az eredményre vezet-e végül, mint ha először kibővítettük volna a BCNF megsértést $A \to BC$-re? Miért?

! **3.7.3. feladat:** Legyen R ugyanaz a reláció, mint a 3.7.2. feladatban, de most az $A \to B$ és $B \to C$ funkcionális függőségek legyenek érvényesek. Újból hasonlítsuk össze R felbontását, amikor az $A \to B$ szerint kezdjük, azzal a felbontással, amikor $A \to BC$-t vesszük először. *Útmutatás:* Amikor felbontást végzünk, gondolnunk kell arra, hogy mely funkcionális függőségek érvényesek a felbontásból származó relációkra. Elegendő-e csak azokat az adott függőségeket venni, amelyekben szereplő attribútumok a felbontás egyik sémájában vannak benne? Mi a helyzet azokkal a függőségekkel, amelyek az adott függőségekből következnek?

! **3.7.4. feladat:** Tegyük fel, hogy van egy $R(A, B, C)$ relációsémánk az $A \to B$ funkcionális függőséggel. Azt is tegyük fel, hogy felbontottuk ezt $S(A, B)$ és $T(B, C)$ sémákra. Adjunk példát az R reláció egy olyan előfordulására, amelynek az S-re és T-re vett vetítéseit újból összekapcsolva, mint a 3.7.6. részben, nem ugyanazt a relációelőfordulást adja vissza.

! **3.7.5. feladat:** Tegyük fel, hogy az $R(A, B, C, D, E)$ relációt felbontjuk $S(A, B, C)$ relációra és más relációkra. Adjuk meg azokat a funkcionális függőségeket, amelyek S-ben fennállnak, ha az R függőségei a következők:

* a) $AB \to DE$, $C \to E$, $D \to C$ és $E \to A$.

b) $A \to D$, $BD \to E$, $AC \to E$ és $DE \to B$.

c) $AB \to D$, $AC \to E$, $BC \to D$, $D \to A$ és $E \to B$.

d) $A \to B$, $B \to C$, $C \to D$, $D \to E$ és $E \to A$.

Minden esetben elegendő megadni az S függőségeiből álló teljes halmaz minimális bázisát.

3.8. Többértékű függőségek

A „többértékű függőség" olyan állítás, amely azt fejezi ki, hogy két attribútum vagy attribútumhalmaz független egymástól. Látjuk később, hogy ez a feltétel a funkcionális függőség fogalmának az általánosítása, abban az értelemben, hogy minden funkcionális függőség implikálja a megfelelő többértékű függőséget. A független attribútumhalmazokkal kapcsolatosan vannak olyan helyzetek, amelyeket nem lehet a funkcionális függőségekkel megmagyarázni. Ebben a fejezetben feltárjuk a többértékű függőség okait és látni fogjuk hogyan használhatjuk őket az adatbázisséma tervezésében.

3.8.1. Attribútumfüggetlenségből származó redundancia

Előfordulhatnak olyan helyzetek, hogy olyan relációsémát tervezünk, amely bár BCNF-ben van, mégis marad benne egyfajta redundancia, amely nem a funkcionális függőséghez kapcsolódik. Amikor az ODL-ből relációkat hozunk létre a 3.2. részben tárgyalt módon, és egy osztálynak a kettő- vagy többértékű attribútumai függetlenek, ilyen esetekben általában redundáns BCNF sémákat kapunk.

3.43. példa: Tegyük fel, hogy a `Színész` osztályt úgy definiáltuk, hogy van egy neve, a lakcímeinek halmaza és azon filmek halmaza, amelyekben a színész szerepelt. A

```
interface Színész {
   attribute string név;
   attribute Set<
        Struct Lakcím {string város, string utca}
     >lakcím;
   relationship Set<Film> szerepelBenne
          inverse Film::szereplők;
};
```

3.38. ábra. *A színészek definíciójában a lakcímek és filmek halmaza*

név	*város*	*utca*	*cím*	*év*
C. Fisher	Hollywood	Maple St. 123	Csillagok háborúja	1977
C. Fisher	Malibu	Locust Ln. 5	Csillagok háborúja	1977
C. Fisher	Hollywood	Maple St. 123	A birodalom visszavág	1980
C. Fisher	Malibu	Locust Ln. 5	A birodalom visszavág	1980
C. Fisher	Hollywood	Maple St. 123	Jedi visszatér	1983
C. Fisher	Malibu	Locust Ln. 5	Jedi visszatér	1983

3.39. ábra. *A lakcímek halmaza független a filmektől*

definíció megegyező lenne a 2.5. ábrán szereplővel, csak a `lakcím` attribútum típusában különbözik. A `Színész` osztály definíciója a 3.38. ábrán található.

A 3.39. ábrán közvetlenül a 3.38. ábra definíciójából származó reláció néhány lehetséges sorát láthatjuk. A lakcímek halmazát pontosan úgy reprezentáltuk, mint ahogyan korábban a 3.8. ábrán szerepelt. Csak az akkori ábra sorait kibővítettük a `Film` osztály kulcsával, a `cím` és `év` attribútumoknak megfelelő komponensekkel. Ezek az attribútumok reprezentálják a `szerepelBenne` kapcsolatban az adott színészhez tartozó filmeket.

A 3.39. ábra középpontjában Carrie Fisher két vélt lakcíme és a három legismertebb filmje áll. Nincs értelme összekapcsolni egy lakcímet az egyik vagy másik filmmel. Így csak egyféleképpen tudjuk kifejezni, hogy a lakcímek és filmek a színészek független jellemzői, mégpedig minden lakcím előfordul minden filmmel. Ha pedig a lakcímeket és filmeket minden kombinációban megismételjük, akkor nyilvánvaló a redundancia. Például a 3.39. ábrán háromszor megismételjük Carrie Fisher lakcímét (minden filmjéhez külön-külön) és kétszer minden filmjét (minden lakcímhez egyszer).

A 3.39. ábrán javasolt relációséma mégsem sérti meg a BCNF-et. Valójában egyáltalán nem is létezik nem triviális függőség. Például a `város` attribútumot nem határozza meg funkcionálisan a másik négy attribútum. Előfordulhat, hogy egy színésznek két lakása van különböző városokban, de ugyanolyan nevű utcában. Ekkor lenne két olyan sor, amely a `városon` kívül minden más attribútumon megegyezne és a `városon` pedig különbözne. Így a

`név utca cím év → város`

funkcionális függőség nem érvényes a `Színészek` relációnkban. Az Olvasóra hagyjuk, hogy ellenőrizze le a többi esetben is, hogy az öt attribútumból egyiket sem határozza meg funkcionálisan a többi négy. Ez az észrevétel elegendő, hogy arra következtessünk, hogy egyáltalán nincs nem triviális függőség (az Olvasó azt is gondolja át, hogy miért érvényes a következtetésünk). Mivel nincs nem triviális funkcionális függőség, ebből következik, hogy az egyetlen kulcsot az öt attribútum együttesen alkotja, és nem sértheti semmi a BCNF-et. □

3.8.2. Többértékű függőségek definíciója

A *többértékű függőség* valamely R relációra vonatkozó állítás, miszerint amikor rögzítjük egy attribútumhalmaz értékeit, akkor bizonyos más attribútumok értékei függetlenek lesznek a relációban szereplő összes többi attribútum értékeitől. Pontosabban kifejezve azt mondjuk, hogy az

$$A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$$

többértékű függőség fennáll az R relációban, ha tekintve az R sorai közül azokat, amelyek minden egyes A-beli attribútumon ugyanazt a bizonyos értéket veszik fel, akkor ezekre a sorokra úgy találjuk, hogy a B-ken felvett értékek halmaza független az R-nek az A-któl és B-któl eltérő összes attribútumain felvett értékek halmazától. Még pontosabban azt mondjuk, hogy ez a többértékű függőség érvényes, ha:

Az R reláció minden olyan t és u sorpárjára, amelyek minden A-n megegyeznek, találhatunk az R-ben olyan v sort, amely megegyezik

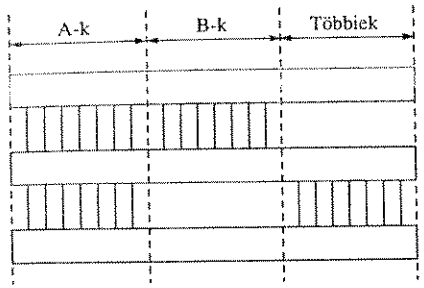
1. mind t-vel és mind u-val az A-kon,

2. t-vel a B-ken, és

3. u-val az R minden olyan attribútumán, amely nincs az A-kban vagy a B-ben.

Megjegyezzük, hogy ebben a szabályban a t-t és az u-t felcserélhetjük, aminek az a következménye, hogy létezik egy negyedik w sor, amely a B-ken megegyezik u-val és a többi eltérő attribútumokon pedig t-vel.* Végeredményben az A-knak valamely rögzített értékeire a B-khez és a többi attribútumokhoz rendelt értékeknek az összes lehetséges kombinációban elő kell fordulniuk a különböző sorokban. A 3.40. ábrában felrajzoltuk v hogyan viszonyul t-hez és u-hoz, amikor fennáll a többértékű függőség.

* Szerkesztői megjegyzés: érdemes megfigyelni, hogy a definíció pontosan azt követeli meg, hogy az $\{A_1...A_n B_1...B_m\}$ és $\{A_1...A_n C_1...C_k\}$ felbontás veszteségmentes összekapcsolású legyen, ahol $C_1, ..., C_k$ az R reláció A-k és B-k között nem szereplő attribútumai.



3.40. ábra. *A többértékű függőség biztosítja* v *létezését*

Általában feltehetjük, hogy egy többértékű függőségben az *A*-k és *B*-k (a függőség bal oldala és jobb oldala) diszjunkt halmazok. Itt is megengedett persze, mint a funkcionális függőségeknél, hogy hozzávegyünk az *A*-ból néhány attribútumot a jobb oldalhoz, ha kívánjuk. Megjegyezzük továbbá, hogy a funkcionális függőségektől eltérően, ahol több egyetlen attribútumból álló jobb oldalból kiindulva rövidítésképpen használhattunk egyetlen attribútumhalmazt a jobb oldalon, ebben az esetben a jobb oldalon álló attribútumhalmazokat közvetlenül úgy kell tekintenünk, ahogyan vannak. Később a 3.45. példában látni fogjuk, hogy nem minden esetben lehetséges a többértékű függőségek jobb oldalát egyszerű attribútumokra felbontani.

3.44. példa: A 3.43. példában láttunk egy többértékű függőséget, amelyet a jelöléseinkkel az alábbi formában tudunk kifejezni:

```
név →→ város utca
```

Azaz minden színésznévre a lakcímek halmaza kapcsolatban van a színész mindegyik filmjével. Például nézzük meg, hogyan alkalmazhatjuk a többértékű függőség formális definícióját a 3.39. ábra első és negyedik sorára:

név	*város*	*utca*	*cím*	*év*
C. Fisher	Hollywood	Maple St. 123	Csillagok háborúja	1977
C. Fisher	Malibu	Locust Ln. 5	A birodalom visszavág	1980

Ha ezt a két sort ebben a sorrendben *t*-vel és *u*-val jelöljük, akkor a többértékű függőség azt mondja ki, hogy az *R*-ben lennie kell olyan sornak, amelyben a név `C. Fisher`, a városon és utcán megegyezik az első sorral *t*-vel, és az egyéb attribútumain (`cím` és `év`) megegyezik az *u*-val. Valóban van ilyen sor, mégpedig a 3.39. ábra harmadik sora.

Hasonlóan vegyük most a fenti *t* sort másodiknak és az *u* legyen az első. Ekkor a többértékű függőség azt mondja ki, hogy az *R*-ben lennie kell olyan sornak, amely a `név, város, utca` attribútumokon megegyezik a másodikkal, és a `név, cím` és `év` attribútumokon megegyezik az elsővel. Ez a sor is létezik, méghozzá ez a 3.39. ábra második sora. □

3.8.3. Többértékű függőségekre vonatkozó szabályok

A 3.6. részben a funkcionális függőségekre tanult levezetési szabályokhoz hasonló számos szabály van a többértékű függőségekre is. Például a többértékű függőségre teljesül

- *A triviális függőségi szabály,* amely szerint, ha $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ többértékű függőség érvényes egy bizonyos relációban, akkor

  $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$

  is érvényes, ahol a *C*-k a *B*-ket jelentik és még egy vagy több *A*-t. Megfordítva, törölhetünk is attribútumukat a *B*-kből, ha azok benne vannak az *A*-kban, azaz

  $A_1A_2...A_n \twoheadrightarrow D_1D_2...D_r$

  többértékű függőség érvényes, ha a *D*-k a *B*-k közül olyanok, amelyek nincsenek benne az *A*-kban.

- *A tranzitivitási szabály,* amely szerint, ha érvényesek $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ és $B_1B_2...B_m \twoheadrightarrow C_1C_2...C_k$ többértékű függőségek egy bizonyos relációra, és semelyik *B* nem szerepel a *C*-k között, akkor

  $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$

  is érvényes.

A többértékű függőség nem tesz eleget a szétvághatósági/összevonhatósági szabályoknak.

3.45. példa: Nézzük meg újból a 3.39. ábrát, amelyben korábban már láttuk, hogy teljesül az alábbi többértékű függőség

```
név →→ város utca
```

Ha a szétvághatósági szabály igaz lenne a többértékű függőségekre is, akkor azt várnánk, hogy

```
név →→ utca
```

is teljesül. E többértékű függőség szerint minden színész lakcímében az `utca` független a többi attribútumtól, beleértve a várost is. Azonban ez az állítás hamis. Tekintsük például a 3.39. ábra első két sorát. A feltételezett többértékű függőség arra engedne következtetni, hogy a sorok az utcákban váltakozhatnak, azaz a

név	*város*	*utca*	*cím*	*év*
C. Fisher	Hollywood	Locust Ln. 5	Csillagok háborúja	1977
C. Fisher	Malibu	Maple St. 123	Csillagok háborúja	1977

sorok is benne lennének a relációban. De ezek nem valódi sorok, hiszen például Locust Ln. 5 Malibuban van, nem pedig Hollywoodban. □

Vannak új szabályok is a többértékű függőségekre. Először:

- Minden funkcionális függőség egyben többértékű függőség is. Azaz amennyiben $A_1A_2...A_n \rightarrow B_1B_2...B_m$, akkor $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$.

A bizonyításhoz legyen R egy tetszőleges reláció, amelyben $A_1A_2...A_n \rightarrow B_1B_2...B_m$ fennáll, és legyenek t és u az R-nek olyan sorai, amelyek az A-kon megegyeznek. Ahhoz, hogy megmutassuk

$$A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$$

érvényességét, azt kell megmutatnunk, hogy az R tartalmaz egy olyan v sort is, amely az A-kon megegyezik a t-vel és u-val, a B-ken t-vel, és a többi attribútumon pedig u-val. De v-nek vehetjük az u-t. Biztos, hogy az u megegyezik az A-kon t-vel és u-val, hiszen feltettük, hogy ez a két sor megegyezik az A-kon. Az $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőség biztosítja, hogy u megegyezik a B-ken t-vel. Természetesen a többi attribútumon pedig az u megegyezik önmagával. Így ha egy funkcionális függőség fennáll, akkor a megfelelő többértékű függőség is fennáll.

Egy másik szabály, aminek nincs megfelelője a funkcionális függőségek világában, a *komplementer szabály:*

- Ha $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ többértékű függőség az R relációban, akkor R kielégíti a $A_1A_2...A_n \twoheadrightarrow C_1C_2...C_k$ többértékű függőséget is, ahol a C-k az R összes attribútumai közül éppen azok, amelyek nincsenek sem az A-k sem a B-k között.

3.46. példa: Nézzük meg újból a 3.39. ábrán látható relációt, amelyre feltettük, hogy teljesíti az alábbi többértékű függőséget

```
név ->> város utca
```

A komplementer szabály szerint

```
név ->> cím év
```

többértékű függőségnek is teljesülnie kell ebben a relációban, mivel a cím és év attribútumok nem szerepeltek az első függőségben. A második függőség intuitíven azt jelenti, hogy minden színész esetén azoknak a filmeknek halmaza, amelyekben a színész játszott, független a színész lakcímétől. □

3.8.4. Negyedik normálforma

Azokat a többértékű függőségek által okozott redundanciákat, amelyeket korábban a 3.8.1. részben találtunk, megszüntethetjük, ha ezeket a függőségeket felhasználjuk relációkra vonatkozó új felbontási algoritmusban. Ebben a szakaszban bevezetünk egy új normálformát, az ún. „negyedik normálformát". Ebben a normálformában az összes „nem triviális" (abban az értelemben nem triviális, ahogyan korábban definiáltuk) többértékű függőségeket megszüntetjük úgy, ahogy tettük az összes olyan funkcionális függőséggel, amelyek megsértették a BCNF-et. Az eredményül kapott, felbontott relációkban nincs sem a funkcionális függőségekből származó olyan redundancia, mint a 3.7.1. részben szerepelt, sem a többértékű függőségekből származó redundancia, mint amilyent a 3.8.1. részben láttunk.

Egy R relációban érvényes $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ többértékű függőség *nem triviális,* ha:

1. Egyik B sincs az A-k között.

2. Az A-k és B-k együttesen sem tartalmazzák R összes attribútumát.

A „negyedik normálforma" feltétel lényegében olyan, mint a BCNF feltétel, csak a funkcionális függőségek helyett többértékű függőségekre alkalmazzuk. Formálisan:

- Egy R reláció *negyedik normálformában* (4NF) van, ha valahányszor az

  $$A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$$

  nem triviális többértékű függőség fennáll, akkor az $\{A_1, A_2, ..., A_n\}$ szuperkulcs.

Azaz, ha egy reláció 4NF-ben van, akkor minden nem triviális többértékű függőség valójában olyan funkcionális függőség, amelynek a bal oldala szuperkulcs. Megjegyezzük, hogy a kulcs és szuperkulcs fogalmak csak a funkcionális függőségektől függnek, a többértékű függőségeket is hozzávéve nem változik a „kulcs" definíciója.

3.47. példa: A 3.39. ábra relációja megsérti a 4NF feltételt. Például, a

```
név ->> város utca
```

nem triviális többértékű függőség, mégis a név önmagában nem szuperkulcs. Tulajdonképpen ehhez a relációhoz az egyetlen kulcs az összes attribútumból áll. □

A negyedik normálforma valóban a BCNF általánosítása. A 3.8.3. részben beláttuk, hogy minden funkcionális függőség egyben többértékű függőség is. Így minden, ami megsérti a BCNF-et, az megsérti a 4NF-et is. Másképpen fogalmazva, minden 4NF-ben levő reláció emiatt BCNF-ben van.

Vannak olyan BCNF-ben levő relációk, amelyek nem 4NF-ek. A 3.39. ábra jó példa erre. Az egyetlen kulcs ehhez a relációhoz mind az öt attribútumból áll, és emiatt nincs nem triviális funkcionális függőség. Így ez biztos, hogy BCNF-ben van. Ugyanakkor a 3.47. példában láttuk, hogy nincs 4NF-ben.

3.8.5. Negyedik normálformájú felbontás*

A 4NF-re való felbontási algoritmus teljesen analóg a BCNF-re való felbontás algoritmusával. Keressünk olyan nem triviális $A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$ többértékű függőséget, amely megsérti a 4NF feltételt, azaz $\{A_1, A_2, ..., A_n\}$ nem szuperkulcs. Megjegyezzük, hogy ez a többértékű függőség vagy valódi funkcionális függőség, vagy a megfelelő $A_1A_2...A_n \rightarrow B_1B_2...B_m$ funkcionális függőségből származik, mivel minden funkcionális függőség egyben többértékű függőség. Ekkor felbontjuk az R relációsémáját két sémára:

1. Az A-k és a B-k.

2. Az A-k és az R minden olyan attribútuma, amely nincs A-ban vagy B-ben.

3.48. példa: Folytassuk a 3.47. példát. Észrevettük, hogy a

```
név →→ város utca
```

megsérti a 4NF feltételt. A fenti felbontási szabály szerint az ötattribútumos sémát két sémával helyettesítjük, az egyik séma az említett többértékű függőség bal és jobb oldalán szereplő három attribútumból áll, a másik séma tartalmazza a bal oldali `név` attribútumot, plusz azokat az attribútumokat, amelyek nem fordultak elő ebben a függőségben. Ezek az attribútumok a `cím` és az `év`. Így a felbontás eredménye a következő két séma:

```
{név, város, utca}
{név, cím, év}
```

Egyik sémában sincs nem triviális többértékű (vagy funkcionális) függőség, így mind a kettő 4NF-ben van. Megjegyezzük, hogy a `{név, város, utca}` sémájú relációban a

```
név →→ város utca
```

* Szerkesztői megjegyzés: jóllehet a szövegkörnyezet is sugallja, mégsem árt hangsúlyozni, hogy most a relációsémák megadásánál egyaránt előírhatunk funkcionális és többértékű függőségeket.

Többértékű függőségek vetítése

A negyedik normálforma felbontásnál meg kell keresnünk, hogy mely többértékű függőségek érvényesek a felbontás eredményeként kapott relációkban. Szeretnénk könnyen megtalálni ezeket a függőségeket. Sajnos itt nincs olyan hasonlóan egyszerű ellenőrzés, mint a funkcionális függőségekre volt az attribútumhalmazok lezárásának kiszámolása (a 3.6.3. részben).Valójában még a funkcionális és többértékű függőségekre érvényes együttes levezetési szabályok teljes halmaza is igen összetett, meghaladja ennek a könyvnek a kereteit. A fejezet végén található Irodalomjegyzékben javasoljuk a további publikációkat, amelyek ezzel a témával foglalkoznak.

Szerencsére gyakran megkaphatjuk a felbontás eredményeire a megfelelő többértékű függőségeket a tranzitivitási szabály, komplementer szabály és a metszet szabály (3.8.7.b) feladat) segítségével. Javasoljuk, hogy az Olvasó nézze meg a példákat és próbálja megoldani a feladatokat.

triviális többértékű függőség, mivel az összes attribútumot tartalmazza. Hasonlóan a `{név, cím, év}` sémájú relációban a

```
név →→ cím év
```

triviális többértékű függőség. Ha a felbontásban legalább az egyik séma nincs 4NF-ben, akkor folytatnunk kell a nem 4NF séma (illetve sémák) felbontását. □

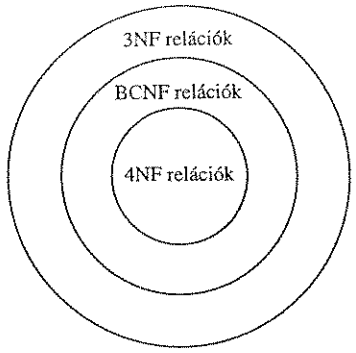
Ugyanúgy, mint a BCNF felbontásnál, minden felbontási lépésben egyre kevesebb attribútumból álló sémát kapunk, mint amilyenből kiindultunk, így végül olyan sémákhoz jutunk, amelyeket már tovább nem kell felbontani, azaz már 4NF-ben vannak. A felbontás helyességének igazolása, amelyet a 3.7.6. rész tartalmazott, végigvihető a többértékű függőségekre is. Amikor felbontunk egy relációt az

$$A_1A_2...A_n \twoheadrightarrow B_1B_2...B_m$$

többértékű függőség miatt, ez a függőség elegendő ahhoz, hogy igazolni tudjuk azt az állítást, hogy az eredeti reláció a felbontásban szereplő relációkból visszaállítható.

3.8.6. Normálformák közötti kapcsolatok

Említettük korábban, hogy a 4NF-ből következik a BCNF, amiből következik a 3NF. A 3.41. ábrán szemléltettük, hogy a normálformákat kielégítő relációelőfordulások halmazai között mi a kapcsolat. Ha a sorok halmaza eleget tesz a 4NF feltételnek, akkor biztos, hogy eleget tesz a másik két normálforma feltételnek is, és ha eleget tesz a BCNF feltételnek, akkor biztos, hogy eleget tesz a 3NF-nek. A sémára feltett funkcio-



3.41. ábra. *A 4NF-ből következik a BCNF, amiből következik a 3NF*

nális függőségektől függ, hogy lehetnek-e olyan sorhalmazok, amelyek 3NF-ben vannak, de nincsenek BCNF-ben. Hasonlóan, bizonyos funkcionális és többértékű függőségek feltételezése esetén létezhetnek olyan BCNF sorhalmazok, amelyek nincsenek 4NF-ben.

A normálformák összehasonlításának másik módja az, hogy adott normálformára való felbontás eredményéül kapott relációk halmaza mit biztosít. Ezt a 3.42. ábrán összegeztük. A BCNF (és így a 4NF) megszünteti a redundanciát és más problémákat, amelyeket a funkcionális függőségek okoznak, ugyanakkor csak a 4NF szünteti meg azt a további redundanciát, amelyet olyan nem triviális többértékű függőségek okoznak, amelyek nem funkcionális függőségek. Gyakran elegendő a 3NF ilyen redundancia megszüntetésére, de vannak olyan esetek, amikor nem. A 3NF-re való felbontás mindig megválasztható úgy, hogy megőrizze a funkcionális függőségeket, azaz elegendő megkövetelésük csupán a felbontott relációkra (ennek az algoritmusát kihagytuk ebből a könyvből). A BCNF nem biztosítja a funkcionális függőségek megőrzését, és egyik normálforma sem biztosítja a többértékű függőségek megőrzését, ám tipikus esetekben a függőségek is érvényben maradnak.

Jellemzői	*3NF*	*BCNF*	*4NF*
Megszünteti a funkcionális függőségekből eredő redundanciát	Gyakran	Igen	Igen
Megszünteti a többértékű függőségekből eredő redundanciát	Nem	Nem	Igen
Megőrzi a funkcionális függőségeket	Igen	Lehet	Lehet
Megőrzi a többértékű függőségeket	Lehet	Lehet	Lehet

3.42. ábra. *A normálformák és a felbontásaik jellemzői*

3.8.7. Feladatok

* **3.8.1. feladat:** Legyen $R(A, B, C)$ egy reláció az $A \twoheadrightarrow B$ többértékű függőséggel. Ha tudjuk, hogy az (a, b_1, c_1), (a, b_2, c_2) és (a, b_3, c_3) sorok benne vannak az R aktuális előfordulásában, akkor milyen további soroknak kell még az R-ben lenniük?

* **3.8.2. feladat:** Vegyük azt a relációt, amelyben feljegyezzük a személyek nevét, személyi számát és a születésnapját. Továbbá a személy összes gyerekének a nevét, személyi számát és a születésnapját, és amennyiben autóval vagy autókkal rendelkezik a személy, az autóinak a sorozatszámát és gyártmányát. Pontosabban ennek a relációnak a sorai az összes olyan

$(n, szsz, szd, gyn, gyszsz, gyszd, as, agy)$

sor, amelyre

1. n a személy neve, akinek a személyi száma $szsz$.
2. szd az n születésnapja.
3. gyn az n egyik gyerekének neve.
4. $gysz$ az gyn személyi száma.
5. $gyszd$ az gyn születésnapja.
6. as az n egyik autójának sorozatszáma.
7. agy az as sorozatszámú autó gyártmánya.

Erre a relációra:

a) Adjuk meg azokat a funkcionális és többértékű függőségeket, amelyektől elvárjuk, hogy érvényesek legyenek.

b) Adjunk egy javaslatot a reláció 4NF-ban való felbontására.

3.8.3. feladat: A következő relációsémákra és funkcionális függőségi halmazokra nézve:

* a) $R(A, B, C, D)$ sémára és $A \twoheadrightarrow B$, $A \twoheadrightarrow C$ többértékű függőségekkel.

b) $R(A, B, C, D)$ sémára és $A \twoheadrightarrow B$, $B \twoheadrightarrow CD$ többértékű függőségekkel.

c) $R(A, B, C, D)$ sémára és $AB \twoheadrightarrow C$ többértékű függőséggel és $B \rightarrow D$ funkcionális függőséggel.

d) $R(A, B, C, D, E)$ sémára és $A \twoheadrightarrow B$ és $AB \twoheadrightarrow C$ többértékű függőségekkel és $A \rightarrow D$ és $AB \rightarrow E$ funkcionális függőségekkel.

Végezzük el az alábbiakat:

i) Keressük meg az összes 4NF-et megsértő függőséget.

ii) Bontsuk fel a fenti relációsémákat 4NF-ben lévő relációsémákra.

! **3.8.4. feladat:** A 2.3.2. feladatban négy különböző feltételt adtunk meg a *Születések* kapcsolatra. Mindegyikhez soroljuk fel azokat a többértékű függőségeket (a funkcionális függőségektől eltérőeket), amelyektől elvárhatjuk, hogy érvényesek az eredmény relációban.

3.8.5. feladat: Adjunk heurisztikus érveket arra, hogy miért nem vártuk a 3.43. példa öt attribútumának egyikétől sem, hogy a másik négy funkcionálisan meghatározza.

! **3.8.6. feladat:** Felhasználva a többértékű függőség definícióját, mutassuk meg, hogy a komplementer szabály érvényes.

! **3.8.7. feladat:** Mutassuk meg az alábbi szabályokat a többértékű függőségekre:

* a) Az *egyesítés szabály*. Ha X, Y és Z attribútumhalmazok, $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y \cup Z)$.

b) A *metszet szabály*. Ha X, Y és Z attribútumhalmazok, $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y \cap Z)$.

c) A *különbség szabály*. Ha X, Y és Z attribútumhalmazok, $X \twoheadrightarrow Y$ és $X \twoheadrightarrow Z$, akkor $X \twoheadrightarrow (Y - Z)$.

d) *Triviális többértékű függőségek*. Ha $Y \subseteq X$, akkor $X \twoheadrightarrow Y$ bármely relációban érvényes.

e) *Másik lehetőség triviális többértékű függőségekre*. Ha $X \cup Y$ tartalmazza az R reláció minden attribútumát, akkor $X \twoheadrightarrow Y$ érvényes az R relációban.

f) *A bal és jobb oldal közös attribútumainak törlése*. Ha $X \twoheadrightarrow Y$ fennáll, akkor érvényes $X \twoheadrightarrow (Y - X)$.

! **3.8.8. feladat:** Adjunk meg olyan ellenpélda relációkat, amelyek azt mutatják, hogy a következő szabályok nem érvényesek:

* a) Ha $A \twoheadrightarrow BC$, akkor $A \twoheadrightarrow B$.

b) Ha $A \twoheadrightarrow B$, akkor $A \rightarrow B$.

c) Ha $AB \twoheadrightarrow C$, akkor $A \twoheadrightarrow C$.

! **3.8.9. feladat:** Az ODL átírása relációkká gyakran vezet többértékű függőségekhez. Adjunk meg néhány elvet, amellyel feltárhatjuk a többértékű attribútumokból és kapcsolatokból származó többértékű függőségeket, ha a 3.2.2. és 3.2.5. részek relációséma-stratégiáit követjük.

3.9. A példaként használt adatbázisséma

Láttuk eddig, hogy milyen problémák merülhetnek fel, amikor közvetlenül egy ODL tervet vagy egy E/K tervet írunk át relációkká, és láttuk mit tehetünk az előforduló problémákkal, anomáliákkal. Rögzítsünk most egyetlen relációs adatbázissémát, amelyet a könyv hátralevő részében használunk, abban a részben, ami arról szól, hogy a felhasználónak milyen adatbázis-programozási lehetőségei vannak. Az adatbázissémánk felhasználja a korábbi filmek, színészek és stúdiók példákat. Olyan normalizált relációkat vettünk, melyek hasonlóak ahhoz, amiket az előző részben terveztünk. Néhány attribútum azonban eddig nem szerepelt még a példáinkban, és van egy új reláció, a `GyártásIrányító`, amely korábban még nem fordult elő. Ezekkel a változtatásokkal az a célunk, hogy lehetőségünk legyen később a különböző adattípusok tanulmányozására és az információ különböző reprezentálási módjainak elemzésére a 4. fejezettől a 8. fejezetig szereplő példákban. A 3.43. ábra mutatja a sémánkat.

A sémánk öt relációból áll. Minden relációnak felsoroltuk az attribútumait és az attribútumokhoz rendelt értéktartományokat. A reláció kulcsát alkotó attribútumokat csupa nagybetűvel jelöltük a 3.43. ábrán, mégis amikor a szövegben hivatkozunk rájuk a szokásos kisbetűs írásmódot használjuk. Például a `SzerepelBenne` reláció kulcsát mind a három attribútuma alkotja. A `Film` relációnak hat attribútuma van, a `cím` és `év` attribútumok együtt alkotják a `Film` kulcsát, ahogyan eddig is. A `cím` attribútum karaktersorozat, az `év` pedig egész.

Összevetve az eddigiekkel a főbb változtatások a sémákban a következők:

- Van egy *azonosító* jelölés a gyártásirányítókhoz, akik a stúdióelnökök és filmproducerek. Ez az azonosító egyértelmű egyedi egész, amit valamilyen külső hivatal tart karban, ami lehet a gyártásirányítás iktatója vagy egy „szervezet".

```
Film (
   CÍM: string,
   ÉV: integer,
   hossz: integer,
   színes: boolean,
   stúdióNév: string,
   producerAzon#: integer)

SzerepelBenne (
   FILMCÍM: string,
   FILMÉV: integer,
   SZÍNÉSZNÉV: string)

FilmSzínész (
   NÉV: string,
   cím: string,
   nem: char,
   születésnap: date)

GyártásIrányító (
   név: string,
   cím: string,
   AZONOSÍTÓ#: integer,
   nettóBevétel: integer)

Stúdió (
   NÉV: string,
   cím: string,
   elnökAzon#: integer)
```

3.43. ábra. *Egy példaként használható adatbázisséma a filmekről*

- A gyártásirányítók kulcsaként az azonosítót használjuk, de mivel a filmszínészeknek nincs mindig azonosítójuk, így a színészek kulcsaként továbbra is a `név` attribútumot használjuk. Ez nem mindig jó a valóságban, ugyanis két színésznek lehet azonos neve, mégis ezt az utat követjük, hogy bemutassuk a különféle lehetőségeket.

- Bevezettük a producert a filmek egy tulajdonságaként. Ezt az információt a `producerAzon` új attribútummal reprezentáltuk a `Film` relációban. Ez az attribútum a producer azonosítója. A producerek benne vannak a gyártásirányítók között, hasonlóan a stúdióelnökökhöz. Ezenkívül még mások is lehetnek a `GyártásIrányító` relációban.

- A `Film` reláció `szalagFajta` attribútumának a típusát megváltoztattuk a felsorolt típusból logikai értékűre, így ez `színes` attribútumként szerepel, és az értéke igaz, ha színes, és hamis, ha fekete-fehér a film. Azért írtuk át, mert nem minden adatbázisnyelv támogatja a felsorolás típusokat.

- A filmszínészekhez hozzávettük a `nem` attribútumot. Ez „karakter" típusú: `N` ha nő, `F` ha férfi. Hozzávettük még a `születésnap` „dátum" típusú attribútumot is (a legtöbb forgalomban levő adatbázisrendszer támogatja a speciális dátum típust, de lehet akár karaktersorozat is, ha nekünk úgy tetszik).

- Minden címet inkább karaktersorozatnak választottunk, nem pedig város és utca párnak. Ennek a célja, hogy a különböző relációkban szereplő címeket könnyebben össze tudjuk hasonlítani, és hogy egyszerűsítsük a címeken a műveleteket.

Befejezésül rövid megjegyzéseket fűzünk az öt relációhoz, az attribútumaikhoz, és hogy hogyan származtak a korábbi ODL vagy E/K tervekből.

1. A `Film` a 3.36. példában szereplő `Film` reláció felbontásának egyik relációja, amihez hozzávettük a `producerAzon#` attribútumot, amely a film producerét reprezentálja.

2. A `SzerepelBenne` a 3.36. példa felbontásának egy másik relációja. Ugyanerre a relációra lenne szükségünk akkor is, ha előbb az ODL `Színész` osztályt átírtuk volna ugyanilyen nevű relációvá, amit azután BCNF-re hoztunk volna. Azaz a 2.5. ábra ODL definíciójából kiindulva, megkapnánk a `Színész` relációt a `név`, `lakcím`, `cím` és `év` attribútumokkal. Az utóbbi kettő a `szerepelBenne` kapcsolatot reprezentálja. Ekkor a {`név`, `cím`, `év`} kulcs, de a `név` → `lakcím` funkcionális függőség fennáll a relációban. Így a relációt fel kellett bontanunk két sémára, a {`név`, `lakcím`} sémára, amelyet kibővítve kaptuk a `FilmSzínész` relációnkat, és a {`név`, `cím`, `év`} sémára, amely lényegében a `SzerepelBenne` relációnk. A `SzerepelBenne` relációnk reprezentálja a 2.8. ábra E/K diagram `SzerepelBenne` kapcsolatát is.

3.10. Összefoglalás

- *Relációs modell:* A relációk információt reprezentáló táblák. Az oszlopok fejlécében attribútumok vannak, minden attribútumhoz tartozik egy tartomány vagy adattípus. A táblasorokat soroknak hívjuk és a reláció minden attribútumához tartozik a sornak egy komponense.

- *Sémák:* A relációnév a reláció attribútumaival együtt adja a relációsémát. A relációsémák összessége az adatbázisséma. Egy relációhoz vagy relációk összességéhez tartozó adatokat az adott relációsémához vagy adatbázissémához tartozó előfordulásnak nevezzük.

- *Egyedhalmazok átírása relációkká:* Az egyedhalmazhoz megfeleltetett relációban az egyedhalmaz minden attribútumához tartozik egy attribútum. Egy kivétel van,

az E gyenge egyedhalmaz, amelyhez tartozó relációban olyan attribútumoknak is benne kell lenniük, amelyek más egyedhalmaz kulcsattribútumai és az E egyedek azonosítását segítik.

- *Kapcsolatok átírása relációkká:* Egy E/K kapcsolat átírásából származó reláció tartalmazza azokat az attribútumokat, amelyek a kapcsolatban részt vevő minden egyes egyedhalmaz kulcsattribútumaiból származnak.

- *ODL osztályok átírása relációkká:* Egy ODL C osztálynak átírásából származó relációban az osztály minden attribútumához megfeleltetünk egy attribútumot. Továbbá a relációnak lehetnek olyan attribútumai, amelyek a C osztállyal kapcsolatban álló D osztály kulcsaiban szerepelnek. Mivel az ODL kapcsolatoknak vannak inverzei, azt javasoljuk, hogy a C-ről D-re sok-egy kapcsolatot a C-vel tároljuk, ne pedig a D-vel.

- *ODL többértékű kapcsolatok átírása relációkká:* Sok-sok kapcsolatok bármelyik osztállyal együtt tárolhatók, csak ennek az eredményeképpen a reláció sorainak száma nagyon megnövekszik. Ezt a hibát a relációs tervezésben a normalizációs folyamattal tudjuk megszüntetni. Egy másik lehetőség, hogy az ODL sok-sok kapcsolatait egy külön relációval reprezentáljuk, ahogy ezt az E/K modell esetében.

- *Alosztályok átírása relációkká:* Az egyik lehetőség az, hogy az egyedeket vagy objektumokat besoroljuk a különböző alosztályokba és minden alosztályhoz létrehozunk egy relációt az összes szükséges attribútummal. A másik lehetőség, hogy az összes egyedet vagy objektumot egyetlen főrelációban reprezentáljuk, mégpedig csak a legáltalánosabb osztály attribútumaival. Az alosztályok egyedei vagy objektumai speciális relációkban is szerepelnek annak megfelelően, melyik alosztályhoz tartoztak. Ezekhez a relációkhoz csak az általános osztály kulcsattribútumai és az alosztály speciális attribútumai tartoznak.

- *Funkcionális függőség:* A funkcionális függőség egy állítás, mely szerint, ha egy reláció két sora megegyezik egy adott attribútumhalmazon, akkor meg kell egyezniük a másik adott attribútumhalmazon is.

- *Kulcsok:* A reláció szuperkulcsa egy attribútumhalmaz, amely funkcionálisan meghatározza a reláció összes attribútumát. A kulcs egy olyan szuperkulcs, amelynek egyik valódi részhalmaza sem határozza meg funkcionálisan az összes attribútumot.

- *Funkcionális függőségekre vonatkozó szabályok:* Több szabály van, amelynek a segítségével következtethetünk arra, hogy egy $X \to A$ funkcionális függőség érvényes-e tetszőleges olyan relációelőfordulásban, amely eleget tesz egy adott funkcionális függőségi halmaznak. A legegyszerűbb lehetőség arra, hogy ellenőrizzük az $X \to A$ érvényességét az, hogy kiszámoljuk az X lezárását, azaz felhasználva az adott függőségeket kibővítjük X-et, és ellenőrizzük, hogy amit kaptunk tartalmaza-e az A-t.

- *Relációk felbontása:* Információvesztés nélkül ketté tudunk bontani egy relációsémát, ha a mindkét sémában szereplő attribútumok a felbontott relációk közül legalább az egyikben szuperkulcsot alkotnak.

- *Boyce-Codd normálforma:* Egy reláció BCNF-ben van, ha csak azok a nem triviális funkcionális függőségek érvényesek, amelyek szerint valamelyik szuperkulcs funkcionálisan meghatároz egy vagy több attribútumot. Tetszőleges relációt információveszteség nélkül fel tudunk bontani BCNF-ben lévő relációk összességére. A BCNF legnagyobb haszna, hogy megszünteti azokat a redundanciát, amelyet a funkcionális függőségek jelenthetnek.

- *Harmadik normálforma:* Előfordul gyakran, hogy a BCNF felbontás nem őrzi meg a funkcionális függőségeket. 3NF-nek nevezzük a BCNF enyhébb alakját, amelyben előfordulhatnak olyan $X \to A$ funkcionális függőségek, ahol ha az X nem szuperkulcs, akkor az A valamely kulcsnak eleme. A 3NF nem mindig szünteti meg a funkcionális függőségekből származó összes redundanciát, de elég gyakran megteszi.

- *Többértékű függőség:* A többértékű függőség egy állítás, mely szerint egy reláció két attribútumhalmazának az értékhalmazai az összes lehetséges kombinációban előfordulnak. A többértékű függőségeket általában az okozza, amikor kettő vagy több többértékű attribútummal vagy kapcsolattal rendelkező ODL osztályt írunk át relációvá.

- *Negyedik normálforma:* A többértékű függőségek is okozhatnak redundanciát egy relációban. A 4NF hasonló a BCNF-hez, de megtiltja a nem triviális többértékű függőségeket is (hacsak nem azokról a funkcionális függőségekről van szó, amelyeket a BCNF megenged). Bármely relációt veszteségmentesen fel lehet bontani 4NF-re.

3.11. Irodalomjegyzék

Codd klasszikus cikke a relációs modellről [7]. Ebben a cikkben vezeti be a funkcionális függőség ötletét és a reláció alapvető fogalmát. A harmadik normálformák is ebben a cikkben szerepelnek, a Boyce-Codd normálformák pedig Codd egyik későbbi cikkében [8].

A többértékű függőségeket és a negyedik normálformát Fagin definiálta [13]. A többértékű függőség ötlete egymástól függetlenül [9] és [24] cikkekben is megjelent.

Armstrong tanulmányozta először a funkcionális függőségekre vonatkozó levezetési szabályokat [2]. A funkcionális függőségek szabályai, amit mi is tárgyaltunk „Armstrong-axiómák" elnevezéssel és a többértékű függőségek szabályai a [4] cikkből származnak.

Több algoritmust, illetve egyes algoritmusok működésének a bizonyításait nem adtuk meg ebben a könyvben. Ilyenek például, hogy a funkcionális függőségek levezetéséhez megadott lezárási algoritmus miért működik, hogyan lehet többértékű függőségekre következtetni, hogyan kell a többértékű függőségeket vetíteni a dekomponált relációkra, hogyan lehet olyan 3NF dekompozíciót előállítani, amely rendelkezik a funkcionális függőségek ellenőrizhetőségi tulajdonságával is. Ezek és a függőségekkel összefüggő egyéb vizsgálatok [21]-ben találhatók meg.

Akik kíváncsiak arra, hogy lehet matematikai precizitással kezelni az adatbázisok témakörét, azoknak az [1]-t ajánljuk.

Az Irodalomjegyzékben feltüntettük a témával kisebb-nagyobb mértékben foglalkozó magyar nyelven megjelent könyveket is.

1. Abiteboul, S., R. Hull, V. Vianu, *Foundations of Databases,* Addison-Wesley Publishing Company, New York, 1988.

2. Armstrong, W. W., „Dependency structures of database relationships", *Proceedings of the 1974 IFIP Congress,* pp. 580–583, 1974.

3. Batini, C., S. Ceri, S. B. Navathe, *Conceptual Database Design. An Entity-Relationship Approach,* Benjamin/Cummings Publishing Company, Redwood City, 1992.

4. Beeri, C., R. Fagin, J. H. Howard, „A complete axiomatization for functional and multivalued dependencies", *ACM SIGMOD International Conference on Management of Data,* pp. 47–61, 1977.

5. Bernstein, P. A., „Syntesizing third normal form relations from functional dependencies", *ACM Transactions on Database Systems* **1**:4, pp. 277–298, 1976.

6. Ceri, S., G. Gottlob, L. Tanca, *Logic Programming and Databases,* Springer-Verlag, Berlin, 1990.

7. Codd, E. F., „A relational model of data for large shared data banks", *Communications of the ACM,* **13**:6, pp. 377–387, 1970.

8. Codd, E. F., „Further normalization of the data base relational model", *Database Systems* (R. Rustin, szerk.), Prentice-Hall, Englewood Cliffs, NJ, 1972.

9. Delobel, C., „Normalization and hierarchical dependencies in the relational model", *ACM Transactions on Database Systems* **3**:3, pp. 201–222, 1978.

10. Demetrovics J., „Relációs adatmodell logikai és strukturális vizsgálata", *MTA SZTAKI Tanulmányok,* 114, Budapest, 1980.

11. Demetrovics J., J. Denev, R. Pavlov, *A számítástudomány matematikai alapjai,* Tankönyvkiadó, Budapest, 1985.

12. Elmasri, R., S. B. Navathe, *Fundamentals of Database Systems,* Benjamin/ Cummings Publishing Company, Redwood City, 1994.

13. Fagin, R., „Multivalued dependencies and a new normal form for relational databases", *ACM Transactions on Database Systems* **2**:3, pp. 262–278, 1977.

14. Halassy B., *Adatmodellezés, Adatbázis-tervezés,* SZÁMOK, Budapest, 1980.

15. Halassy B., *Adatmodellezés a rendszerfejlesztésben,* SZÁMALK, Budapest, 1983.

16. Halassy B., *Az adatbázis-tervezés alapjai és titkai,* IDG Magyarországi Lapkiadó Kft., Budapest, 1994.

17. Kim, W., *Modern Database Systems,* Addison-Wesley, 1995.

18. Korth H. F., A. Silberschatz, *Database systems concepts,* McGraw-Hill Inc., 1991.

19. Paredaens, J., P. De Bra, M. Gyssens, D. Van Gucht, *The structure of the relational database model,* Springer-Verlag, Berlin, 1989.

20. Szelezsán J., *Adatbázisok,* LSI Oktatóközpont, Gábor Dénes Főiskola, Budapest, 1997.

21. Ullman, J. D., *Principles of Database and Knowledge-Base Systems,* Volume I, Computer Science Press, New York, 1988.

22. Quittner P., *Adatbázis-kezelés a gyakorlatban,* Akadémia kiadó, Budapest, 1993.

23. Vathy Á., Németh K., *Adatmodellezési feladatok,* Veszprémi Egyetem, 1996.

24. Zaniolo, C., M. A. Melkanoff, „On the design of relational database schemata", *ACM Transactions on Database Systems* **6**:1, pp. 1–47, 1981.

4. fejezet

Műveletek a relációs modellben

Ebben a fejezetben az adatbázisokat a felhasználó szemszögéből kezdjük el tanulmányozni. A felhasználó számára gyakran a legfontosabb probléma az adatbázis *lekérdezése*, amely nem más, mint olyan programok írása, amelyek az adatbázis aktuális előfordulására vonatkozó kérdésekre felelnek. Ebben a fejezetben az adatbázis-lekérdezések problémáját egy absztrakt nézőpontból mutatjuk be, megadva a fő lekérdező operátorokat.

Az ODL-ben a metódusok segítségével tetszőleges adatműveletet meg lehet adni, az E/K modell nem foglal magában adatok manipulálására szolgáló módszert, ezzel szemben a relációs modell tartalmaz egy szabványos adatműveletekből álló halmazt. Éppen ezért az adatbázis-műveleteknek ez az absztrakt bemutatása a relációs modellre és a modell műveleteire helyezi a hangsúlyt. Ezek a műveletek kifejezhetők egyrészt algebrai formában, amelyet „relációs algebrának" nevezünk, másrészt logikai formában, amelyet „Datalognak" nevezünk. Ebben a fejezetben mindkét jelölési módot bemutatjuk.

A későbbi fejezetek lehetőséget biztosítanak arra, hogy betekintsünk azokba a nyelvekbe és speciális lehetőségekbe, amelyeket a mai kereskedelmi adatbázisrendszerek biztosítanak a felhasználók számára. Az absztrakt lekérdező operátorok elsősorban az SQL lekérdezőnyelvben jelennek meg mint műveletek. (Lásd az 5–7. fejezeteket.) Mindamellett megtalálhatók a 8. fejezetben említett OQL nyelvben is.

4.1. Relációs algebra

A relációkon értelmezett műveletek tanulmányozásának elkezdéséhez ismerkedjünk meg egy speciális algebrával, a *relációs algebrával*, amely néhány egyszerű, de hatható módszert ad arra nézve, hogy miként építhetünk új relációkat a régi relációkból. A relációs algebrai *kifejezések* alapjait képezik a relációk, mint operandusok. Egy reláció megadható a nevével (pl. R vagy `Film`) vagy közvetlenül, sorainak egy listájával. Ezután, alkalmazva az alábbiakban bemutatásra kerülő operátorokat a relációkra vagy egyszerűbb relációs algebrai kifejezésekre, fokozatosan egyre bonyolultabb kifejezéseket építhetünk fel. Egy *lekérdezés* tulajdonképpen egy relációs algebrai kifejezés. Ilyenformán a relációs algebra az első konkrét példánk lekérdezőnyelvre.

A relációs algebrai műveletek négy osztályba sorolhatók:

1. A hagyományos halmazműveletek – egyesítés, metszet és különbség – relációkra alkalmazva.

2. Műveletek, amelyek a reláció egyes részeit eltávolítják: a „kiválasztás" kihagy bizonyos sorokat, a „vetítés" bizonyos oszlopokat hagy ki.

3. Műveletek, amelyek két reláció sorait kombinálják: a „Descartes-szorzat", amely a relációk sorait párosítja az összes lehetséges módon és a különböző típusú „összekapcsolási" műveletek, amelyek szelektíven párosítják össze a két reláció sorait.

4. Egy művelet, az „átnevezés", amelyik nem befolyásolja a reláció sorait, de megváltoztatja a reláció sémáját, azaz az attribútumok neveit és/vagy a reláció nevét.

Ezek a műveletek nem elegendők a relációkon elvégezhető összes számításhoz, tulajdonképpen eléggé korlátozott lehetőséget biztosítanak. Ennek ellenére mégis jelentős részét tartalmazzák mindazoknak a műveleteknek, melyeket az adatbázisokkal valójában tenni akarunk. Amint az 5. fejezetben látni fogjuk, ezek a műveletek nagy részét képezik a szabványos relációs lekérdezőnyelvnek, az SQL-nek. Mindamellett a 4.6. és 4.7. alfejezetekben röviden megvizsgáljuk azokat a számítási lehetőségeket, amelyek adottak az olyan lekérdezőnyelvekben mint az SQL, és mégsem részei a relációs algebrának.

4.1.1. Relációkon értelmezett halmazműveletek

Halmazokon a három leggyakoribb művelet az egyesítés, metszet és különbség. Feltételezzük, hogy az olvasó ismeri ezeket a műveleteket, amelyeket a következő módon definiálunk tetszőleges R és S halmazok esetén:

- $R \cup S$, R és S *egyesítése*, azon elemek halmaza, amelyek vagy az R-ben vagy az S-ben vannak. Egy elem csak egyszer szerepel az egyesítésben, még akkor is, ha jelen van az R-ben is és az S-ben is.

- $R \cap S$, R és S *metszete*, azon elemek halmaza, amelyek az R-ben és az S-ben is benne vannak.

- $R - S$, R és S *különbsége*, azon elemek halmaza, amelyek benne vannak R-ben, de nincsenek S-ben. Figyeljük meg, hogy $R - S$ nem ugyanaz, mint $S - R$; az utóbbi azon elemek halmaza, amelyek benne vannak S-ben, de nincsenek R-ben.

név	*cím*	*nem*	*születésnap*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88

Az R reláció

név	*cím*	*nem*	*születésnap*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Harrison Ford	789 Palm Dr., Beverly Hills	F	7/7/77

Az S reláció

4.1. ábra. *Két reláció*

Ezen műveletek relációkra történő alkalmazásakor néhány feltételt kell szabnunk:

1. Az R és S relációk sémája ugyanazt az attribútumhalmazt kell tartalmazza.
2. Mielőtt kiszámolnánk a halmazelméleti egyesítését, metszetét vagy különbségét a sorhalmazoknak, az R és S oszlopait rendezni kell úgy, hogy az attribútumok sorrendje egyforma legyen mindkét reláció esetén.

Néha szeretnénk kiszámolni két olyan reláció egyesítését, metszetét vagy különbségét, amelyek attribútumainak a száma megegyezik, viszont az attribútumok neve különbözik. Ebben az esetben az egyik vagy mindkét reláció sémájának megváltoztatásához használhatjuk a 4.1.8. alfejezetben bemutatott átnevezés operátort, és így ugyanolyan attribútumhalmazt tudunk adni a relációknak.

4.1. példa: Tegyük fel, hogy van két relációnk, az R és az S, amelyek a 3.9. alfejezetben található `FilmSzínész` reláció előfordulásai. Az R és S aktuális előfordulásait a 4.1. ábrán láthatjuk. Az R és S egyesítése, $R \cup S$, a következő reláció:

név	*cím*	*nem*	*születésnap*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88
Harrison Ford	789 Palm Dr., Beverly Hills	F	7/7/77

Figyeljük meg, hogy a Carrie Fisher adatait tartalmazó sor csak egyszer jelenik meg az eredményben, annak ellenére, hogy mindkét relációban szerepelt.

Az $R \cap S$ metszet:

név	*cím*	*nem*	*születésnap*
Carrie Fisher	123 Maple St., Hollywood	N	9/9/99

Most csak a Carrie Fisher adatait tartalmazó sor jelenik meg, mert csak ez a sor szerepel mindkét relációban.

Az $R - S$ különbség:

név	*cím*	*nem*	*születésnap*
Mark Hamill	456 Oak Rd., Brentwood	F	8/8/88

Az R-ben a Fisher és Hamill adatait tartalmazó sorok szerepelnek, ezért ők mindketten szerepelhetnének az $R - S$ különbségben. Viszont a Fisher adatait tartalmazó sor szerepel az S-ben is, ezért nem szerepelhet az $R - S$ különbségben. □

4.1.2. Vetítés

A *vetítés* operátorral a régi R relációból olyan új reláció hozható létre, amelyik csak az R bizonyos oszlopait tartalmazza. A $\pi_{A_1, A_2, \ldots, A_n}(R)$ kifejezés értéke az a reláció, amelyik az R relációnak csak az $A_1, A_2, \ldots, A_n$ attribútumokhoz tartozó oszlopait tartalmazza. Az eredmény sémája az $\{A_1, A_2, \ldots, A_n\}$ attribútumhalmaz, amit mi megegyezés szerint egy rendezett listával jelölünk.

cím	*év*	*hossz*	*színes*	*stúdióNév*	*producerAzon*
Csillagok háborúja	1977	124	igaz	Fox	12345
Rút kiskacsa	1991	104	igaz	Disney	67890
Wayne világa	1992	95	igaz	Paramount	99999

4.2. ábra. *A `Film` reláció*

4.2. példa: Tekintsük a 3.9. alfejezetben megadott sémájú `Film` relációt. A reláció egy előfordulását a 4.2. ábrán láthatjuk. Ezt a relációt a következő kifejezés segítségével vetíthetjük az első három attribútumára:

$\pi_{cím,év,hossz}$(`Film`)

Az eredményül kapott reláció a következő:

cím	*év*	*hossz*
Csillagok háborúja	1977	124
Rút kiskacsa	1991	104
Wayne világa	1992	95

De például levetíthetjük a `Film` relációt a `színes` attribútumra a $\pi_{színes}$(`Film`) kifejezéssel. Ekkor az eredményül kapott reláció csak egyetlen oszlopot tartalmaz:

színes
igaz

Figyeljük meg, hogy az eredményben csak egyetlen sor található, mivel a 4.2. ábrán látható mindhárom sor `színes` attribútumának értéke ugyanaz. □

4.1.3. Kiválasztás

Az R relációra alkalmazott *kiválasztás* operátor olyan új relációt hoz létre, amely az R sorainak egy részhalmazát tartalmazza. Az eredménybe azok a sorok kerülnek, amelyek teljesítenek egy adott, az R attribútumaira megfogalmazott C feltételt. Ezt a műveletet a $\sigma_c(R)$ kifejezéssel jelöljük. Az eredményreláció sémája megegyezik az R sémájával, és megegyezés szerint az attribútumokat ugyanabban a sorrendben tüntetjük fel, mint ahogyan az R relációban használtuk.

A C egy olyan feltételes kifejezés, amilyet megszoktunk a hagyományos programozási nyelveknél. Például az `if` kulcsszót feltételes kifejezés követi mind a C, mind pedig a Pascal programozási nyelvekben. Az egyetlen különbség az, hogy a C feltételben levő operandusok vagy konstansok, vagy az R attribútumai. Alkalmazzuk a C feltételt az R minden egyes t sorára, oly módon, hogy a C-ben előforduló minden egyes A attribútumra behelyettesítjük a t sornak az A attribútumhoz tartozó komponensét. Ha a C feltétel összes attribútumát behelyettesítve a C értéke igaz, akkor a t sor egyike azoknak a soroknak, amelyek megjelennek a $\sigma_c(R)$ eredményében; egyébként a t nincs az eredményben.

4.3. példa: Tekintsük a 4.2. ábrán látható `Film` nevű relációt. Ebben az esetben a $\sigma_{hossz \geq 100}(\texttt{Film})$ kifejezés értéke a következő:

cím	*év*	*hossz*	*színes*	*stúdióNév*	*producerAzon*
Csillagok háborúja	1977	124	igaz	Fox	12345
Rút kiskacsa	1991	104	igaz	Disney	67890

Az első sor kielégíti a *hossz*≥100 feltételt, hiszen ha behelyettesítjük a *hossz* helyébe az első sor megfelelő komponensét, a 124-et, akkor a feltétel így néz ki: 124≥100. Ez utóbbi feltétel igaz, ezért az első sort elfogadjuk. Ugyanilyen indokok magyarázzák, hogy a 4.2. ábra második sora miért kerül be az eredménybe.

A harmadik sor `hossz` komponense 95. Ezért, amikor behelyettesítünk a *hossz* attribútumba, a 95≥100 hamis feltételt kapjuk. Ennélfogva a 4.2. ábra utolsó sora nincs az eredményben. □

4.4. példa: Tegyük fel, hogy a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

relációból azon sorok halmazát szeretnénk megkapni, amelyek a Fox stúdió legalább 100 perces filmjeit tartalmazzák. Ezeket a sorokat egy olyan bonyolultabb feltétel segítségével kaphatjuk meg, amelyet két részfeltétel AND összekapcsolásával nyerünk. A keresett kifejezés:

$\sigma_{hossz \geq 100 \text{ AND } stúdióNév = \text{'Fox'}}(\texttt{Film})$

Az eredmény:

cím	*év*	*hossz*	*színes*	*stúdióNév*	*producerAzon*
Csillagok háborúja	1977	124	igaz	Fox	12345

Az eredményrelációnak egyetlenegy sora van. □

4.1.4. Descartes-szorzat

Két halmaz, R és S *Descartes-szorzata* (vagy egyszerűen csak *szorzata*) azon párok halmaza, amelyeknek első eleme az R tetszőleges eleme, a második pedig az S egy eleme. A szorzat jelölése $R \times S$. Amikor R és S relációk, a szorzat – lényegéből adódóan – szintén reláció. Azonban, mivel az R és S elemei sorok, mégpedig általában egynél több komponensből álló sorok, ezért az R egy sorának párosítása az S egy sorával olyan hosszabb sort eredményez, amelyben az alkotó sorok mindegyik komponense megjelenik. Az R komponensei megelőzik sorrendben az S komponenseit.

Az eredményreláció sémája az R és S sémájának egyesítése. Azonban előfordulhat, hogy az R és S relációknak vannak közös attribútumai. Ekkor minden azonos nevű attribútumot tartalmazó pár esetén legalább az egyiknek új nevet kell adni. Egy olyan A attribútum egyértelművé tételéhez, amelyik mind az R, mind az S sémájában szerepel, a nevek megkülönböztetésére az $R.A$, illetve $S.A$ jelöléseket használjuk, attól függően, hogy az R reláció A attribútumáról vagy az S reláció A attribútumáról van szó.

4.5. példa: A tömörség kedvéért használjunk egy absztrakt példát a szorzás művelet szemléltetésére. Tekintsük a 4.3. ábrán látható R és S relációkat, az ábrán megadott sémákkal és sorokkal. Ekkor, amint az ábrán is látható, az $R \times S$ szorzat hat sorból áll. Figyeljük meg, hogyan párosítottuk az R két sorának mindegyikét az S három sorának mindegyikével. Mivel a B mindkét sémának attribútuma, ezért az $R.B$ és $S.B$ jelölést használtuk az $R \times S$ sémájában. A többi attribútum egyértelmű, és ezeknek a nevei változatlanok az eredmény sémájában. □

A	*B*
1	2
3	4

Az R reláció

B	*C*	*D*
2	5	6
4	7	8
9	10	11

Az S reláció

A	*R.B*	*S.B*	*C*	*D*
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

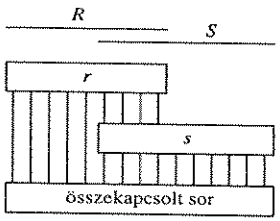
Az $R \times S$ eredménye

4.3. ábra. *Két reláció és a Descartes-szorzatuk*

4.1.5. Természetes összekapcsolás

Két reláció szorzásánál jóval gyakrabban van szükségünk arra, hogy *összekapcsoljunk* relációkat oly módon, hogy csak azokat a sorokat párosítsuk, amelyek valamilyen módon összeillenek. Az összeillesztés legegyszerűbb módja két reláció *természetes összekapcsolása,* amelynek jelölése $R \bowtie S$, és amelyben R-nek és S-nek csak azokat a sorait párosítjuk össze, amelyek értékei megegyeznek az R és S sémájának összes közös attribútumán. Még pontosabban: legyenek $A_1, A_2, ..., A_n$ azok az attribútumok, amelyek megtalálhatók mind a R, mind a S sémájában. Ekkor az R egy r sorának és S egy s sorának párosítása akkor és csak akkor sikeres, ha az r és s megfelelő értékei megegyeznek az összes $A_1, A_2, ..., A_n$ attribútumon.

Ha az r és s sorok párosítása az $R \bowtie S$ összekapcsolásban sikeres, akkor a párosítás eredménye egy olyan, *összekapcsolt sornak* nevezett sor, amelyben az R és S sémájának egyesítésében szereplő összes attribútumhoz egyetlen komponens tartozik. Az összekapcsolt sor megegyezik az r sorral az R összes attribútumán, és megegyezik az s sorral S összes attribútumán. Mivel az r és s összekapcsolása sikeres volt, így tudjuk, hogy az r és s megegyezik azokon az attribútumokon, amelyek mind az R, mind pedig az S sémájában szerepelnek. Ezért az összekapcsolt sor is megegyezik mind az r, mind az s sorokkal a mindkét sémában szereplő attribútumokon. Az összekapcsolt sorok szerkezetét szemlélteti a 4.4. ábra.



4.4. ábra. *Sorok összekapcsolása*

Megjegyezzük, hogy ez az összekapcsolási művelet ugyanaz, mint amit a 3.7.6. alfejezetben használtunk olyan relációk újbóli összekapcsolására, amelyeket úgy kaptunk, hogy levetítettünk egy relációt az attribútumainak két részhalmazára. Akkor a motiváció az volt, hogy megmutassuk, a BCNF dekompozíciónak van értelme. A 4.1.7. alfejezetben a természetes összekapcsolás egy másik felhasználási módját látjuk majd: két reláció összekapcsolását abból a célból, hogy olyan lekérdezést írhassunk, amelyik mindkettőnek az attribútumaira vonatkozik.

4.6. példa: A 4.3. ábrán látható R és S relációk természetes összekapcsolásának eredménye:

A	B	C	D
1	2	5	6
3	4	7	8

Az R és S egyetlen közös attribútuma a B. Ennek következtében a sorok sikeres párosításához elegendő, hogy a sorok B attribútumán levő értékek megegyezzenek. Ekkor az eredménysornak lesz egy komponense az A attribútumhoz (az R-ből), a B attribútumhoz (az R-ből vagy S-ből), a C attribútumhoz (az S-ből), és a D attribútumhoz (az S-ből).

Ebben a példában az R első sora csak az S első sorával párosítható sikeresen; mindkét sor B attribútumának értéke 2. Ennek a párosításnak az eredménye az első sor: (1, 2, 5, 6). Az R második sora csak az S második sorával párosítható sikeresen, és ez a párosítás a (3, 4, 7, 8) sort eredményezi. Megfigyelhetjük, hogy az S harmadik sora nem párosítható az R egyetlen sorával sem, ezért nincs is semmi hatása az $R \bowtie S$ eredményére nézve. Az olyan sort, amelyet nem lehet sikeresen párosítani az összekapcsolásban szereplő másik reláció egyetlen sorával sem, *lógó sornak* is nevezzük. □

4.7. példa: Az előző példa nem szemlélteti a természetes összekapcsolással felmerülő összes lehetőséget. Így például nem volt olyan sor, amelyik egynél több sorral is párosítható lett volna, és a két reláció sémájának csak egyetlen egy közös attribútuma volt. A 4.5. ábrán látható U és V relációk sémájának két közös attribútuma van, a B és a C. Ráadásul az ábrán szereplő egyik előfordulásnak van egy olyan sora, amelyik több sorral is összekapcsolható.

A sikeresen párosítható sorok B és C komponenseinek is egyeznie kell. Ekképpen az U első sora sikeresen párosítható a V első két sorával, viszont az U második és harmadik sora csak a V harmadik sorával párosítható sikeresen. A négy párosítás eredménye a 4.5 ábrán látható. □

A	B	C
1	2	3
6	7	8
9	7	8

Az U reláció

B	C	D
2	3	4
2	3	5
7	8	10

A V reláció

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Az $U \bowtie V$ eredménye

4.5. ábra. *Relációk természetes összekapcsolása*

4.1.6. Théta-összekapcsolás

A természetes összekapcsolás előírja, hogy egyetlen speciális feltétel szerint párosítsuk a sorokat. Bár a relációk összekapcsolásának leggyakoribb kiindulási pontja a közös attribútumokban levő értékek egyenlővé tétele, azért néha szükség lehet két reláció sorainak más szempontból történő párosítására. Ezért bevezetjük a *théta-összekapcsolás* műveletet: a „théta" egy tetszőleges feltételre utal, amit mi θ helyett inkább C-vel jelölünk.

R és S relációknak C feltételre vonatkozó théta-összekapcsolásának jelölése $R \bowtie_C S$. Ennek a műveletnek az eredményét a következő módon kapjuk:

1. Kiszámoljuk R és S szorzatát.

2. Kiválasztjuk a szorzatból azokat a sorokat, amelyek eleget tesznek a C feltételnek.

Éppúgy, mint a szorzat műveletnél, az eredmény sémája itt is az R és S sémáinak az egyesítése. Ha szükséges megjelölni, hogy az attribútumok melyik sémából származnak, akkor használjuk az attribútumokhoz az „R.", illetve „S." előtagokat.

4.8. példa: Tekintsük az $U \bowtie_{A<D} V$ műveletet, ahol U és V a 4.5. ábrán látható két reláció. Figyelembe kell vennünk a relációk sorainak mind a kilenc lehetséges párosítását, és meg kell néznünk, hogy vajon az U sorának A komponense kisebb-e mint a V sorának D komponense. Az U első sorában az A komponens értéke 1, és ez a sor sikeresen párosítható a V összes sorával. Az U második és harmadik sorában az A komponens értéke 6, illetve 9, ezért ezek csak a V utolsó sorával párosíthatók sikeresen. Az öt sikeres párosításból eredően az eredménynek öt sora lesz. Az eredményreláció a 4.6. ábrán látható. □

Figyeljük meg, hogy a 4.6. ábrán látható eredmény sémájában mind a hat attribútum szerepel. A B és C attribútumok a megfelelő előtagokkal szerepelnek, megkülönböztetendő, hogy az U vagy a V attribútumai. Tehát a théta-összekapcsolás különbözik a természetes összekapcsolástól, hiszen ez utóbbiban a közös attribútumok csak egyszer szerepelnek. Természetesen ennek csak a természetes összekapcsolásnál van értelme, hiszen ott csak azokat a sorokat párosítjuk, amelyek megegyeznek a közös attribútumokon. A théta-összekapcsolás esetében az összehasonlító operátor más is lehet, mint az =, éppen ezért nem biztos, hogy az összehasonlított attribútumok megegyeznek az eredményben.

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

4.6. ábra. *Az* $U \bowtie_{A<D} V$ *eredménye*

4.9. példa: Vegyük a korábbi U, V relációk théta-összekapcsolását egy összetettebb feltétellel:

$$U \bowtie_{A<D \text{ AND } U.B \neq V.B} V$$

A sikeres párosításhoz már nem elég, hogy az U sorának A komponense kisebb legyen mint a V sorának D komponense, hanem annak is teljesülni kell, hogy a két sor értéke legyen különböző a B attribútumokon. Az egyetlen sor, ami teljesíti ezt a feltételt, a következő:

A	$U.B$	$U.C$	$V.B$	$V.C$	D
1	2	3	7	8	10

Tehát a théta-összekapcsolás eredménye a fenti egyetlenegy sort tartalmazó reláció. □

4.1.7. Lekérdezések megfogalmazása műveletek segítségével

Ha lekérdezéseket csak a műveletek egy vagy két relációra történő alkalmazásával fogalmazhatnánk meg, akkor a relációs algebra nem lenne annyira hasznos, mint amilyen hasznos valójában. Azonban a relációs algebra is, mint minden algebra, lehetőséget ad arra, hogy tetszőleges bonyolultságú kifejezéseket képezzünk. Ekkor az operátorokat vagy az adott relációkra alkalmazzuk, vagy olyan relációkra, amelyek más operátorok relációkra történő alkalmazásának eredményei.

Relációs algebrai kifejezések megadásakor, amikor a műveleteket részkifejezésekre alkalmazzuk, használhatunk zárójeleket az operandusok csoportosításának egyértelművé tételére. Lehetséges a kifejezések kifejezésfával történő megadása is: ez utóbbit nekünk könnyebb olvasni, de géppel kevésbé olvasható.

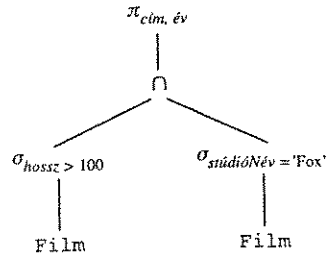
4.10. példa: Tekintsük a 3.32. példa dekompozíció utáni `Film` relációját. Tegyük fel, hogy a következő kérdésre szeretnénk megkapni a választ: „Melyek a Fox stúdióban készült, legalább 100 perc hosszúságú filmek, és ezek mikor készültek?" A kérdés megválaszolására az egyik lehetőség:

1. Kiválasztjuk a `Film` relációból azokat a sorokat, amelyekre a *hossz* ≥ 100.

2. Kiválasztjuk a `Film` relációból azokat a sorokat, amelyekre a *stúdióNév* = `'Fox'`.

3. Kiszámoljuk az (1) és (2) metszetét.

4. A (3) lépésben megkapott relációt levetítjük a `cím` és `év` attribútumokra.



4.7. ábra. *Egy relációs algebrai kifejezés kifejezésfája*

Ekvivalens kifejezések és a lekérdezések optimalizálása

Minden adatbázisrendszernek van egy lekérdezés-válaszoló rendszere, ahol a lekérdezés legtöbbször olyan nyelvre épül, amely kifejező erejét tekintve hasonló a relációs algebrához. Éppen ezért a felhasználó által megfogalmazott lekérdezéshez létezhet több *ekvivalens kifejezés* (az ekvivalens kifejezések olyan kifejezések, amelyeket ha ugyanazokon a relációkon értékelünk ki, ugyanazt az eredményt adják). Ezek között vannak olyanok, amelyek gyorsabban kiértékelhetők. Az 1.2.3. alfejezetben tárgyalt lekérdezés-válaszolónak egyik fontos feladata éppen az, hogy egy relációs algebrai kifejezést olyan ekvivalens kifejezéssel helyettesítsen, amely hatékonyabban értékelhetők ki.

A 4.7. ábrán a fenti lépéseknek megfelelő kifejezésfát láthatjuk. A két kiválasztást tartalmazó csúcs az (1) és (2) lépéseknek felel meg. A metszetet tartalmazó csúcs a (3) lépésnek felel meg, a vetítést tartalmazó csúcs pedig a (4) lépés.

Ugyanezt a kifejezést felírhatjuk zárójelek használatával a hagyományos lineáris jelöléssel is. A következő formula ugyanazt a kifejezést jelenti.

$$\pi_{cím,év}(\sigma_{hossz \geq 100}(\texttt{Film}) \cap \sigma_{stúdióNév = \texttt{'Fox'}}(\texttt{Film}))$$

Egyébként gyakori, hogy több relációs algebrai kifejezésnek is ugyanaz az eredménye. Például a fenti lekérdezés felírható egyetlen kiválasztás használatával, ha a metszetet az AND operátorral helyettesítjük. A következő kifejezés

$$\pi_{cím,év}(\sigma_{hossz \geq 100 \text{ AND } stúdióNév=\texttt{'Fox'}}(\texttt{Film}))$$

ekvivalens az előző kifejezéssel. □

4.11. példa: A természetes összekapcsolás egyik felhasználási módja olyan relációknak az újraösszekapcsolása, amelyek egyetlen reláció BCNF dekompozíciójából jöttek létre. A 3.32. példa dekompozíció utáni relációi:[1]

`Film1` sémája{ `cím, év, hossz, szalagFajta, stúdióNév`}
`Film2` sémája{ `cím, év, színészNév`}

Adjuk meg azt a kifejezést, amelyik a következő kérdésre felel: „Kik azok a színészek, akik szerepelnek olyan filmekben, amelyek legalább 100 perc hosszúak?" Ez a

[1] Ne feledjük, hogy annak a `Film` relációnak a sémája, amit ebben a példában használtunk, különbözik a 3.9. alfejezetben bevezetett, és a 4.2., 4.3. és 4.4. példákban használt `Film` reláció sémájától.

kérdés a `Film2` `színészNév` attribútumára és a `Film1` `hossz` attribútumára vonatkozik. A két reláció összekapcsolásával teremthetjük meg a két attribútum közötti kapcsolatot. A természetes összekapcsolás csak azokat a sorokat párosítja össze sikeresen, amelyeknél a `cím` és az `év` megegyezik, vagyis ugyanarra a filmre vonatkoznak. Tehát, a `Film1` ⋈ `Film2` egy olyan relációs algebrai kifejezés, amelyik a 3.32. példa `Film` relációját adja meg. Ennek a relációnak a sémája tartalmazza mind a hat attribútumot, ezért abban az esetben, ha egy filmnek több szereplője is van, több sort is tartalmaz ugyanarról a filmről, tehát a reláció nem BCNF.

A `Film1` és `Film2` összekapcsolása után ki kell választanunk azokat a filmeket, amelyek legalább 100 perc hosszúak. Ezután levetítjük őket a keresett `színészNév` attribútumra. A következő kifejezés

$$\pi_{színészNév}(\sigma_{hossz \geq 100}(\texttt{Film1} \bowtie \texttt{Film2}))$$

éppen a kívánt lekérdezés relációs algebrai megfelelője. □

4.1.8. Átnevezés

Relációs algebrai műveletek alkalmazásával újabb relációkat kapunk. Ahhoz, hogy ezen relációk attribútumainak nevét szabályozni tudjuk, gyakran szükséges egy olyan operátor használata, amelyik kifejezetten átnevezi a relációkat. Egy R reláció átnevezéséhez a $\rho_{S(A_1, A_2, ..., A_n)}(R)$ operátort használjuk. Az eredményreláció neve S, sorai megegyeznek az R soraival és attribútumainak neve balról jobbra: $A_1, A_2, ..., A_n$. Ha az attribútumok neveit meg akarjuk őrizni és csak a reláció nevét szeretnénk megváltoztatni, akkor egyszerűen csak annyit írunk: $\rho_S(R)$.

4.12. példa: A 4.5. példában kiszámoltuk a 4.3. ábrán látható két reláció, R és S szorzatát. Megállapodtunk abban is, hogyha egy attribútum mindkét operandusban szerepel, akkor ezeket az attribútumokat átnevezzük oly módon, hogy az új névben az attribútumokhoz tartozó relációk neve is jelenjen meg, előtagként. A 4.8. ábrán szintén ez a két reláció, az R és az S látható.

Tegyük fel azonban, hogy a B attribútum két előfordulását nem $R.B$, illetve $S.B$ névvel szeretnénk illetni, hanem az R reláció B attribútumának a nevét szeretnénk megőrizni, az S reláció B attribútumát pedig X névvel szeretnénk illetni. E célból átnevezhetjük az S attribútumait úgy, hogy az elsőnek a neve legyen X. A $\rho_{S(X,C,D)}(S)$ kifejezés eredménye pontosan egy olyan S nevű reláció, amelyik ugyanolyan, mint a 4.3. ábrán látható S reláció, csak az első attribútumának a neve nem B, hanem X.

Amikor megszorozzuk az R relációt ezzel az új relációval, már nem lesz gond az egyforma attribútumnevekkel, további átnevezésekre így nincs szükség. Vagyis az $R \times \rho_{S(X,C,D)}(S)$ kifejezés eredménye a 4.3. ábrán látható $R \times S$ reláció, kivéve, hogy az öt attribútum neve balról jobbra: A, B, X, C és D. A 4.8. ábrán ez a reláció látható.

Egy másik lehetőség az, hogy amint a 4.5. példában is tettük, kiszámoljuk a szorzatot átnevezés nélkül, ezt követően pedig átnevezzük eredményt. A $\rho_{RS(A,B,X,C,D)}(R \times S)$

A	B
1	2
3	4

Az R reláció

B	C	D
2	5	6
4	7	8
9	10	11

Az S reláció

A	B	X	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Az $R \times \rho_{S(X,C,D)}(S)$ eredménye

4.8. ábra. *Két reláció és a szorzatuk*

kifejezés eredményének ugyanazok az attribútumai, mint a 4.8. ábrán látható relációnak, viszont a neve *RS*, míg a 4.8. ábrán látható relációnak nincs neve. □

4.1.9. Műveletek függetlensége

A 4.1. alfejezetben bemutatott műveletek között vannak olyanok, amelyek kifejezhetők más relációs algebrai műveletek segítségével. Például a metszet kifejezhető halmazok különbségével:

$$R \cap S = R - (R - S)$$

Vagyis, ha R és S két, egyforma sémával rendelkező reláció, akkor R és S metszete kiszámolható úgy, hogy először kiszámoljuk azt a T relációt, amelyben mindazok a sorok benne vannak, amelyek R-ben is benne vannak, de S-nek nem sorai. Azaz kivonjuk R-ből az S-t. Ezután kivonjuk az R-ből a T-t, ezáltal R-nek azokat a sorait kapjuk, amelyek S-ben is benne vannak.

A két összekapcsolás szintén kifejezhető más műveletek segítségével. A théta-összekapcsolás szorzás és kiválasztás segítségével fejezhető ki:

$$R \bowtie_C S = \sigma_C(R \times S)$$

R és S természetes összekapcsolásának kifejezéséhez először vegyük az $R \times S$ szorzatot. Ezután alkalmazzuk a kiválasztás operátort a következő alakú C feltétellel:

$$R.A_1 = S.A_1 \text{ AND } R.A_2 = S.A_2 \text{ AND } \ldots \text{ AND } R.A_n = S.A_n$$

ahol az $A_1, A_2, \ldots, A_n$ olyan attribútumok, amelyek mind az R, mind az S sémájában szerepelnek. Utolsó lépésként pedig mindegyik közös attribútumból csak egy példányt őrizzünk meg. Legyen az L egy olyan attribútumlista, amelyben szerepel az R összes attribútuma és emellett az S-ből mindazok az attribútumok, amelyek nincsenek benne az R sémájában. Ekkor:

$$R \bowtie S = \pi_L(\sigma_C(R \times S))$$

4.13. példa: A 4.5. ábrán látható U és V relációk természetes összekapcsolása felírható szorzás, kiválasztás és vetítés segítségével, a következőképpen:

$$\pi_{A,U.B,U.C,D}(\sigma_{U.B = V.B \text{ AND } U.C = V.C}(U \times V))$$

Vagyis vesszük az $U \times V$ szorzatot. Ezután kiválasztjuk azokat a sorokat, amelyekben az egyforma nevű – jelen esetben a B és a C – attribútumokhoz tartozó értékek egyenlők. Végül pedig egy vetítést végzünk az összes attribútumra, kivéve egy B és egy C attribútumot: választásunk azon V attribútumok elhagyására esett, amelyek benne vannak az U sémájában.

Egy másik példa a 4.9. példában kiszámolt théta-összekapcsolás átírása:

$$\sigma_{A < D \text{ AND } U.B \neq V.B}(U \times V)$$

Vagyis, kiszámoljuk az U és V szorzatát, és utána alkalmazunk egy kiválasztást a théta-összekapcsolásban szereplő feltétel szerint. □

Ebben az alfejezetben megemlített redundanciák csak a bevezetett műveletek közötti „redundanciák". A fennmaradó hat operátor – egyesítés, különbség, kiválasztás, vetítés, szorzat és átnevezés – független halmazt alkot, egyik sem fejezhető ki a másik öt segítségével.

4.1.10. Feladatok

4.1.1. feladat: Ebben a feladatban bevezetjük az egyik állandó példánk relációs adatbázissémáját és megadunk hozzá néhány mintaadatot. Az adatbázis sémája négy relációs sémából áll, amelyek a következők:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez,
       képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A `Termék` reláció tartalmazza a különböző termékek gyártóját, modellszámát és típusát (PC, laptop vagy nyomtató). Az egyszerűség kedvéért feltételezzük, hogy a modellszámok egyediek, minden gyártó és termék esetén; ez a feltételezés nem túl valószerű, hiszen egy igazi adatbázisban a gyártó kódja része lenne a modellszámnak. A `PC` reláció minden PC modellszámához tartalmazza a processzor sebességét megahertzben, a RAM méretét megabájtban, a merevlemez méretét gigabájtban, a CD olvasó sebességét (pl. 4×) és az árát. A `Laptop` reláció hasonló, csak a CD sebessége helyett a képernyő méretét tartalmazza, hüvelykben mérve. A `Nyomtató` reláció minden nyomtatóhoz tartalmazza, hogy színes-e a nyomtató vagy sem (igaz vagy hamis), típusát (lézer, tintasugaras vagy mátrix) és az árát.

gyártó	*modell*	*típus*
A	1001	pc
A	1002	pc
A	1003	pc
B	1004	pc
B	1006	pc
B	3002	nyomtató
B	3004	nyomtató
C	1005	pc
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc

gyártó	*modell*	*típus*
D	2001	laptop
D	2002	laptop
D	2003	laptop
D	3001	nyomtató
D	3003	nyomtató
E	2004	laptop
E	2008	laptop
F	2005	laptop
G	2006	laptop
G	2007	laptop
H	3005	nyomtató
I	3006	nyomtató

4.9. ábra. *A* `Termék` *reláció mintaadatai*

A `Termék` reláció néhány mintaadata a 4.9. ábrán látható. A másik három reláció mintaadatai a 4.10. ábrán láthatók. A gyártók, illetve a modellszámok fiktív adatok, de a többi adat az 1996. év piacát tükrözi.

Írjuk fel a következő lekérdezésekhez tartozó relációs algebrai kifejezéseket. A 4.9. és 4.10. ábrán látható adatok alapján számoljuk ki a lekérdezések eredményeit. A válaszul felírt kifejezéseknek más adatokra is a helyes választ kell megadniuk, nem csak az ábrán látható adatokra. *Útmutatás:* A nehezebb feladatok esetén ajánlatos közbeeső relációkat használni, és ezeket írni a végeredménybe, majd visszahelyettesíteni őket a megfelelő kifejezésekkel.

* a) Melyek azok a PC modellek, amelyek sebessége legalább 150?

b) Mely gyártók készítenek legalább egy gigabájt méretű merevlemezzel rendelkező laptopot?

c) Adjuk meg a *B* gyártó által gyártott összes termék modellszámát és árát, típustól függetlenül.

d) Adjuk meg valamennyi színes lézernyomtató modellszámát.

e) Melyek azok a gyártók, akik laptopot árulnak, PC-t viszont nem?

*! f) Melyek azok a merevlemezméretek, amelyek legalább két PC-ben megtalálhatók?

! g) Adjuk meg azokat a PC párokat, amelyek ugyanolyan gyorsak és a memóriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha már szerepel az (*i*, *j*), akkor a (*j*, *i*) ne jelenjen meg.

modell	*sebesség*	*memória*	*merevlemez*	*cd*	*ár*
1001	133	16	1,6	6×	1595
1002	120	16	1,6	6×	1399
1003	166	24	2,5	6×	1899
1004	166	32	2,5	8×	1999
1005	166	16	2,0	8×	1999
1006	200	32	3,1	8×	2099
1007	200	32	3,2	8×	2349
1008	180	32	2,0	8×	3699
1009	200	32	2,5	8×	2599
1010	160	16	1,2	8×	1495

a) A `PC` reláció mintaadatai

modell	*sebesség*	*memória*	*merevlemez*	*képernyő*	*ár*
2001	100	20	1,10	9,5	1999
2002	117	12	0,75	11,3	2499
2003	117	32	1,00	10,4	3599
2004	133	16	1,10	11,2	3499
2005	133	16	1,00	11,3	2599
2006	120	8	0,81	12,1	1999
2007	150	16	1,35	12,1	4799
2008	120	16	1,10	12,1	2099

b) A `Laptop` reláció mintaadatai

modell	*színes*	*típus*	*ár*
3001	igaz	tintasugaras	275
3002	igaz	tintasugaras	269
3003	hamis	lézer	829
3004	hamis	lézer	879
3005	hamis	tintasugaras	180
3006	igaz	mátrix	470

c) A `Nyomtató` reláció mintaadatai

4.10. ábra. *A 4.11. feladat relációinak mintaadatai*

*!! h) Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 133 megahertzen működő számítógépet (PC-t vagy laptopot)?

!! i) Melyik gyártó gyártja a leggyorsabb számítógépet (PC-t vagy laptopot)?

!! j) Melyik gyártó gyárt legalább három, különböző sebességű PC-t?

!! k) Melyek azok a gyártók, akik pontosan három típusú PC-t forgalmaznak?

4.1.2. feladat: Az előző feladatban megkapott kifejezések mindegyikéhez rajzoljuk fel a megfelelő kifejezésfát.

hajóosztály	*típus*	*ország*	*ágyúkSzáma*	*kaliber*	*vízkiszorítás*
Bismarck	h	Németország	8	15	42000
Iowa	h	USA	9	16	46000
Kongo	c	Japán	8	14	32000
North Carolina	h	USA	9	16	37000
Renown	c	Nagy-Britannia	6	15	32000
Revenge	h	Nagy-Britannia	8	15	29000
Tennessee	h	USA	12	14	32000
Yamato	h	Japán	9	18	65000

a) A Hajóosztály reláció mintaadatai

név	*dátum*
North Atlantic	41/5/24-27
Guadalcanal	42/11/15
North Cape	43/12/26
Surigao Strait	44/10/25

b) A Csaták reláció mintaadatai

hajó	*csata*	*állapot*
Bismarck	North Atlantic	elsüllyedt
California	Surigao Strait	ép
Duke of York	North Cape	ép
Fuso	Surigao Strait	elsüllyedt
Hood	North Atlantic	elsüllyedt
King George V	North Atlantic	ép
Kirishima	Guadalcanal	elsüllyedt
Prince of Wales	North Atlantic	sérült
Rodney	North Atlantic	ép
Scharnhorst	North Cape	elsüllyedt
South Dakota	Guadalcanal	sérült
Tennessee	Surigao Strait	ép
Washington	Guadalcanal	ép
West Virginia	Surigao Strait	ép
Yamashiro	Surigao Strait	elsüllyedt

c) A Kimenetelek reláció mintaadatai

4.11. ábra. *A 4.1.3. feladat adatai*

4.1.3. feladat: Ebben a feladatban egy újabb állandó példát mutatunk be. Ez a példa a II. világháború csatahajóival foglalkozik és a következő sémájú relációkat tartalmazza:

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, állapot)
```


név	*hajóosztály*	*felavatva*
California	Tennessee	1921
Haruna	Kongo	1915
Hiei	Kongo	1914
Iowa	Iowa	1943
Kirishima	Kongo	1915
Kongo	Kongo	1913
Missouri	Iowa	1944
Musashi	Yamato	1942
New Jersey	Iowa	1943
North Carolina	North Carolina	1941
Ramillies	Revenge	1917
Renown	Renown	1916
Repulse	Renown	1916
Resolution	Revenge	1916
Revenge	Revenge	1916
Royal Oak	Revenge	1916
Royal Sovereign	Revenge	1916
Tennessee	Tennessee	1920
Washington	North Carolina	1941
Wisconsin	Iowa	1944
Yamato	Yamato	1941

4.12. ábra. *A Hajók reláció mintaadatai*

Az egyforma típusú hajókat „hajóosztályokba" soroljuk, és egy hajóosztályt általában az első hajójáról nevezünk el. A Hajóosztályok reláció tartalmazza minden egyes osztálynak a nevét, a típusát (a csatahajókat h-val, a cirkálóhajókat c-vel jelöljük), a hajógyártó ország nevét, a főágyúk számát, a főágyúk kaliberét (a kaliber az ágyúcső átmérője hüvelykben) és a hajó vízkiszorítását (tonnában mérve). A Hajók reláció tartalmazza a hajók nevét, annak a hajóosztálynak a nevét, amelybe az adott hajó tartozik és a felavatás évét. A Csaták reláció megadja a tengeri csaták nevét és dátumát, míg a Kimenetelek reláció megadja a csatákban részt vevő hajók nevét és a csata utáni állapotukat (elsüllyedt, sérült vagy ép).

A 4.11. és a 4.12. ábra a fenti négy reláció mintaadatait tartalmazza.[2] Figyeljük meg, hogy a 4.1.1. feladat adataival ellentétben, ezek az adatok tartalmaznak „lógó sorokat". Ilyenek például azok a hajók, amelyeket a Kimenetelek reláció tartalmaz, de a Hajók reláció nem.

Írjuk fel a következő kérdésekhez tartozó relációs algebrai kifejezéseket. A 4.11. és 4.12. ábrán látható adatok alapján számoljuk ki a lekérdezések eredményeit. A válaszul felírt kifejezéseknek más adatokra is a helyes választ kell megadniuk, nem csak az ábrán látható adatokra.

[2] Forrás: J. N. Westwood, *Fighting Ships of World War II*, Follett Publishing, Chicago, 1975 és R. C. Stern, *US Battleships in Action*, Squadron/Signal Publications, Carrollton, TX, 1980.

a) Adjuk meg azokat a hajóosztályokat a gyártó országok nevével együtt, amelyeknek az ágyúi legalább 16-os kaliberűek.

b) Melyek azok a hajók, amelyeket 1921-ben avattak fel?

c) Adjuk meg a North Atlantic-csatában elsüllyedt hajók nevét.

d) Az 1921-es washingtoni egyezmény betiltotta a 35 000 tonnánál súlyosabb hajókat. Adjuk meg azokat a hajókat, amelyek megszegték az egyezményt.

e) Adjuk meg a Guadalcanal csatában részt vett hajók nevét, vízkiszorítását és ágyúinak a számát.

f) Adjuk meg az adatbázisban szereplő összes hadihajó nevét. (Ne feledjük, hogy a `Hajók` relációban nem szerepel az összes hajó!)

! g) Adjuk meg azokat az osztályokat, amelyekbe csak egyetlenegy hajó tartozik.

! h) Melyek azok az országok, amelyeknek csatahajóik is és cirkálóhajóik is voltak?

! i) Adjuk meg azokat a hajókat, amelyek „újjáéledtek", azaz egyszer már megsérültek egy csatában, de egy későbbi csatában újra harcoltak.

4.1.4. feladat: Az előző feladatban megkapott kifejezések mindegyikéhez rajzoljuk fel a megfelelő kifejezésfát.

* **4.1.5. feladat:** Adott az $R \bowtie S$ természetes összekapcsolás és az $R \bowtie_C S$ théta-összekapcsolás. A C feltétel az összes olyan A attribútumra, amely az R-ben és S-ben is szerepel, tartalmazza az $R.A = S.A$ egyenlőséget. Mi a különbség a két összekapcsolás között?

! **4.1.6. feladat:** Egy relációkon értelmezett operátor akkor *monoton*, hogyha bármelyik argumentumrelációhoz egy újabb sort hozzáveszünk, az eredmény tartalmazza az összes olyan sort, amit addig tartalmazott és esetleg újabb sorokat is. Ebben a fejezetben felsorolt operátorok közül melyek monotonak? Mindegyik operátorra indokoljuk, hogy miért monoton, illetve ha nem monoton, adjunk ellenpéldát.

! **4.1.7. feladat:** Tegyük fel, hogy az R relációnak n, az S relációnak pedig m sora van. Adjuk meg a következő kifejezések eredményeiben keletkezhető sorok maximális és minimális számát.

* a) $R \cup S$

b) $R \bowtie S$

c) $\sigma_C(R) \times S$, tetszőleges C feltétel esetén.

d) $\pi_L(R) - S$, tetszőleges L attribútumlista esetén.

! **4.1.8. feladat:** Az R és S relációk félig-összekapcsolása, $R \ltimes S$, az R azon sorainak halmaza, amely sorok megegyeznek az S legalább egy sorával az R és S összes közös attribútumán. Adjunk meg olyan három különböző relációs algebrai kifejezést, amelyek ekvivalensek az $R \ltimes S$ kifejezéssel.

! **4.1.9. feladat:** Az R reláció sémája

$$(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

és az S reláció sémája $(B_1, B_2, \ldots, B_m)$, azaz S összes attribútuma benne van az R attribútumainak halmazában. Az R és S *hányadosa*, $R \div S$, megadja azon $(A_1, A_2, \ldots, A_n)$ attribútumú t sorok halmazát, amelyekre igaz, hogy az S reláció minden s sorára a ts sor benne van az R relációban. Írjuk fel az eddig bemutatott operátorok segítségével azt a relációs algebrai kifejezést, amely ekvivalens az $R \div S$ kifejezéssel.

4.2. Relációkon értelmezett logika

A relációk lekérdezésének létezik másik megközelítése is, olyan, amelyik az algebra helyett a logikán alapul. Érdekes módon ez a két megközelítés (ez az algebrai és ez a logikai) kifejezőerő szempontjából a lekérdezések ugyanazon osztályához vezet.* Ebben az alfejezetben egy új lekérdezőnyelvet vezetünk be, amelynek a neve *Datalog*. (A név az angol „database logic" kifejezésből ered.) A Datalog if-then (ha-akkor) szabályokból áll. A szabályok segítségével olyan elképzelést fogalmazunk meg, hogy megadott relációk sorainak bizonyos kombinációjából következtethetünk valamely más reláció sorára, vagy egy adott kérdés válaszára.

4.2.1. Predikátumok és atomok

Datalogban a relációkat *predikátumokkal* reprezentáljuk. Minden predikátumnak rögzített számú argumentuma van. Az argumentumaival együtt felírt predikátumot *atomnak* nevezzük. Az atomok szintaxisa olyan, mint a hagyományos programozási nyelvekben a függvényhívás szintaxisa. Például: a $P(x_1, x_2, \ldots, x_n)$ atom az $x_1, x_2, \ldots, x_n$ argumentumú P predikátumból áll.

Lényegében egy predikátum egy logikai visszatérési értékű függvény neve. Ha R egy n attribútumú reláció, akkor a relációnak megfelelő predikátumot szintén R-nek

* Szerkesztői megjegyzés: itt határozottan két konkrét megközelítésről van szó. Az állítás általában nem igaz, lásd majd a rekurziónál.

nevezzük. Az $R(a_1, a_2, \ldots, a_n)$ atom értéke IGAZ, ha az $(a_1, a_2, \ldots, a_n)$ sor az R egy sora, egyébként az atom értéke HAMIS.

4.14. példa: Tekintsük a 4.3. ábra R relációját:

A	B
1	2
3	4

Ebben az esetben az $R(1, 2)$ és $R(3, 4)$ igaz, és minden más x és y értékre az $R(x, y)$ hamis. □

Egy predikátum argumentumai lehetnek konstansok és változók egyaránt. Ha egy atom argumentumai között szerepel egy vagy több változó, akkor az egy olyan logikai függvény, amely a változók értékeitől függően IGAZ vagy HAMIS értéket ad vissza.

4.15. példa: Ha az R a 4.14. példa predikátuma, akkor az $R(x, y)$ az a függvény, amelyik tetszőleges x és y értékekre megadja, hogy az (x, y) sor benne van-e az R relációban. Ha az R relációnak a 4.14. példában megadott megvalósítását tekintjük, akkor az $R(x, y)$ a következő két esetben ad vissza IGAZ értéket:

1. $x = 1$ és $y = 2$, vagy

2. $x = 3$ és $y = 4$.

Minden más esetben az $R(x, y)$ értéke HAMIS. De ha az $R(1, z)$ atomot tekintjük, akkor ennek az értéke akkor IGAZ, ha $z = 2$ és minden más esetben HAMIS. □

4.2.2. Aritmetikai atomok

A Datalogban létezik egy másik fajta fontos atom, az *aritmetikai atom*. Ez a fajta atom tulajdonképpen két aritmetikai kifejezés összehasonlítása, mint például $x < y$ vagy $x + 1 \geq y + 4 \times z$. A kiemelés kedvéért a 4.2.1. alfejezetben bevezetett atomokat *relációs atomoknak* nevezzük.

Figyeljük meg, hogy az aritmetikai és relációs atomok egyaránt logikai függvények, és visszatérési értékük a változók értékétől függ. Valójában a < vagy ≥ típusú aritmetikai összehasonlítás tekinthető relációnak is, mégpedig olyan relációnak, amely minden olyan sort tartalmaz, amelyre az adott aritmetikai összehasonlítás eredménye igaz. Ilyen értelemben elképzelhetjük a < összehasonlításból származó „<" relációt. A reláció sorai azok a számpárok lennének, amelyekre igaz, hogy az első szám kisebb, mint a második, mint például az (1, 2) vagy (–1,5, 65,4). Ne feledjük, hogy egy adatbázis relációi mindig végesek és általában időről időre változnak. Ezzel ellentétben az olyan aritmetikai összehasonlításból származó relációk, mint például a <, végtelen sok sort tartalmaznak és tartalmuk is változatlan.

4.2.3. Datalog szabályok és lekérdezések

A Datalogban *szabályok* segítségével írhatunk le relációs algebrai műveletekhez hasonló műveleteket. Egy szabály a következőkből áll:

1. Egy *fejnek* nevezett relációs atom.

2. Ezt követi a ← szimbólum, amelyet gyakran „if"-nek („ha") olvasunk ki.

3. Ezt követi a *törzs*, amely egy vagy több *részcélnak* nevezett atomból áll. A részcélok lehetnek relációs vagy aritmetikai atomok. A részcélokat AND („és") szócskával kapcsoljuk össze, és a részcélok elé, ha szükséges, kitehetjük a tagadást jelentő NOT logikai műveletet is.

4.16. példa: A következő Datalog szabályt:

```
HosszúFilm(c,é) ← Film(c,é,h,sz,s,p) AND h ≥ 100
```

a 100 percnél hosszabb, úgynevezett hosszú filmek meghatározására használhatjuk. Ez a szabály a 3.9. alfejezetben definiált Film relációra utal, amelynek sémája:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

A szabály feje a *HosszúFilm*(c, $é$) atom. A szabály törzse két részcélból áll:

1. Az első részcél a *Film* nevű predikátumból és a Film reláció hat attribútumának megfelelő hat argumentumból áll. Az argumentumok különböző nevű változók: c a cím komponens, $é$ az év komponens, h a hossz komponens stb. Ezt a részcélt tekinthetjük úgy is, mint a Film reláció aktuális előfordulásának (c, $é$, h, sz, s, p) sorát. Pontosabban fogalmazva, a *Film*(c, $é$, h, sz, s, p) akkor igaz, ha a hat darab változó értéke megegyezik a Film reláció egy sorának hat komponensével.

2. A második részcél, $h \geq 100$, akkor igaz, ha a Film reláció megfelelő sorában a hossz komponens értéke legalább 100.

Az egész szabályt pedig a következőképpen tekinthetjük: a *HosszúFilm*(c, $é$) akkor igaz, ha létezik a Film relációnak egy olyan sora, amelyre teljesülnek a következők:

a) A cím és év attribútumoknak megfelelő két első komponens értéke c és $é$.

b) A hossz attribútumnak megfelelő harmadik komponens – a h – értéke legalább 100.

c) A negyedik, ötödik és hatodik komponensek értéke tetszőleges.

Figyeljük meg, hogy ez a szabály ekvivalens a következő relációs algebrai megfeleltetésnek:

!! **4.2.3. feladat:** Ahhoz, hogy a fejhez rendelt reláció véges legyen abban az esetben, ha mindegyik részcél predikátumához rendelt reláció véges, a biztonságos Datalog szabályokra adott feltétel elégséges, viszont túl erős. Adjunk példát olyan Datalog szabályra, amely nem biztonságos és mégis teljesíti a relációs predikátumokhoz véges relációkat hozzárendelve a fejhez rendelt reláció is véges.

4.3. Relációs algebra kifejezése Datalog szabályok segítségével

A relációs algebrai operátorok mindegyike kifejezhető Datalog szabályok segítségével. Ebben az alfejezetben valamennyi operátort egyenként megvizsgáljuk. Ezt követően a bonyolult relációs algebrai kifejezések Datalog szabályokká történő átírásával foglalkozunk.

4.3.1. Metszet

Két reláció metszete kifejezhető egyetlen Datalog szabály segítségével. Ez a szabály két, AND szócskával összekapcsolt részcélt tartalmaz, a két relációnak megfelelően. A részcélokban ugyanazok a változók szerepelnek a megfelelő argumentumokban.

4.20. példa: Vegyük a 4.1. ábrán látható R és S relációkat. Mindkét relációnak ugyanazok az attribútumai: *név, cím, nem, születésnap*. A két reláció metszetét a következő Datalog szabály segítségével írhatjuk fel:

```
M(v,c,n,sz) ← R(v,c,n,sz) AND S(v,c,n,sz)
```

Ebben a szabályban az M egy IDB predikátum. Ha mindkét részcél igaz, az azt jelenti, hogy a (v, c, n, sz) sor benne van az R és az S relációban is. Tehát az M IDB predikátumhoz rendelt reláció a szabály kiértékelése után pontosan az $R \cap S$ lesz. □

4.3.2. Egyesítés

Két reláció egyesítését két Datalog szabály segítségével írhatjuk fel. A szabályok feje ugyanaz az IDB predikátum. Törzsükben egyetlen atom szerepel, az első szabály törzsében az első relációnak megfelelő atom, a másodikban pedig a második relációnak megfelelő atom. Mindkét szabályban a fejben szereplő argumentumok megegyeznek a szabály részcéljainak argumentumaival.

4.21. példa: A 4.20. példában használt R és S relációk egyesítését a következő két szabály segítségével írhatjuk fel:

Egy szabályban használt változónevek lokálisak

Meg kell jegyeznünk, hogy egy szabály felírásakor a változók nevei tetszőlegesek és semmi kapcsolatuk nincs a többi szabályban használt változók neveivel. Ez azért lehetséges, mert minden egyes szabályt egyenként értékelünk ki, és a szabály fejéhez rendelt relációba a többi szabálytól függetlenül kerülnek be a megfelelő sorok. Írjuk át például a 4.21. példában felírt második szabályt a következő módon:

```
E(w,x,y,z) ← S(w,x,y,z)
```

Az első szabályt hagyjuk meg úgy, ahogyan a 4.21. példában felírtuk. A két szabály természetesen így is az R és S relációk egyesítését adja meg. Figyeljük meg azonban, hogyha egy a nevű változó nevét kicseréljük b-re egy szabályban, akkor az a valamennyi előfordulását ki kell cserélnünk b-re ebben a szabályban. Továbbá, az a változó nevét csak olyan névre cserélhetjük ki, amelyik nem szerepel máshol ebben a szabályban.

```
1. E(v,c,n,sz) ← R(v,c,n,sz)
2. E(v,c,n,sz) ← S(v,c,n,sz)
```

Az első szabály által elérjük, hogy az R reláció valamennyi sora bekerül az E IDB predikátumhoz rendelt relációba. A második szabállyal azt érjük el hasonló módon, hogy az S reláció valamennyi sora bekerül az E IDB relációba. Tehát a két szabály kiértékelése után az E IDB relációba az $R \cup S$ valamennyi sora bekerül. Ha nincs több olyan szabályunk, amelynek a feje az E predikátum, akkor nem kerülhet be több sor az E relációba. Tehát ebben az esetben az E pontosan az $R \cup S$ sorait tartalmazza.[3] Ne feledjük, hogy a relációk tulajdonképpen halmazok, így azok a sorok, amelyek benne vannak mind az R, mind az S relációkban, csak egyszer jelennek meg az E relációban. □

4.3.3. Különbség

Az R és S relációk különbségét egyetlen szabály segítségével írhatjuk fel. A szabály egy negált és egy nem negált részcélból áll. A nem negált részcél predikátuma az R, a negált részcél predikátuma pedig az S. A részcélokban és a fejben azonos változók szerepelnek a megfelelő argumentumokban.

[3] Tulajdonképpen valamenyi példában feltételezzük, hogy egy felírt szabály fejében szereplő IDB predikátum nem szerepel más, általunk fel nem írt szabályok fejében. Ha léteznének más szabályok is, amelyek fejében ugyanez az IDB predikátum szerepelne, akkor nem zárhatnánk ki más sorok létezését sem a vizsgált IDB relációban.

4.22. példa: Tekintsük a 4.20. példa *R* és *S* relációját. A

```
KÜL(v,c,n,sz) ← R(v,c,n,sz) AND NOT S(v,c,n,sz)
```

szabály kiértékelése után a *KÜL* reláció pontosan az *R* – *S* különbséggel lesz egyenlő. □

4.3.4. Vetítés

Az *R* reláció vetítéséhez egyetlen szabályt használunk. A szabálynak egyetlen részcélja van, az *R* predikátum. A részcél argumentumában a reláció különböző attribútumainak különböző változók felelnek meg. A fej argumentumában azoknak az attribútumoknak megfelelő változók szerepelnek, amelyekre a vetítés történik, és olyan sorrendben, ahogy a vetítésben megkívántuk.

4.23. példa: Tegyük fel, hogy a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

relációt szeretnénk levetíteni az első három attribútumára (lásd 4.2. példa). A

```
V(c,é,h) ← Film(c,é,h,sz,s,p)
```

szabály pontosan a vetítés eredményének megfelelő *V* relációt adja meg. □

4.3.5. Kiválasztás

A kiválasztás operátor kifejezése Datalogban már bonyolultabb dolog, mint az eddigi operátorok átírása. A legegyszerűbb eset, amikor a kiválasztás feltétele egy vagy több aritmetikai összehasonlítás AND szócskával történő összekapcsolásából áll. Ebben az esetben egyetlen szabályt kell készítenünk, amely a következőket fogja tartalmazni:

1. Egy relációs részcélt, amely megfelel annak a relációnak, amelyen a kiválasztás történik. Ez az atom annyi különböző változót tartalmaz, ahány attribútummal a reláció rendelkezik.

2. A kiválasztás minden egyes összehasonlításához egy aritmetikai részcélt, amely megegyezik az összehasonlítással. Amíg a kiválasztás feltételében attribútumnevek szerepelnek, addig az aritmetikai részcélokban azokat a változókat kell használnunk, amelyekkel az előző lépésben a relációs részcélt megadtuk.

4.24. példa: A 4.4. példában felírt kiválasztást

$\sigma_{hossz \geq 100 \text{ AND } stúdióNév = \text{'Fox'}}(\texttt{Film})$

a következő Datalog szabállyal fejezhetjük ki:

```
K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND h ≥ 100 AND s = 'Fox'
```

A kiértékelés eredménye a *K* reláció. Figyeljük meg, hogy mindkét részcélban a *h* és *s* változók a `Film` reláció `hossz` és `stúdióNév` attribútumainak felelnek meg. □

Most pedig nézzük meg mi történik azokkal a kiválasztásokkal, amelyek feltételében vannak olyan összehasonlítások, melyek OR (vagy) szócskával vannak összekapcsolva. Az ilyen kiválasztásokat nem tudjuk egyetlen Datalog szabály segítségével kifejezni. Viszont az ilyen kiválasztások átalakíthatók több kiválasztás egyesítésévé. Azonban az olyan kiválasztás, amelynek feltétele két részfeltétel OR összekapcsolásából áll, ekvivalens két olyan kiválasztás egyesítésével, amelyek az egyik illetve másik részfeltételhez tartoznak. Tehát az *n* feltétel OR szócskával történő összekapcsolását tartalmazó kiválasztás kifejezhető *n* darab, egyforma fejű szabállyal. Az *i*-edik szabály a kiválasztás *i*-edik feltételének felel meg.

4.25. példa: Cseréljük ki a 4.24. példa kiválasztásában az AND szócskát OR szócskára:

$\sigma_{hossz \geq 100 \text{ OR } stúdióNév = \text{'Fox'}}(\texttt{Film})$

Azaz, válasszuk ki azokat a filmeket, amelyek vagy hosszú filmek, vagy a Fox stúdióban készültek. Felírjuk a két feltételnek megfelelő két szabályt:

```
1. K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND h ≥ 100

2. K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND s = 'Fox'
```

Az első szabály megadja a 100 percnél hosszabb filmeket, a második pedig a Fox stúdióban készült filmeket. □

Az AND, OR és NOT logikai operátorok tetszőleges alkalmazásával már meglehetősen bonyolult kiválasztási feltételek is megfogalmazhatók. Egy jól ismert módszer segítségével (amelynek az ismertetésétől most eltekintünk), ezek a logikai kifejezések diszjunktív normál formára hozhatók. A diszjunktív normál forma több konjunkció OR szócskával történő összekapcsolása. Egy *konjunkció* literálok AND szócskával történő összekapcsolása, egy *literál* pedig egy negált vagy egy nem negált összehasonlítás.[4]

A literálokat részcélokkal tudjuk reprezentálni, melyek előtt szerepelhet egy NOT szócska is. Negált aritmetikai részcél esetén a NOT szócska bevihető az összehasonlításba. Például, a NOT x ≥ 100 feltétel felírható x < 100 alakban. Bármely konjunkció megfelel egyetlen olyan Datalog szabálynak, amelynek részcéljai maguk az össze-

[4] Lásd például: A. V. Aho és J. D. Ullman, *Foundation of Computer Science*, Computer Science Press, New York, 1992.

hasonlítások. Végül, bármely diszjunktív normál formájú kifejezés felírható a konjunkcióinak megfelelő Datalog szabályok segítségével. A szabályok kiértékelésének eredménye tulajdonképpen a konjunkciók eredményeinek egyesítése.

4.26. példa: A 4.25. példában a fenti algoritmus egy egyszerű példáját láthattuk. Ha az előző példa feltételét negáljuk, akkor egy jóval bonyolultabb kifejezést kapunk:

$\sigma_{\text{NOT (}hossz \geq 100 \text{ OR } stúdióNév = \text{'Fox'}\text{)}}$ (Film)

Azaz, válasszuk ki azokat a filmeket, amelyek sem nem hosszúak, sem nem a Fox stúdióban készültek.

Ebben az esetben a NOT nem egy egyszerű összehasonlítás előtt áll, ezért, a *DeMorgan-szabály* alkalmazásával be kell vinnünk a NOT operátort a kifejezés belsejébe. A következő kifejezést kapjuk:

$\sigma_{\text{(NOT (}hossz \geq 100\text{)) AND (NOT (}stúdióNév = \text{'Fox'}\text{))}}$ (Film)

A NOT operátort bevihetjük az összehasonlítások belsejébe:

$\sigma_{hossz < 100 \text{ AND } stúdióNév \neq \text{'Fox'}}$ (Film)

Ez a kifejezés Datalogban a következő szabállyal írható fel:

```
K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND h < 100 AND s ≠ 'Fox'
```
□

4.27. példa: Vegyünk egy hasonló példát, ahol a kiválasztás feltételében AND operátorral összekapcsolt feltételek negáltja szerepel. Ezúttal a DeMorgan-szabály második formáját alkalmazzuk. Kezdjük a relációs algebrai kifejezéssel:

$\sigma_{\text{NOT (}hossz \geq 100 \text{ AND } stúdióNév = \text{'Fox'}\text{)}}$ (Film)

Azaz, válasszuk ki azokat a filmeket, amelyek nem a Fox stúdióban készült nem hosszú filmek.

A NOT operátort a DeMorgan-szabály segítségével bevisszük a kifejezés belsejébe:

$\sigma_{\text{NOT (}hossz \geq 100\text{) OR NOT (}stúdióNév = \text{'Fox'}\text{)}}$ (Film)

Ezután bevisszük a NOT operátort az összehasonlítások belsejébe:

$\sigma_{hossz < 100 \text{ OR } stúdióNév \neq \text{'Fox'}}$ (Film)

Legvégül felírunk két Datalog szabályt, egyet-egyet az OR két oldalán található feltételeknek megfelelően:


```
1. K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND h < 100

2. K(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND s ≠ 'Fox'
```
□

4.3.6. Szorzat

Két reláció szorzata, $R \times S$, kifejezhető egyetlen Datalog szabály segítségével. A szabálynak két részcélja van, egyik az R-nek, a másik az S-nek felel meg. A részcélok változói megfelelnek a relációk attribútumainak, de egymástól különbözők. A fejben szereplő IDB predikátum argumentumában az összes olyan változó szerepel, amely valamely részcélban megjelenik, és a sorrendet tekintve, az R részcél változóit követik az S részcél változói.

4.28. példa: Tekintsük a 4.20. példa relációit. Az R és S egyaránt négy attribútummal rendelkezik. Az $R \times S$ szorzatot a következő Datalog szabállyal fejezhetjük ki:

```
SZ(a,b,c,d,w,x,y,z) ← R(a,b,c,d) AND S(w,x,y,z)
```

Az R részcél változóinak nevét az ábécé elejéről vettük, míg az S részcél változóinak nevét az ábécé végéről. Az R és S részcélok mind a nyolc változója megjelenik a szabály fejében. □

4.3.7. Összekapcsolás

Két reláció természetes összekapcsolásának kifejezése Datalogban nagyon hasonlít a két reláció szorzatának megfelelő szabályhoz. A különbség csak annyi, hogy amikor az $R \bowtie S$ természetes összekapcsolást akarjuk felírni, akkor az R és S relációk közös attribútumaihoz tartozó változóknak ugyanazt a nevet kell adnunk és különböző változókat kell a többi helyen használni. Akár az attribútumok neveit is használhatjuk változónévként. A fej egy olyan IDB predikátum kell legyen, amelyben minden egyes változó megjelenik, méghozzá egyszer.

4.29. példa: Vegyük az $R(A, B)$ és $S(B, C, D)$ relációkat a megadott sémákkal. Természetes összekapcsolásukat a következő szabállyal fejezhetjük ki:

```
TÖ(a,b,c,d) ← R(a,b) AND S(b,c,d)
```

Figyeljük meg, hogy a részcélokban használt változók egyértelműen megfelelnek az R és S relációk attribútumainak. □

A théta-összekapcsolás hasonló módon írható fel Datalogban. A 4.1.9. alfejezetben

láthattuk, hogy a théta-összekapcsolás kifejezhető szorzás és egy ezt követő kiválasztás segítségével. Ha a kiválasztás feltétele egy konjunkció, azaz összehasonlítások AND szócskával történő összekapcsolása, akkor egyszerűen felírjuk a szorzásnak megfelelő Datalog szabályt, majd ehhez hozzáadjuk az összehasonlításoknak megfelelő aritmetikai részcélokat.

4.30. példa: Tekintsük a 4.9. példából az $U(A, B, C)$ és $V(B, C, D)$ relációk théta-összekapcsolását:

$$U \bowtie_{A<D \text{ AND } U.B \neq V.B} V$$

A következő Datalog szabállyal ugyanezt a műveletet végezzük el:

```
Ö(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND
                     a<d AND ub≠vb
```

Az U reláció B attribútumához rendelt változónak az *ub* nevet adtuk, és ugyanígy kapták nevüket a *vb*, *uc*, *vc* változók is. A két relációnak összesen hat attribútuma van, és a hozzájuk rendelt változók tetszőleges különböző nevet kaphattak volna. Az első két részcél az összekapcsolni kívánt relációknak, míg a két utolsó részcél a théta-összekapcsolás feltételében szereplő összehasonlításoknak felel meg. □

Ha a théta-összekapcsolás feltétele nem egy konjunkció, akkor a feltételt 4.3.5. alfejezetben tárgyalt módon átalakítjuk diszjunktív normál formára. Ezután a feltétel minden egyes konjunkciójához felírjuk a megfelelő szabályt. Egy ilyen szabály a szorzathoz tartozó részcélokkal kezdődik, amelyet a konjunkcióban szereplő literálokhoz tartozó részcélok követnek. Minden szabály feje egyforma, és argumentumként annyi változót tartalmaz, ahány attribútuma a théta-összekapcsolásban részt vevő két relációnak öşszesen van.

4.31. példa: Módosítsunk egy kicsit a 4.30. példában megadott relációs algebrai kifejezésen. Cseréljük ki az AND operátort OR operátorra.

$$U \bowtie_{A<D \text{ OR } U.B \neq V.B} V$$

Ily módon a feltétel rögtön diszjunktív normál formában van, hiszen nem szerepel benne negáció. Két konjunkciónk van, mindkettő egyetlen literálból áll.

Ha a változókat ugyanúgy nevezzük, mint a 4.30. példában, akkor a következő két szabályt kapjuk:

```
1. Ö(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND a < d

2. Ö(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND ub ≠ vb
```

A szabályokban a két relációnak megfelelő részcél mellett az $A<D$ vagy az $U.B \neq V.B$ összehasonlításnak megfelelő részcél szerepel. □

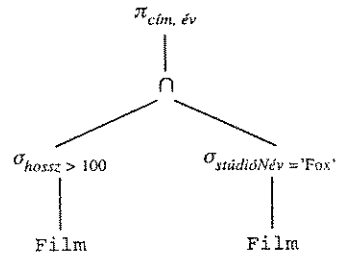
4.3.8. Kifejezések megadása Datalogban

Datalog szabályokkal nem csak elemi relációs algebrai operátorokat fejezhetünk ki, hanem alapjában véve bármilyen relációs algebrai kifejezést. A trükk mindössze annyi, hogy felrajzoljuk a relációs algebrai kifejezéshez tartozó kifejezésfát, és minden egyes belső csúcshoz megadjuk a megfelelő IDB predikátumot. Az IDB predikátumhoz tartozó szabályt vagy szabályokat úgy kapjuk, hogy a fa megfelelő csúcsában található operátorra alkalmazzuk az előzőekben áttekintett átírási módszert. A kifejezésfa extenzionális operandusaira (amelyek tulajdonképpen az adatbázis relációi) a hozzájuk tartozó predikátumokkal hivatkozhatunk. A belső csúcsok operandusaira a megfelelő IDB predikátumokkal hivatkozhatunk.

4.32. példa: Vegyük a 4.10. példa relációs algebrai kifejezését.

$$\pi_{cím,év}(\sigma_{hossz \geq 100}(\texttt{Film}) \cap \sigma_{stúdióNév = \text{'Fox'}}(\texttt{Film}))$$

A megfelelő kifejezésfát a 4.7. ábrán láthatjuk, de a könnyebb követhetőség érdekében a 4.14. ábrán is ugyanez látható.



4.14. ábra. *A kifejezésfa*

Négy IDB predikátumot kell felírnunk a négy belső csúcsnak megfelelően. Mindegyik predikátumhoz egyetlen szabály fog tartozni, amint az a 4.15. ábrán látható.

A két alsó belső csúcs egyszerű kiválasztás a Film EDB relációra alkalmazva. A hozzájuk tartozó IDB predikátumok a W és az X, amelyeket a 4.15. ábra első két szabályában írtunk fel. Az első szabály például a Film reláció azon sorait adja meg, amelyekben a film hosszúsága legalább 100 perc.

```
1. W(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND h ≥ 100
2. X(c,é,h,sz,s,p) ← Film(c,é,h,sz,s,p) AND s = 'Fox'
3. Y(c,é,h,sz,s,p) ← W(c,é,h,sz,s,p) AND X(c,é,h,sz,s,p)
4. Z(c,é) ← Y(c,é,h,sz,s,p)
```

4.15. ábra. *Összetett relációs algebrai kifejezésnek megfeleltetett Datalog szabályok*

A harmadik szabály a W és X metszetének megfelelő Y predikátumot adja meg, a 4.3.1. alfejezetben bemutatott módon. Végül a negyedik szabály az Y levetítése a `cím` és `év` attribútumokra, a 4.3.4. alfejezetben bemutatott módon. A Z predikátum tulajdonképpen a „válasz" predikátum. A szabályokat kiértékelve a Z predikátumnak megfelelő relációban ugyanazok a sorok lesznek, mint a kiinduló relációs algebrai kifejezés eredményében. (Természetesen csak akkor, ha a `Film` reláció mindkét kiértékeléskor ugyanazokat a sorokat tartalmazza.)

Figyeljük meg, hogy ebben a példában a 4.15. ábra negyedik szabályában az Y részcél helyettesíthető a harmadik szabály törzsével. Sőt, a W és az X részcélok is helyettesíthetők az első, illetve a második szabály törzsével. A `Film` részcél mindkét törzsben szerepel, de elég csak egyet megtartanunk belőle. Végeredményben a Z egyetlen szabály segítségével is megadható:

```
Z(c,é) ← Film(c,é,h,sz,s,p) AND h ≥ 100 AND s = 'Fox'
```

Általában azonban nem igaz, hogy tetszőleges bonyolultságú relációs algebrai kifejezés átírható egyetlen Datalog szabállyá. □

4.3.9. Feladatok

4.3.1. feladat: Adottak az $R(a, b, c)$, $S(a, b, c)$ és $T(a, b, c)$ relációk. Írjuk fel a következő relációs algebrai kifejezéseket egy vagy több Datalog szabály segítségével:

a) $R \cup S$

b) $R \cap S$

c) $R - S$

* d) $(R \cup S) - T$

! e) $(R - S) \cap (R - T)$

f) $\pi_{a,b}(R)$

*! g) $\pi_{a,b}(R) \cap \rho_{U(a,b)}(\pi_{b,c}(S))$

4.3.2. feladat: Adott az $R(x, y, z)$ reláció. Írjuk fel egy vagy több Datalog szabályok egyesítésével a $\sigma_C(R)$ kifejezést, ahol C a következő alakú:

a) $x = y$

* b) $x < y$ AND $y < z$

c) $x<y$ OR $y<z$

d) NOT($x<y$ OR $x>y$)

*! e) NOT(($x<y$ OR $x>y$) AND $y<z$)

! f) NOT(($x<y$ OR $x<z$) AND $y<z$)

4.3.3. feladat: Adottak az $R(a, b, c)$, $S(b, c, d)$ és $T(d, e)$ relációk. Írjuk fel a következő természetes összekapcsolásokat egyetlen Datalog szabály segítségével:

a) $R \bowtie S$

b) $S \bowtie T$

! c) $(R \bowtie S) \bowtie T$ (*Megjegyzés:* mivel a természetes összekapcsolás asszociatív és kommutatív, ezért mindegy, hogy milyen sorrendben kapcsoljuk össze a három relációt.)

4.3.4. feladat: Adottak az $R(x, y, z)$ és $S(x, y, z)$ relációk. Írjuk fel az $R \bowtie_C S$ théta-összekapcsolásnak megfelelő Datalog szabályt (illetve szabályokat), ha a C feltétel a 4.3.2. példában felsorolt feltételek egyike. A feltételekben található aritmetikai összehasonlításokat tekintsük úgy, mint amelyik sorrendben az R egy attribútumát hasonlítja össze az S egy attribútumával. Az $x<y$ alakú összehasonlítás értelmezése tehát $R.x<S.y$.

! **4.3.5. feladat:** Datalog szabályok is felírhatók velük ekvivalens relációs algebrai kifejezésekkel. Próbálkozzunk meg néhány egyszerű esettel, annak ellenére, hogy nem mutattunk be erre szolgáló általános módszert. A következő Datalog szabályokat írjuk át olyan relációs algebrai kifejezésekké, amelyek eredménye megegyezik a fejnek megfelelő relációval.

* a)
```
P(x,y) ← Q(x,z) AND R(z,y)
```

b)
```
P(x,y) ← Q(x,z) AND Q(z,y)
```

c)
```
P(x,y) ← Q(x,z) AND R(z,y) AND x<y
```

4.4. Rekurzív programozás Datalogban

Nem minden számítást tudunk relációs algebrai kifejezés segítségével felírni, annak ellenére, hogy a relációs algebrában sok hasznos művelet kifejezhető. Nem tudjuk például relációs algebrai kifejezéssel felírni az olyan tipikus adatműveletet, amely hason-

ló, de növekvő algebrai kifejezés végtelen ismétlődő sorozatát tartalmazza, amit *rekurziónak* nevezünk.

4.33. példa: A filmiparban elterjedt folytatásos filmek jelensége egy jó példa a rekurzióra. A sikeres filmeknek gyakran elkészítik a folytatását is, sőt ha a folytatás is sikeres, akkor ennek is lesz folytatása, és így tovább. Így aztán egy filmből egy egész sorozat kiindulhat. Tegyük fel, hogy van egy olyan `Folytatása(film, folytatás)` relációnk, amelybe azok a filmpárok kerülnek, amelyek egymás közvetlen folytatásai. Például:

film	*folytatás*
`Csupasz pisztoly`	`Csupasz pisztoly` $21/2$
`Csupasz pisztoly` $21/2$	`Csupasz pisztoly` $33 1/3$

Ennél jóval általánosabb fogalom egy adott filmből kiinduló *sorozat*, amely tartalmazza a film folytatását, a folytatás folytatását, és így tovább. A fenti relációban a *Csupasz pisztoly 33 1/3* benne van a *Csupasz pisztoly* indította sorozatban, de szigorúan véve nem folytatása a *Csupasz pisztoly* című filmnek. Ha csak a közvetlen folytatásokat tároljuk és a `sorozat` relációt mindig kiszámoljuk, amikor szükségünk van rá, azzal helyet takaríthatunk meg. Az előbbi példában csak eggyel kevesebb sort kell így tárolnunk, de ha az öt folytatással rendelkező *Rocky* filmre gondolunk, akkor ez a megtakarítás már hat sor, sőt, a 18 részes *Péntek 13* esetén ez a megtakarítás már 136 sor.

A `Sorozat` reláció kiszámítása a `Folytatása` relációból nem is olyan egyszerű feladat. A folytatások folytatását megkaphatjuk, ha a `Folytatása` relációt önmagával összekapcsoljuk. Ahhoz, hogy az összekapcsolás valóban a kívánt természetes összekapcsolássá váljon, átnevezéseket kell végrehajtani:

$$\pi_{első,harmadik}(\rho_{R(első,második)}(\texttt{Folytatása}) \bowtie \rho_{S(második,harmadik)}(\texttt{Folytatása}))$$

Ebben a kifejezésben a `Folytatása` relációt kétszer is átneveztük. Az első átnevezéskor az attribútumok az `első`, illetve `második` nevet kapják, a második átnevezés alkalmából pedig a `második`, illetve `harmadik` nevet kapják. Ezt azért tettük, mert a természetes összekapcsoláshoz a `Folytatása` olyan (f_1, f_2) és (f_3, f_4) sorai szükségesek, ahol $f_2 = f_3$. Ebből olyan (f_1, f_4) sorok keletkeznek, ahol az f_4 az f_1 film közvetlen folytatásának a közvetlen folytatása.

Hasonló módon összekapcsolhatjuk a `Folytatása` relációt háromszor is önmagával. Ekkor a folytatás folytatásának folytatását kapjuk meg (lásd *Rocky* és *Rocky IV*). Sőt, akármilyen rögzített i értékre megkaphatjuk egy film i-edik folytatását, ha a `Folytatása` relációt $(i-1)$-szer összekapcsoljuk önmagával. Ha ezután a `Folytatása` relációt egyesítjük a véges sok összekapcsolásokból kapott relációkkal, akkor megkaphatunk egy adott filmből kiinduló egész sorozatot rögzített i értékig.

Van valami, amit viszont relációs algebrában nem tudunk kifejezni. Nem vehetjük a „végtelen egyesítését" annak a végtelen kifejezéssorozatnak, amely minden egyes

i-re megadja a filmek i-edik folytatását. Ne feledjük, hogy a relációs algebrában csak két reláció egyesítését írhatjuk fel, végtelen számú relációt nem tudunk ily módon egyesíteni. Az egyesítés operátort alkalmazhatjuk ugyan tetszőlegesen véges sokszor egy relációs algebrai kifejezésben, de végtelen sok reláció egyesítését relációs algebrában nem lehet kifejezni. □

4.4.1. Fixpontoperátor

Szerencsére nem kell bevezetnünk a relációs algebrába egy oda nem illő jelölést végtelen sok „hasonló" kifejezés egyesítésére. Létezik egy szokásos megoldás arra, hogyan fejezzünk ki egy, a `Sorozat(x, y)`-hoz hasonló relációt (ahol y az x film általános értelmű folytatása a 4.33. példa szerint), amelyet egy végtelen, de szabályos eljárással kapunk más relációkból, mint esetünkben a `Folytatása` relációból. Felírunk egy olyan egyenletet, amelyben a `Sorozat` relációt önmagával és a `Folytatása` relációval fejezzük ki. és ezután azt mondjuk, hogy a `Sorozat` értéke az a legkisebb reláció *(legkisebb fixpont)*, amelyik kielégíti ezt az egyenletet. Az egyenlet legkisebb fixpontjára az egyenlet elé írt ϕ szimbólummal hivatkozhatunk.

4.34. példa: A `Sorozat(x, y)` relációt a következő kifejezéssel adhatjuk meg:

$$\phi(\texttt{Sorozat} = \rho_{Folytatása(x,y)}(\texttt{Folytatása}) \cup \rho_{R(x,y)}(\pi_{film,y}(\texttt{Folytatása} \bowtie_{folytatás = x} \texttt{Sorozat})))$$

Az egyenlet szemléletes jelentése a következő: az y film akkor tagja az x film indította sorozatnak, ha vagy közvetlen folytatása az x filmnek, vagy pedig egy olyan sorozat tagja, amelyet az x film egyik közvetlen folytatása indított el.

Az egyenlet megértéséhez figyelembe kell vennünk, hogy a `Sorozat` reláció két attribútuma x és y. A `Sorozat` relációt két kifejezés egyesítésével tettük egyenlővé. Az első kifejezés, a $\rho_{Folytatása(x,y)}(\texttt{Folytatása})$ tulajdonképpen maga a `Folytatása` reláció, csak az attribútumokat neveztük át azért, hogy megegyezzenek a `Sorozat` attribútumaival. A második kifejezés magja egy théta-összekapcsolás, a $\texttt{Folytatása} \bowtie_{folytatás = x} \texttt{Sorozat}$, amely a `Folytatása` összes (a, b) sorát összekapcsolja a `Sorozat` (b, c) soraival. Az eredmény olyan (a, b, b, c) típusú sorokból áll, amelyek attribútumai: `film, folytatás, x, y`. Ezután ezt a théta-összekapcsolást levetítjük az első és a negyedik attribútumára, azaz a `film` és az `y` attribútumokra. Végül átnevezzük a két attribútumot `x`-re és `y`-ra.

A fixpontegyenletben a `Sorozat` relációt egyenlővé tettük a `Folytatása` reláció és a folytatásokból eredő sorozatok egyesítésével. Így aztán a `Sorozat` minden olyan (x, y) filmpárt tartalmaz, ami vagy tagja a `Folytatása` relációnak, vagy az y egy olyan film, ami tagja az x film egy közvetlen folytatásából eredő sorozatnak. Másképp fogalmazva, az y film az x film folytatásának a folytatásának a folytatásának ... a folytatása, ahol a három pont akárhány „folytatásának" szót jelenthet. □

4.4.2. A legkisebb fixpont kiszámítása

Talán nem teljesen érthető a 4.34. példában felírt egyenletből, hogy a legkisebb megoldása miért adja meg pontosan az általunk keresett `Sorozat` relációt. Ahhoz, hogy a legkisebb fixpontoperátor jelentését megértsük, ismernünk kell az operátor kiszámításának módját. A következő módszer mindazokra az egyenletekre használható, amelyekben nem szerepel a különbség operátor. A 4.4.4. alfejezetben majd részletesebben foglalkozunk a különbség okozta problémákkal. Tehát a módszer:

1. Első lépésként feltételezzük, hogy az egyenlet bal oldalán álló R reláció üres.

2. Az egyenlet jobb oldalának ismételt kiértékelésével, felhasználva az R régi értékét, kiszámoljuk az R reláció új értékét.

3. Az előző lépést mindaddig ismételjük, amíg a régi és az új érték meg nem egyezik.

4.35. példa: Nézzük meg, a `Sorozat` reláció kiszámítását, ha a `Folytatása` reláció a következő három sort tartalmazza:

film	*folytatás*
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

Az első menetben a `Sorozat` reláció üres. Ezért a fixpontegyenletben a `Folytatása` és a `Sorozat` összekapcsolása is üres lesz. Sorokat csak az egyesítés bal oldali kifejezéséből, a `Folytatása` relációból kapunk. Tehát az 1. menet után a `Sorozat` reláció megegyezik a `Folytatása` relációval, amint azt a 4.16. ábrán láthatjuk.

x	*y*
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

4.16. ábra. *A `Sorozat` reláció az 1. menet után*

A második menetben ismét kiértékeljük az egyenlet jobb oldalát, de itt már a `Sorozat` helyébe a 4.16. ábrán látható relációt tesszük. Az egyesítés bal oldalán álló `Folytatása` reláció ugyanazt a három sort tartalmazza, mint amelyeket az előző menetben már megkaptunk. Az egyesítés jobb oldali kifejezésében össze kell kapcsolnunk a `Folytatása` relációt a `Sorozat` reláció aktuális értékével, ami megegyezik a `Folytatása` relációval, és a 4.16. ábrán látható. Az összekapcsolás elvégzéséhez megkeressük a két reláció azon sorait, amelyekre igaz az, hogy a `Folytatása` reláció sorának második komponense egyenlő a `Sorozat` első komponensével.

Összepárosítjuk a `Folytatása (Rocky, Rocky II)` és a `Sorozat (Rocky II, Rocky III)` sorait és a `(Rocky, Rocky III)` sort kapjuk. Ugyanígy a `Folytatása (Rocky II, Rocky III)` és a `Sorozat (Rocky III, Rocky IV)` soraiból a `(Rocky II, Rocky IV)` sort kapjuk. A megkapott sorokat beteszszük a `Sorozat` relációba. A `Folytatása` reláció többi sorát nem lehet a `Sorozat` reláció egyetlen régi sorával sem összepárosítani. Így aztán a második menet után a `Sorozat` relációnak öt sora lesz, amint az a 4.17. ábrán látható. Míg a 4.16. ábra csak a közvetlen folytatásokat tartalmazza, addig a 4.17. ábra a közvetlen folytatásokat és ezek közvetlen folytatásait is tartalmazza.

x	*y*
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky II
Rocky II	Rocky IV

4.17. ábra. *A `Sorozat` reláció a 2. menet után*

A harmadik menetben ismét kiértékeljük a fixpontegyenlet jobb oldalát, de most a `Sorozat` reláció helyébe a 4.17. ábrán látható relációt helyettesítjük. Természetesen az összes eddig megkapott sorunk megmarad. Sőt, a `Folytatása (Rocky, Rocky II)` és a `Sorozat (Rocky II, Rocky IV)` sorainak összepárosításával egy új sort is kapunk, a `(Rocky, Rocky IV)` sort. Így aztán a 3. menet után a `Sorozat` reláció a 4.18. ábrán látható sorokat tartalmazza.

x	*y*
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV
Rocky	Rocky IV

4.18. ábra. *A `Sorozat` reláció a 3. menet után*

A negyedik menetben már nem kapunk újabb sorokat, így a fixpont kiszámítása befejeződött. A fixpontszámítással megkapott `Sorozat` reláció a 4.18. ábrán látható. □

4.4.3. Fixpontegyenletek Datalogban

A gyakorlatban is fontos fixpontegyenletek felírásához szükséges relációs algebrai kifejezések néha egészen bonyolulttá válhatnak. Ilyenkor egyszerűbb őket Datalog szabályokkal kifejezni. Éppen ezért a fejezet további részében ezt a jelölési formát fogjuk

használni. Amint azt az 5.10. alfejezetben látni fogjuk, az SQL3-ban implementált fixpontjelölés inkább algebrai, mint logikai, mert az algebrai megközelítés közelebb áll az SQL valódi jellegéhez.

Nézzük meg mi is áll a logikai fixpont egyenletek hátterében. Először vesszük azokat a relációkat, amelyek értékét ismerjük. Ezek az extenzionális vagy röviden EDB relációk. Az összes többi reláció megjelenik egy vagy több szabály fejében, ezek az intenzionális vagy röviden IDB relációk. Ezek azok a relációk, amelyeket mi definiálunk a szabályok segítségével. A szabályok törzsében részcélok szerepelnek. A részcélok predikátumai lehetnek EDB vagy IDB relációk éppúgy, mint aritmetikai atomok. Ha egy vagy több olyan IDB reláció van, amelyeket együttesen olyan szabályok határoznak meg, amelyek törzsében ugyanezek az IDB relációk is előfordulnak, akkor ezek a szabályok valójában egy ugyanolyan fixpontegyenlet segítségével definiálják az IDB relációt vagy relációkat, mint a 4.34. példa relációs algebrai egyenlete.

4.36. példa: A `Sorozat` nevű IDB relációt a következő két Datalog szabállyal definiálhatjuk:

```
1. Sorozat(x,y) ← Folytatása(x,y)

2. Sorozat(x,y) ← Folytatása(x,z) AND Sorozat(z,y)
```

A kiindulási pont az első szabály, amely szerint egy film valamennyi közvetlen folytatása tagja a film indította sorozatnak. Ez a szabály megfelel a 4.34. példa egyenletében szereplő egyesítés első tagjának.

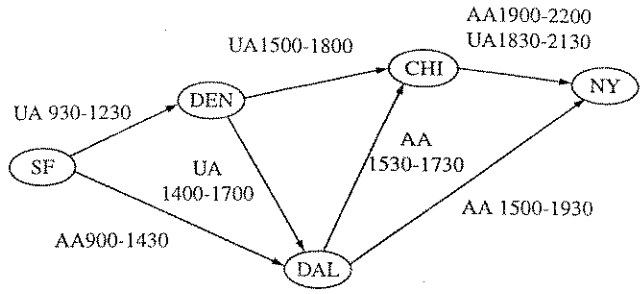
A második szabály szerint egy film közvetlen folytatásából indult sorozat valamennyi filmje tagja a film indította sorozatnak. Pontosabban szólva: ha z az x közvetlen folytatása és y tagja a z indította sorozatnak, akkor y tagja az x indította sorozatnak is. □

A 4.36. példa szabályai ugyanazt fejezik ki, mint a 4.35. példa fixpontegyenlete. Éppen ezért a `Sorozat` kiszámítása is ugyanúgy történik, mint a 4.35. példában. A negációmentes Datalog szabályokra általánosan érvényes az IDB szabályoknak ez a kiszámítási módja. Üres IDB relációkkal indulunk, és addig alkalmazzuk a szabályokat az EDB relációkra valamint az IDB relációk megfelelő értékeire – ezáltal megkapva az IDB relációk új értékét –, ameddig az összes IDB reláció értéke az alkalmazás után meg nem egyezik az alkalmazás előtti értékekkel.

4.37. példa: A rekurzió alkalmazásának egy bonyolultabb módját láthatjuk a következő, útvonalakkal foglalkozó gráfelméleti példában. A 4.19. ábrán látható gráfon két képzeletbeli légitársaság repülőjáratait tüntettük fel, San Francisco, Denver, Dallas, Chicago és New York között. A két fiktív légitársaság neve: UA *(Untried Airlines)* és AA *(Arcane Airlines)*.

A repülőjáratok adatait a következő EDB reláció tartalmazza:

```
Járatok(légitársaság, honnan, hová, felszállás,
megérkezés)
```

4.19. ábra. *Repülőjáratok térképe*

A fenti reláció 4.19. ábrának megfelelő adatait a 4.20. ábrán láthatjuk.

A legegyszerűbb rekurzív kérdés: „Mely (x, y) városokra igaz az, hogy x városból a megadott járatok segítségével valahány átszállással eljuthatunk y városba?" A következő két szabály által meghatározott reláció pontosan ezeket a városokat tartalmazza.

```
1. Eljut(x,y) ← Járatok(l,x,y,f,m)

2. Eljut(x,y) ← Eljut(x,z) AND Eljut(z,y)
```

Az első szabály alapján az `Eljut` tartalmazza az összes olyan várost, amelyek között létezik közvetlen járat. Az, hogy a járat melyik légitársaság gépe, mikor indul és mikor érkezik ebből a szempontból lényegtelen, tehát az l, f, m tetszőleges változók. A második szabály azt mondja ki, hogyha x városból el lehet jutni z városba, z városból viszont el lehet jutni y városba, akkor ily módon x városból el lehet jutni y városba. Figyeljük meg, hogy itt nem lineáris rekurziót használtunk. (Lásd „A rekurzió néhány formája" című keretes részt.) Ebben az esetben a rekurziónak ez a formája alkalmasabb, hiszen ha a `Járatok` relációt is használnánk a szabályban, akkor hárommal több változóra lenne szükségünk a `Járatok` fentebb említett három lényegtelen komponense miatt.

légitársaság	*honnan*	*hová*	*felszállás*	*megérkezés*
UA	SF	DEN	930	1230
AA	SF	DAL	900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

4.20. ábra. *A `Járatok` reláció sorai*

A rekurzió néhány formája*

A 4.34. és a 4.36. példában a `Sorozat` rekurzív reláció a `Folytatása` EDB reláció után szerepel, a rekurziónak ezt a formáját nevezzük *jobb oldali rekurziónak*. Ha a rekurzív relációt az EDB reláció elé írjuk, a következő *bal oldali rekurzív* szabályokat kapjuk:

```
1. Sorozat(x,y) ← Folytatása(x,y)

2. Sorozat(x,y) ← Sorozat(x,z) AND Folytatása(z,y)
```

Azaz, *y* akkor tagja az *x* indította sorozatnak, ha vagy az *x* közvetlen folytatása, vagy pedig egy olyan film közvetlen folytatása, amely tagja az *x* indította sorozatnak.

A rekurzív relációt kétszer is használhatjuk, ekkor *nem lineáris rekurzióról* beszélünk.

```
1. Sorozat(x,y) ← Folytatása(x,y)

2. Sorozat(x,y) ← Sorozat(x,z) AND Sorozat(z,y)
```

Azaz, *y* akkor tagja az *x* indította sorozatnak, ha vagy az *x* közvetlen folytatása, vagy pedig benne van egy olyan film indította sorozatban, amely film tagja az *x* indította sorozatnak. A szabályok mindhárom formája ugyanazt a Sorozat relációt adja, azokat az (*x*, *y*) párokat, amelyre az *y* az *x* folytatásának a folytatásának a folytatásának ... a folytatása, ahol a három pont akárhány „folytatásának" szót jelenthet.

Az `Eljut` reláció kiértékelését a 4.4.2. alfejezetben ismertetett módon végezzük. Az első szabállyal kezdjük, ezáltal az `Eljut` relációba a következő sorok kerülnek: `(SF, DEN)`, `(SF, DAL)`, `(DEN, CHI)`, `(DEN, DAL)`, `(DAL, CHI)`, `(DAL, NY)` és `(CHI, NY)`. Ezek megfelelnek a 4.19. ábrán látható irányított gráf éleinek.

A következő lépésben a második szabályt alkalmazzuk, amely rekurzív. Ezáltal olyan élpárokat olvasztunk eggyé, amelyekre igaz, hogy az egyik él kezdő csúcsa megegyezik a másik él végcsúcsával. Ezáltal a következő új párokat kapjuk: `(SF, CHI)`, `(DEN, NY)` és `(SF, NY)`. A következő menetben az eddig megkapott 1, illetve 2 hosszúságú útvonalakat kombináljuk össze, azért, hogy újabb, maximum 4 hosszúságú útvonalakat kapjunk. A mi példánkban azonban ebben a menetben már

* Szerkesztői megjegyzés: Ezek a leggyakrabban előforduló egyszerű rekurziós formák, egyetlen rekurzív relációt tartalmaznak. A 4.40. példában két reláció kölcsönös rekurziójára látunk példát.

nem kapunk újabb útvonalakat. Az `Eljut` reláció ezek után az összes olyan (*x*, *y*) városokat tartalmazza, amelyekre igaz, hogy a 4.19. ábra alapján *x* városból el lehet jutni *y* városba. Történetesen ez éppen azokat az (*x*, *y*) párokat jelenti, amelyeknél az *y* város a 4.19. ábrán az *x* várostól jobbra helyezkedik el. □

4.38. példa: Jogos igény, hogy átszálláskor a következő gép indulásáig legyen legalább egy óránk. Ilyen feltételnek megfelelő útvonalak meghatározása viszont bonyolultabb feladat. Erre a célra az `Összeköttetés(x,y,f,m)` IDB relációt használjuk, amely reláció soraira igaz, hogy ha *x* városból *f* időpontban indulunk, akkor *m* időpontban érkezünk meg *y* városba. Ráadásul, ha átszállásra is szükség van, mindegyik csatlakozásig van legalább egy óránk.

Az `Összeköttetés` relációt meghatározó szabályok:[5]

```
1. Összeköttetés(x,y,f,m) ← Járatok(l,x,y,f,m)

2. Összeköttetés(x,y,f,m) ← Összeköttetés(x,z,f,t1)
        AND Összeköttetés(z,y,t2,m) AND t1<=t2-100
```

Az első menetben az `Összeköttetés` relációba csak az első szabály által kerülnek új sorok. A 4.21. ábra ezt a nyolc tényt tartalmazza. Mindegyik tény a 4.19. ábrán feltüntetett járatok egyikének felel meg. Figyeljük meg, hogy a 4.19. ábrán látható hét él egyike tulajdonképpen két, különböző időpontban közlekedő járatot jelöl.

x	*y*	*f*	*m*
SF	DEN	930	1230
SF	DAL	900	1430
DEN	CHI	1500	1800
DEN	DAL	1400	1700
DAL	CHI	1530	1730
DAL	NY	1500	1930
CHI	NY	1900	2200
CHI	NY	1830	2130

4.21. ábra. *Az `Összeköttetés` reláció sorai az első menet után*

A következő menetben ezeket a sorokat próbáljuk összekombinálni a második szabály alapján. Például, a második és az ötödik sor összekombinálásából a `(SF, CHI, 900, 1730)` sort kapjuk. Viszont a második és hatodik sort nem kombinálhatjuk össze, mivel az első gép 1430-kor érkezik Dallasba, és a második gép mindössze fél óra múlva, 1500-kor indul. A 4.22. ábra az `Összeköttetés` reláció második menet utáni sorait tartalmazza. A vonal fölött az első menetből származó sorokat láthatjuk, a vonal alatt a második menetben bekerült sorok találhatók.

[5] Ezek a szabályok csak akkor helyesek, ha feltételezzük, hogy egyetlen repülő sem indul vagy érkezik pont éjfélkor.

x	*y*	*f*	*m*
SF	DEN	930	1230
SF	DAL	900	1430
DEN	CHI	1500	1800
DEN	DAL	1400	1700
DAL	CHI	1530	1730
DAL	NY	1500	1930
CHI	NY	1900	2200
CHI	NY	1830	2130
SF	CHI	900	1730
SF	CHI	930	1800
SF	DAL	930	1700
DEN	NY	1500	2200
DAL	NY	1530	2130
DAL	NY	1530	2200

4.22. ábra. *Az* `Összeköttetés` *reláció sorai a második menet után*

A harmadik menetben, a második szabály alkalmazásakor a törzsben szereplő két `Összeköttetés` relációba elvileg be kell helyettesítenünk a 4.22. ábra mindegyik sorát, az összes lehetséges párosításban. A vonal fölötti sorokat azonban már a második menetben is használtuk, ezért azokban az esetekben, amikor mindkét sor a vonal fölött van, új sorokat nem kapunk. Ahhoz, hogy új sort kapjunk, a törzsben használt `Összeköttetés` sorok legalább egyike azon sorokból kell kikerüljön, amelyeket előző lépésben kaptunk. Ezek pedig a 4.22. ábra vonal alatti sorai.

x	*y*	*f*	*m*
SF	DEN	930	1230
SF	DAL	900	1430
DEN	CHI	1500	1800
DEN	DAL	1400	1700
DAL	CHI	1530	1730
DAL	NY	1500	1930
CHI	NY	1900	2200
CHI	NY	1830	2130
SF	CHI	900	1730
SF	CHI	930	1800
SF	DAL	930	1700
DEN	NY	1500	2200
DAL	NY	1530	2130
DAL	NY	1530	2200
SF	NY	900	2130
SF	NY	900	2200
SF	NY	930	2200

4.23. ábra. *Az* `Összeköttetés` *reláció sorai a harmadik menet után*

A harmadik menetben csupán három új sort kapunk. Ezeket a 4.23. ábra alján láthatjuk. Az ábrán látható két vonal elválasztja az első, a második és a harmadik menetben kapott sorokat. A negyedik menetben újabb sorokat már nem kapunk, ily módon a kiértékelés itt befejeződik. A teljes `Összeköttetés` relációt a 4.23. ábrán láthatjuk. □

4.4.4. Rekurzív szabályokban előforduló negáció

Rekurzív szabályokban is szükségünk lehet negáció használatára. A rekurzió és a negáció együttes használata történhet biztonságos és nem biztonságos módon. Általánosan a negáció használata csak olyan esetben helyénvaló, ha a negáció nem jelenik meg a fixpontművelet belsejében. A következő két példa közül az első ezt az elvet követi, a második nem, ami paradox helyzethez vezet. Láthatjuk majd, hogy rekurzió esetén csak a rétegzett negáció használható. A rétegezés pontos meghatározását a példák után láthatjuk.

4.39. példa: Tegyük fel, hogy szeretnénk meghatározni a 4.19. ábra azon (*x*, *y*) városait, amelyekre igaz, hogy *x* városból a UA járataival eljuthatunk *y* városba (esetleg más városokon keresztül), de az AA járataival nem. Írjunk fel a 4.37. példa `Eljut` predikátumához hasonlóan egy `UAeljut` predikátumot. Csupán a légitársaság nevét kell rögzítenünk.

```
1. UAeljut(x,y) ← Járatok(UA,x,y,f,m)

2. UAeljut(x,y) ← UAeljut(x,z) AND UAeljut(z,y)
```

Hasonló módon írhatjuk fel az `AAeljut` predikátumot, amelynek (*x*, *y*) soraira fennáll, hogy AA járatok segítségével eljuthatunk *x* városból *y* városba.

```
1. AAeljut(x,y) ← Járatok(AA,x,y,f,m)

2. AAeljut(x,y) ← AAeljut(x,z) AND AAeljut(z,y)
```

Most már a következő nem rekurzív szabály alapján könnyedén kiszámolhatjuk a `CsakUA` predikátumot, mely azon (*x*, *y*) párokat tartalmazza, amelyekre igaz, hogy *x* városból a UA járataival eljuthatunk *y* városba, de az AA járataival nem.

```
CsakUA(x,y) ← UAeljut(x,y) AND NOT AAeljut(x,y)
```

Ez a szabály tulajdonképpen a `UAeljut` és az `AAeljut` halmazelméleti különbségét adja meg.

A 4.19. ábrából kiindulva és a 4.4.2. alfejezetben bemutatott iteratív fixpontszámítási módszert alkalmazva a `UAeljut` a következő párokból áll: `(SF, DEN)`,

(SF, DAL), (SF, CHI), (SF, NY), (DEN, DAL), (DEN, CHI) (DEN, NY) és (CHI, NY). Hasonló módon számoljuk ki az AAeljut értékeit: (SF, DAL), (SF, CHI), (SF, NY), (DAL, CHI), (DAL, NY) és (CHI, NY). Ha vesszük a két halmaz különbségét, a következő párokat kapjuk: (SF, DEN), (DEN, DAL), (DEN, CHI) és (DEN, NY). A négy pár alkotta halmaz megfelel a CsakUA predikátum értékének. □

4.40. példa: Lássunk egy olyan absztrakt példát, ahol a dolgok nem működnek ilyen jól. Tegyük fel, hogy egyetlen EDB relációnk van, az *R*. Ez egy unáris (egyargumentumú) predikátum, egyetlen sora van, a (0). Két, ugyancsak unáris IDB predikátumunk van, *P* és *Q*. Ezeket a következő két szabállyal adjuk meg:

```
1. P(x) ← R(x) AND NOT Q(x)

2. Q(x) ← R(x) AND NOT P(x)
```

Ez tulajdonképpen azt jelenti, hogy az *R* reláció *x* eleme *P*-nek vagy *Q*-nak eleme, de mindkettőnek egyszerre nem. Figyeljük meg, hogy a szabályok rekurzívak, hiszen kölcsönösen hivatkoznak a másik szabály fejében található predikátumra.

Amikor a 4.4.1. alfejezetben a rekurzív szabályok jelentését definiáltuk, azt mondtuk, hogy ezekkel tulajdonképpen a legkisebb fixpontot akarjuk meghatározni, azokat a legkisebb relációkat, amelyek a szabályoknak megfelelő algebrai egyenleteknek megoldásai. Az első szabály a $P = R - Q$, a második szabály pedig a $Q = R - P$ kifejezésnek felel meg. Mivel az *R* egyetlen sora a (0), azt tudjuk, a *P*-ben és a *Q*-ban is csak a (0) lehet. De vajon melyikben van a (0)? Abban az esetben, ha mindkettő üres lenne, akkor az egyenletek nem teljesülnének; hiszen például a $P = R - Q$ egyenletből a $\emptyset = \{(0)\} - \emptyset$ hamis egyenlőséget kapnánk.

A $P = \{(0)\}$ és $Q = \emptyset$ értékek megoldásai mindkét egyenletnek. A $P = R - Q$ egyenletből a $\{(0)\} = \{(0)\} - \emptyset$ igaz egyenlőséget kapjuk míg a $Q = R - P$ egyenletből a $\{(0)\} = \emptyset - \{(0)\}$ egyenlőséget kapjuk, ami szintén igaz.

Azonban vehetjük a $P = \emptyset$ és a $Q = \{(0)\}$ értékeket is, és ezek az értékek is kielégítik mindkét egyenletet. Ezáltal két megoldásunk is van:

a) $P = \{(0)\}$ $\quad Q = \emptyset$

b) $P = \emptyset$ $\quad Q = \{(0)\}$

Mindkét megoldás minimális abban az értelemben, hogyha a relációk bármely sorát elhagynánk, a kapott relációk nem teljesítenék a szabályokat. Így aztán nem tudunk választani a két minimális fixpont, az a) és b) között. Ebből következően még arra az egyszerű kérdésre sem tudunk válaszolni, hogy „igaz-e a *P*(0)?". □

A 4.40. példában láthattuk, hogy ha a rekurzió és a negáció túlságosan egybegabalyodnak, akkor a rekurzív szabályokra, illetve a fixpontegyenletekre adott minimális

fixpont értelmezésünk nem használható. Több minimális fixpont is létezhet és ezek a fixpontok egymásnak ellentmondóak is lehetnek. Jó lenne, ha a rekurzív negáció értelmezésének létezne másik, használható megközelítése, de sajnos nincs általános egyetértés abban, hogy az ilyen szabályok, illetve egyenletek mit is jelentenek.

A hagyományoknál maradva, csak azokkal a rekurziókkal foglalkozunk tovább, amelyekben a negáció *rétegzett*. Az 5.10. alfejezetben tárgyalt SQL3 szabványban is megtalálhatjuk ezt a korlátozást. Látni fogjuk, hogy abban az esetben, ha a negáció rétegzett, létezik egy olyan algoritmus, amely egy kitüntetett fixpontot határoz meg (jóllehet több más fixpont is létezhet), és ez megfelel a szabályok jelentésére vonatkozó intuitív elvárásunknak. A rétegzettség meghatározása a következő:

1. Rajzoljunk egy olyan gráfot, amelynek csomópontjai az IDB predikátumok.

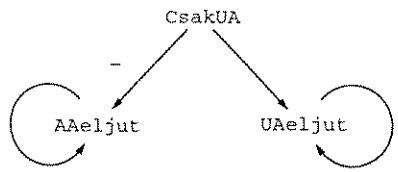
2. Ha az *A* predikátum egy szabály feje, a *B* predikátum pedig ugyanannak a szabálynak egy negált részcélja, akkor az *A* csúcsból kiindulva húzzunk élt a *B* csúcsba. Az ilyen élekre írjunk – jelet, jelezve, hogy ezek *negatív* élek.

3. Ha az *A* predikátum egy szabály feje, a *B* predikátum pedig ugyanannak a szabálynak egy nem negált részcélja, akkor az *A* csúcsból kiindulva húzzunk élt a *B* csúcsba. Az ilyen élekre ne írjunk semmit.

Ha az ily módon megkapott gráf tartalmaz negatív élt vagy éleket is tartalmazó kört, akkor a rekurzió nem rétegzett. Ellenkező esetben a gráf rétegzett. Az IDB predikátumokat rétegekbe sorolhatjuk. Egy *A* predikátum rétegének sorszámát úgy kapjuk meg, hogy megnézzük az *A* csúcsból kiinduló utak közül melyikben szerepel legtöbb negatív él, és ezen negatív élek száma adja meg a réteg sorszámát.

Ha a rekurzió rétegzett, akkor az IDB predikátumokat a rétegek sorszáma szerint értékeljük ki. Ez a stratégia a szabályok egy minimális fixpontját adja meg. Ennél is fontosabb, hogy az IDB predikátumok rétegek szerinti kiértékelésének mindig van értelme és a megfelelő fixpontot szolgáltatja. Ezzel ellentétben, a nem rétegzett rekurzió esetén még akkor is fixpont nélkül maradhatunk a kiértékelés során, ha több fixpont is létezik. (Lásd 4.40. példa.)

4.41. példa: A 4.39. példában szereplő szabályok gráfja a 4.24. ábrán látható. Az AAeljut és UAeljut predikátumok rétegének sorszáma 0, mert a csúcsokból kiinduló egyetlen út sem tartalmaz egyetlen negatív élt sem. A CsakUA rétegének sorszá-



4.24. ábra. *Egy rétegzett rekurzió gráfja*

4.25. ábra. *Egy nem rétegzett rekurzió gráfja*

ma 1, mivel létezik olyan út, amely ebből a csúcsból indul ki, és egy negatív élt tartalmaz, de ennél több negatív élt tartalmazó út nem indul ki ebből a csúcsból. Tehát mielőtt hozzákezdenénk a `CsakUA` kiértékelésének, ki kell értékelnünk az `AAeljut` és az `UAeljut` predikátumokat.

A 4.40. példa IDB predikátumaihoz tartozó gráfot a 4.25. ábrán láthatjuk. Az első szabály feje P, és szerepel benne egy negált részcél, a Q. Ezért a gráfon van egy negatív él, a P és Q csúcsok között. A második szabály feje Q, és a P negált részcél szerepel benne. Tehát a gráf két csúcsa között van egy ellenkező irányú negatív él is. Ily módon a szabályok nem rétegezettek, hiszen a gráf tartalmaz egy negatív kört. □

4.4.5. Feladatok

4.4.1. feladat: A 4.37. példa `Eljut` relációját, a 4.38. példa `Összeköttetés` relációját, valamint a 4.39. példa `UAeljut` és `AAeljut` relációit is megváltoztathatjuk azzal, ha a 4.19. ábrában éleket törlünk ki vagy ugyanehhez az ábrához új éleket rajzolunk. Adjuk meg az új relációk értékét a következő esetekben:

* a) `CHI` és `SF` közé új, AA 1900-2100 címkével ellátott élt rajzolunk.

b) `NY` és `DEN` közé új, UA 9000-1100 címkével ellátott élt rajzolunk.

c) Az a) és b) pontban megadott mindkét élt felrajzoljuk.

d) A `DEN` és `DAL` közötti élt kitöröljük.

4.4.2. feladat: Írjuk fel a 4.33. példában bevezetett filmsorozat fogalmának következő módosításait Datalog szabályok segítségével. Ha szükség van negációra, használjunk rétegzett negációt. A szabályok felírásához a `Folytatása` nevű EDB relációt és a 4.36. példában definiált `Sorozat` relációt használhatjuk.

* a) `P(x, y)`; azon (x, y) filmpárokat jelenti, amelyekre igaz, hogy az y film tagja az x filmből kiindult sorozatnak, de nem közvetlen folytatása az x filmnek.

b) `Q(x, y)`; azon (x, y) filmpárokat jelenti, amelyekre igaz, hogy az y film tagja az x filmből kiindult sorozatnak, de nem közvetlen folytatása sem az x filmnek, sem pedig az x film egyetlen közvetlen folytatásának sem.

! c) `R(x)`; azon x filmeket jelenti, amelyeknek van még legalább két folytatása. Lehet, hogy mindkettő közvetlen folytatás, de az is lehet, hogy csak az egyik közvetlen folytatás, a másik ennek a közvetlen folytatásnak a közvetlen folytatása.

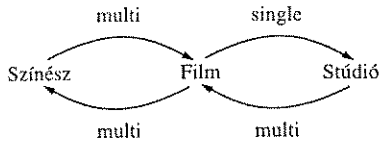
! d) `S(x, y)`, azon (x, y) filmpárokat jelenti, amelyekre igaz, hogy az y film tagja az x filmből kiindult sorozatnak, de az y filmnek legfeljebb egy folytatása van.

4.4.3. feladat: Az ODL osztályokat és a köztük levő kapcsolatokat megadhatjuk a `Rel(osztály, kosztály, mult)` reláció segítségével. A `mult` attribútum a reláció multiplicitását adja meg, értéke `multi`, ha többértékű a kapcsolat, és `single`, ha egyértékű. Az első két attribútum a kapcsolatban érintett osztályokat adja meg. A 4.26. ábra relációja a 2.6. ábrán látható, filmekkel foglalkozó állandó példánk 3 ODL osztályát tartalmazza.

osztály	*kosztály*	*mult*
`Színész`	`Film`	`multi`
`Film`	`Színész`	`multi`
`Film`	`Stúdió`	`single`
`Stúdió`	`Film`	`multi`

4.26. ábra. *ODL kapcsolatok megadása reláció segítségével*

A reláció adatait megjeleníthetjük gráf formájában is. A gráf csúcsait az osztályok képezik, az érintett osztályok közötti élek pedig a `mult` attribútum értékének megfelelő címkét kapják. A 4.27. ábrán a 4.26. ábra relációjának megfelelő gráfot láthatjuk.



4.27. ábra. *A kapcsolatok gráffal történő felírása*

A következő predikátumok mindegyikéhez írjuk fel a megfelelő Datalog szabályokat. Ha szükség van negációra, használjunk rétegzett negációt. A szabályok felírásához a `Rel` nevű EDB relációt használhatjuk. Végezzük el a felírt szabályok 4.26. ábrán látható adatokra történő kiértékelését is.

a) `P(osztály, bosztály)`; azt jelenti, hogy a gráfban létezik út[6] `osztály` és `bosztály` között. A `bosztály` ezáltal be van ágyazva az `osztály`-ba, hiszen tagja egy olyan objektumnak, ami tagja egy olyan objektumnak, ..., ami tagja az első osztály egy objektumának.

[6] Az üres utakat ebben a feladatban nem vesszük figyelembe.

*! b) `S(osztály, bosztály)` és `M(osztály, bosztály)`; az első azt jelenti, hogy `osztály` és `bosztály` között „egyértékű beágyazásról" beszélünk, azaz az `osztály` és `bosztály` közötti út mindegyik éle a `single` szócskával van megcímkézve. A második első azt jelenti, hogy `osztály` és `bosztály` között „többértékű beágyazásról" beszélünk, azaz az `osztály` és `bosztály` közötti út legalább egy éle a `multi` szócskával van megcímkézve.

c) `Q(osztály, bosztály)`; azt jelenti, hogy a gráfban létezik út `osztály` és `bosztály` között, de nem létezik egyértékű út. Használhatjuk az előző pontokban definiált IDB predikátumokat.

4.5. Relációkra vonatkozó megszorítások

A relációs modell rendelkezik olyan eszközzel, amellyel kifejezhetők az olyan gyakori megszorítások, mint például a 2.5. alfejezetben bevezetett hivatkozási épség. Sőt, egészen sok fajta megszorítás kényelmesen kifejezhető a relációs algebra segítségével, mint amint azt majd látni fogjuk. A 4.44. példával bemutatjuk, hogy még a funkcionális függőségek is kifejezhetők ily módon. A megszorítások meglehetősen fontosak az adatbázisok világában. A 6. fejezet bemutatja, hogy az SQL alapú adatbázisrendszerek hogyan valósítják meg ezeket a relációs algebrában kifejezhető megszorításokat.

4.5.1. Megszorítások megadása relációs algebra segítségével

Megszorításokat két módon fejezhetünk ki relációs algebrai kifejezésekkel:

1. Ha R egy relációs algebrai kifejezés, akkor az $R = \emptyset$ egy olyan megszorítás, amelynek jelentése „az R-nek üresnek kell lennie" vagy másképpen fogalmazva „az R eredményében egyetlen sor sincs".

2. Ha R és S relációs algebrai kifejezések, akkor $R \subseteq S$ egy olyan megszorítás, melynek jelentése: „az R eredményének minden sora benne kell legyen az S eredményében". Természetesen, az S eredménye tartalmazhat az R sorain kívül más sorokat is.

A megszorítások megadásának ez a két módja a kifejezőerő szempontjából ekvivalens egymással, viszont bizonyos esetekben az egyik forma érthetőbb vagy tömörebb. Az $R \subseteq S$ megszorítás $R - S = \emptyset$ alakban is felírható, hiszen ha R minden sora benne van S-ben, akkor biztos, hogy az $R - S$ üres. Fordítva, ha $R - S$ egyetlen sort sem tartalmaz, akkor az R minden sorának benne kell lennie S-ben, hiszen másképpen az $R - S$ eredményében lenne.

Másrészről az $R = \emptyset$ formájú megszorítások is felírhatók úgy, mint $R \subseteq \emptyset$. Technikailag a $\emptyset$ nem egy relációs algebrai kifejezés, de mivel az $R - R$ kifejezés eredménye $\emptyset$, nyugodtan használhatjuk relációs algebrai kifejezésként.

A következő alfejezetekben láthatjuk majd, hogy miként fejezhetők ki fontos megszorítások a két stílus egyikével. Amint azt a 6. fejezetben látni fogjuk, SQL-ben a megszorításokat általában az első módon, üres halmazok segítségével fejezzük ki. A megszorítást azonban nyugodtan megfogalmazhatjuk halmazok tartalmazásaként is, majd pedig az előző gondolatmenetet követve átalakíthatjuk az üres halmazt használó formára.

4.5.2. Hivatkozási épség

A megszorítások egy gyakori formája a 2.5. alfejezetben bemutatott hivatkozási épség, mely szerint ha egy érték megjelenik valahol egy környezetben, akkor ugyanez az érték egy másik, az előzővel összefüggő környezetben is megjelenik. A hivatkozási épséggel azt fejeztük ki, hogy a kapcsolatok értelmesek, azaz, ha egy A objektum (egyed) kapcsolatban áll a B objektummal (egyeddel), akkor a B-nek valóban léteznie kell. ODL kifejezésekkel például ez azt jelenti, hogy ha az A objektumban egy kapcsolatot mutató segítségével reprezentálunk, akkor ez a mutató nem lehet null és valódi objektumra kell mutatnia.

A relációs modellben a hivatkozási épség valamelyest másképpen néz ki. Ha az R reláció egy sorában szerepel egy v érték, akkor tervezési szándékaink miatt elvárhatjuk, hogy ez a v érték egy másik S reláció valamely sorának egy bizonyos komponensében is megjelenjen. A következő példában megmutatjuk, hogy miként lehetséges relációs modellben a hivatkozási épséget relációs algebra segítségével kifejezni.

4.42. példa: Vegyük a filmekről szóló szokásos példánkból a következő két reláció sémáját:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Ésszerűnek tűnik a feltételezés, miszerint mindegyik film producere szerepel a `GyártásIrányító` relációban. Ha ez nem így van, akkor ez hiba, és szeretnénk, ha a relációs adatbázisrendszerünk legalább tájékoztatna arról, hogy van olyan film az adatbázisban, amelynek produceréről nincsenek adataink.

Hogy pontosabbak legyünk, a `Film` reláció mindegyik sorának `producerAzon` komponense meg kell jelenjen a `GyártásIrányító` reláció valamely sorának `azonosító` komponensében. Mivel minden egyes gyártásirányító egyedi azonosítóval rendelkezik, ezért azt kell biztosítanunk, hogy egy film producere szerepeljen a film gyártásirányítói között. Ezt a megszorítást a következő tartalmazással fejezhetjük ki:

$$\pi_{\texttt{producerAzon}}(\texttt{Film}) \subseteq \pi_{\texttt{azonosító}}(\texttt{GyártásIrányító})$$

A bal oldali kifejezés megadja a `Film` reláció sorainak `producerAzon` komponenseiben szereplő összes azonosító halmazát. Hasonló módon, a jobb oldali kifejezés megadja a `GyártásIrányító` reláció sorainak azonosító komponenseiben szereplő

összes azonosító halmazát. A megszorítás tehát azt mondja ki, hogy ez utóbbi halmaz tartalmazza az előző halmazt.

Mellesleg, ugyanazt a megszorítást üres halmaz segítségével is kifejezhetjük:

$$\pi_{\texttt{producerAzon}}(\texttt{Film}) - \pi_{\texttt{azonosító}}(\texttt{GyártásIrányító}) = \emptyset$$

□

4.43. példa: Egynél több attribútum értékére vonatkozó hivatkozási épséget hasonló módon fejezhetünk ki. Például, biztosítani akarjuk, hogy a

```
SzerepelBenne(filmCím, év, színészNév)
```

relációban szereplő mindegyik film szerepeljen a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

relációban. A filmeket mindkét relációban a cím és a készítési év segítségével reprezentáljuk, hiszen megegyeztünk, hogy a filmek azonosítására egyik attribútum sem lenne egyedül elegendő. A következő megszorítás

$$\pi_{\texttt{filmCím,év}}(\texttt{SzerepelBenne}) \subseteq \pi_{\texttt{cím,év}}(\texttt{Film})$$

a relációk megfelelő komponenseinek levetítésével nyert cím-év párok összehasonlításával fejezi ki a hivatkozási épséget. □

4.5.3. További példák megszorításokra

A megszorításoknak ez a kifejezési formája a hivatkozási épség megadásán túl jóval többre ad lehetőséget. Így például, bármely funkcionális függőség kifejezhető relációs algebrai megszorításként, habár ez a jelölés jóval kényelmetlenebb, mint a funkcionális függőségekre eddig használt jelölés.

4.44. példa: Fejezzük ki algebrai megszorításként a

```
FilmSzínész(név, cím, nem, születésnap)
```

relációra vonatkozó

```
név → cím
```

funkcionális függőséget. Az ötletet az adja, hogy ha megszerkesztenénk a `FilmSzínész` reláció sorainak összes lehetséges $\langle t_1, t_2 \rangle$ párosítását, akkor nem létezhetne olyan pár, amelyben sorok `név` komponense megegyezik, de a `cím` komponensük eltérő. A párok megszerkesztéséhez direkt szorzatot használunk, a funkcionális függőséget megsértő sorokat kiválasztással keressük meg. Ezután az eredményt egyenlővé tesszük az üres halmazzal, és ez az egyenlőség fejezi ki a megszorítást.

Mivel egy relációnak önmagával vett szorzatát kell kiszámolnunk, ahhoz, hogy a szorzat attribútumainak különböző neve legyen, a reláció legalább egyik másolatát át kell neveznünk. A tömörség kedvéért használjuk inkább az `FSZ1` és `FSZ2` két új nevet a `FilmSzínész` relációra történő hivatkozáskor. A funkcionális függőséget a következő algebrai megszorítással fejezhetjük ki:

$$\sigma_{\texttt{FSZ1.név = FSZ2.név AND FSZ1.cím} \neq \texttt{FSZ2.cím}}(\texttt{FSZ1} \times \texttt{FSZ2}) = \emptyset$$

A fenti `FSZ1` × `FSZ2` szorzatban az `FSZ1` a következő átnevezés rövidítése:

$$\rho_{\texttt{FSZ1}(\textit{név, cím, nem, születésnap})}(\texttt{FilmSzínész})$$

Az `FSZ2` a `FilmSzínész` hasonló átnevezése. □

A megszorítások egy másik típusát a tartományra vonatkozó megszorítások képezik. A tartományra vonatkozó megszorítások gyakran egyszerűen egy attribútum értékeinek típusára vonatkoznak. Például, hogy az attribútum típusa legyen egész szám, vagy 30 hosszúságú karakterlánc. Az ilyen megszorításokat nem tudjuk kifejezni relációs algebrában, hiszen az olyan típusok, mint például az egész szám, nincsenek benne ebben az algebrában. Azonban a tartományra vonatkozó megszorítások gyakran az attribútumok értékeire vonatkoznak. Ha a megengedett értékek halmaza kifejezhető egy kiválasztás feltételében, akkor az ilyen megszorításokat ki tudjuk fejezni az algebrai megszorítások nyelvén.

4.45. példa: Tegyük fel, hogy a `FilmSzínész` reláció `nem` attribútumának megengedett értékei a `'N'` és a `'F'`. Ezt a megszorítást algebrailag a következőképpen fejezhetjük ki:

$$\sigma_{\texttt{nem} \neq \texttt{'N' AND nem} \neq \texttt{'F'}}(\texttt{FilmSzínész}) = \emptyset$$

Ez azt jeleni, hogy a `FilmSzínész` reláció azon sorainak halmaza, amelyekben a `nem` komponens értéke sem nem `'N'`, sem nem `'F'`, üres. □

Végül pedig nézzük meg, mi a helyzet azokkal a megszorításokkal, amelyek nem tartoznak a 2.5. alfejezetben felsorolt kategóriák egyikébe sem. Az algebrai megszorítások nyelvén sok új típusú megszorítás is kifejezhető. Mi most egyetlen ilyen példát mutatunk be.

4.46. példa: Tegyük fel, hogy valaki csak akkor lehet egy filmstúdió elnöke, ha a nettó bevétele legalább 10 000 000 $. Ez a megszorítás nem hivatkozási épséget fejez ki és nem is az attribútumok értékére vagy típusára vonatkozik, mégis kifejezhető algebrában. Szükségünk van a következő két reláció théta-összekapcsolására:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

A théta-összekapcsolás feltétele az, hogy a `Stúdió` reláció `elnökAzon` komponense egyenlő legyen a `GyártásIrányító` reláció `azonosító` komponensével. Az összekapcsolással olyan sorpárokat kapunk, amelyben a gyártásirányító egyúttal a stúdió elnöke. Ha ebből a relációból kiválasztjuk azokat a sorokat, ahol a nettó bevétel kisebb, mint tízmillió, akkor olyan halmazt kapunk, ami a megszorításunk értelmében üres. Tehát a megszorítást a következőképpen fejezhetjük ki:

$$\sigma_{\texttt{nettóBevétel<10000000}}(\texttt{Stúdió} \bowtie_{\texttt{elnökAzon=azonosító}} \texttt{GyártásIrányító}) = \emptyset$$

Ennek a megszorításnak a kifejezésére egy másik lehetőség az, ha a stúdiók elnökeinek azonosítóiból kapott halmazt hasonlítjuk össze azon gyártásirányítók azonosítóinak halmazával, akiknek a nettó bevétele legalább 10 000 000 $. Ez előző részhalmaza kell legyen az utóbbinak. A következő tartalmazás éppen ezt fejezi ki:

$$\pi_{\texttt{elnökAzon}}(\texttt{Stúdió}) \subseteq \pi_{\texttt{azonosító}}(\sigma_{\texttt{nettóBevétel}\geq\texttt{10000000}}(\texttt{GyártásIrányító}))$$

□

4.5.4. Feladatok

4.5.1. feladat: Írjuk fel a következő megszorításokat a 4.1.1. feladat relációira:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez,
    képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A megszorításokat tartalmazással és üres halmaz segítségével is kifejezhetjük. A 4.1.1. feladat adatait használva, valamennyi megszorítás esetén mutassuk meg azokat a sorokat, amelyek megsértik az adott megszorítást.

* a) Az olyan PC-ket, amelyek processzorának sebessége kisebb, mint 150, nem árulhatják drágábban, mint 1500 $.

b) A 11 hüvelyknél kisebb képernyőjű laptopoknak a merevlemeze legalább 1 gigabájt kell legyen, ellenkező esetben csak 2000 $-nál olcsóbban árulhatják.

! c) A PC-gyártók nem gyárthatnak laptopokat.

*!! d) Ha egy gyártó készít PC-t, akkor készítenie kell olyan laptopot is, amelynek a sebessége legalább akkora, mint a PC sebessége.

! d) Ha egy laptopnak nagyobb memóriája van, mint egy PC-nek, akkor a laptop drágább kell legyen, mint a PC.

4.5.2. feladat: Fejezzük ki relációs algebrában a következő megszorításokat. A megszorítások a 4.1.3. feladat relációira vonatkoznak.

```
Hajóosztályok(hajóosztály, típus, ország, ágyúkSzáma,
kaliber)
Hajók(név, hajóosztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, állapot)
```

A megszorításokat tartalmazással vagy üres halmaz segítségével is kifejezhetjük. A 4.1.3. feladat adatait használva, valamennyi megszorítás esetén mutassuk meg azokat a sorokat, amelyek megsértik az adott megszorítást.

a) Egyik hajóosztály ágyúinak kalibere sem lehet nagyobb, mint 16 hüvelyk.

b) Ha egy hajóosztály ágyúinak száma több mint 9, akkor a kaliberük nem lehet nagyobb 14 hüvelyknél.

! c) A hajóosztályok nem tartalmazhatnak 2-nél több hajót.

! d) Egy országnak sem lehet csatahajója és cirkálóhajója is egyszerre.

!! e) A több mint 9 ágyúval rendelkező hajó nem csatázhat olyan 9 ágyúnál kevesebbel rendelkező hajóval, amelyik elsüllyedt.

4.5.3. feladat: A megszorításokat éppúgy kifejezhetjük Datalogban, mint relációs algebrában. Egy vagy több szabály segítségével felírjuk azt az IDB predikátumot, amelynek értéke a megszorítás értelmében üres halmaz. Írjuk fel Datalogban a következő megszorításokat:

* a) A 4.42. példában szereplő megszorítás.

b) A 4.43. példában szereplő megszorítás.

c) A 4.44. példában szereplő megszorítás.

d) A 4.45. példában szereplő megszorítás.

e) A 4.46. példában szereplő megszorítás.

! **4.5.4. feladat:** Tegyük fel, hogy van két relációnk, az R és az S. Legyen a C egy olyan hivatkozási épség, amely azt állítja, hogy ha az R relációban szerepel egy olyan sor, amelynek az $A_1, A_2, \ldots, A_n$ attribútumokon vett értéke rendre $v_1, v_2, \ldots, v_n$, akkor az S relációban szerepelnie kell egy olyan sornak, amelynek a $B_1, B_2, \ldots, B_n$ attribútumokon vett értéke rendre $v_1, v_2, \ldots, v_n$. Írjuk fel ezt a megszorítást relációs algebrában.

!! **4.5.5. feladat:** Tekintsük az R relációt és az R attribútumain értelmezett funkcionális függőséget: $A_1, A_2, \ldots, A_n \rightarrow B$. Írjuk fel relációs algebrában azt a megszorítást, amely azt fejezi ki, hogy az R teljesíti ezt a funkcionális függőséget.

4.6. Multihalmazokon értelmezett relációs műveletek

Adatbázisokban az adatok egy természetes megjelenítési módja az, hogy a relációkat sorok halmazaként kezeljük. A kereskedelmi adatbázisok viszont ritkán alapulnak tisztán halmazokon, hiszen bizonyos esetekben a relációknál megengedett az azonos sorok jelenléte is. Ne feledjük, hogy azt a „halmazt", amelyben ugyanaz az elem többször is szerepelhet, multihalmaznak nevezzük. Ebben az alfejezetben olyan relációkkal foglalkozunk, amelyek nem igazi halmazok, hanem multihalmazok. Éppen ezért ugyanaz a sor többször is megjelenhet egy adott relációban. Az olyan relációt, amelyben nem megengedett az egyforma sorok jelenléte, „halmaznak" nevezzük, azt a relációt viszont, amelyben az egyforma sorok jelenléte is lehetséges, „multihalmaznak" nevezzük.

4.47. példa: A 4.28. ábrán látható reláció tulajdonképpen sorokból álló multihalmaz, amelyben az (1, 2) sor háromszor és a (3, 4) sor egyszer szerepel. Ha a 4.28. ábra relációját halmaznak tekintenénk, akkor ki kellene küszöbölni az (1, 2) sor két megjelenését. A multihalmazként kezelt relációban megengedjük ugyan, hogy ugyanazon sor többször szerepeljen, viszont a sorok sorrendje itt sem számít, éppúgy, mint a halmazként kezelt relációk esetén. □

4.6.1. Mire jók a multihalmazok?

A relációk hatékony megvalósításának szempontjából a relációk multihalmazokként való kezelése több módon is gyorsíthatja a relációs műveleteket. Például, ha vetíteni akarunk és az eredményrelációt multihalmaznak tekintjük, akkor függetlenül dolgozhatunk minden egyes sorral. Ha az eredményt halmazként kezeljük, akkor minden egyes sornál a kívánt vetítés eredményét össze kell hasonlítanunk az eredményül kapott összes többi sorral, azért, hogy biztosak legyünk benne, hogy a kapott sor még nem szerepel az eredményben. Azonban, ha az eredményt multihalmazként kezeljük, akkor minden egyes sorra elvégezzük a vetítést és az eredménysort egyszerűen csak hozzáadjuk az eredményhez, anélkül hogy összehasonlítanánk az eredmény többi sorával.

4.48. példa: A 4.28. ábrán látható multihalmaz akár a 4.29. ábrán látható reláció A és B attribútumokra történő vetítésének eredménye is lehetne, feltéve, ha megengednénk, hogy az (1, 2) sor többször is szerepeljen az eredményben, azaz az eredményt multihalmazként kezelnénk. Ha a hagyományos relációs algebrai vetítési operátort használnánk, és ily módon kiküszöbölnénk az azonos sorokat az eredményből, akkor a következő relációt kapnánk:

A	B
1	2
3	4

Figyeljük meg, hogy annak ellenére, hogy a multihalmazként kezelt eredmény több sort tartalmaz, mégis gyorsabban ki lehet számolni, hiszen multihalmazok esetén az egyes lépésekben megkapott sorokat nem kell összehasonlítani az előzőleg már megkapott sorokkal.

Továbbá, ha azért vetítünk le egy relációt valamely attribútumára, mert valamilyen összesítést szeretnénk végezni (lásd 5.5. alfejezet), akkor a levetített relációt nem kezelhetjük halmazként. Ha például szeretnénk kiszámolni a 4.29. ábrán az A attribútum értékeinek átlagát és a vetítés eredményét halmaznak tekintjük, akkor az A átlagos értéke 2, hiszen a 4.29. ábrán az A attribútumnak két különböző értéke van, az 1 és a 3. Viszont ha a 4.29. ábrán az A oszlopot az {1, 3, 1, 1, 1} mulithalmaznak tekintjük, akkor megkapjuk az A értékeinek helyes átlagát, ami 1,5. □

Két reláció egyesítésekor szintén időt takaríthatunk meg, ha az eredményt multihalmazként kezeljük. Ha az $R \cup S$ kiszámítását hagyományosan végezzük, akkor az S valamennyi sorára meg kell nézni, hogy nincs-e benne az R relációban. A vizsgált sor csak akkor kerül be az egyesítés eredményébe, ha nincs benne R-ben. Viszont ha elfogadjuk, hogy az eredmény egy multihalmaz, akkor egyszerűen csak be kell másolni az R és az S valamennyi sorát az eredménybe, függetlenül attól, hogy az adott sor esetleg mindkét relációban szerepel.

A	B
1	2
3	4
1	2
1	2

4.28. ábra. *Egy multihalmaz*

A	B	C
1	2	5
3	4	6
1	2	7
1	2	8

4.29. ábra. *A 4.48. példához tartozó multihalmaz*

4.6.2. Multihalmazok egyesítése, metszete, különbsége

Multihalmazok egyesítésekor mindegyik sor annyiszor szerepel az egyesítés eredményében, ahányszor a két multihalmazban összesen szerepelt. Azaz, ha az R multihalmazban a t sor n-szer szerepel, és az S multihalmazban a t sor m-szer szerepel, akkor az $R \cup S$ multihalmazban a t sor $(n + m)$-szer kell szerepeljen. Megjegyzendő, hogy akár az n, akár az m, vagy akár mindkét érték lehet 0 is.

Ha az R és S multihalmazok metszetét számoljuk, és a t sor n-szer szerepel az R-ben és m-szer az S-ben, akkor a t sor $\min(n, m)$-szer szerepel az $R \cap S$ eredményben. Ha az $R - S$ különbséget számoljuk, akkor a t sor $\max(0, n - m)$-szer szerepel majd az eredményben. Azaz, ha a t sor többször szerepel az R-ben mint az S-ben, akkor az $R - S$ különbségben a t sor annyiszor fog szerepelni, ahányszor az R-ben szerepel mínusz ahányszor az S-ben szerepel. Tehát, ha a t sor legalább annyiszor szerepel az S-ben, mint ahányszor az R-ben, akkor a t nem jelenik meg az $R - S$ különbségben. Ezt úgy is tekinthetjük, hogy a t minden egyes előfordulása az S-ben „semlegesíti" a t egy előfordulását az R-ben.

Multihalmaz-műveletek halmazokon

Képzeljük el, hogy van két halmazunk, az R és az S. Bármely halmaz tekinthető multihalmaznak, olyan multihalmaznak, amelynek történetesen mindegyik sora különböző. Tegyük fel, hogy az $R \cap S$ metszetet akarjuk kiszámolni, de R-et és S-t multihalmaznak tekintjük és az ennek megfelelő szabályt alkalmazzuk. Ebben az esetben ugyanazt az eredményt kapjuk, mintha az R-et és S-t halmaznak tekintettük volna. Ha R-et és S-t multihalmaznak tekintjük, akkor egy t sor annyiszor szerepel majd az $R \cap S$ eredményben, amennyi a két multihalmazban található előfordulások minimuma. Mivel az R és S halmazok, ezért a t előfordulási száma 0 vagy 1. Azaz, mindegy, hogy halmaz vagy multihalmaz szabállyal számoljuk ki a metszetet, a t sor legfeljebb egyszer szerepelhet az $R \cap S$ eredményben, és pontosan egyszer akkor fog megjelenni, ha mind az R-ben, mind az S-ben szerepel a t sor. Hasonló módon, ha az $R - S$ vagy az $S - R$ kiszámításához multihalmaz-különbséget használunk, akkor ugyanazt az eredményt kapjuk, mintha halmazkülönbséget használtunk volna.

Az egyesítés azonban különböző eredményt ad, attól függően, hogy az R-et és az S-t halmaznak vagy multihalmaznak tekintjük. Ha multihalmaz szabályt alkalmazunk az $R \cup S$ kiszámításához, akkor előfordulhat, hogy az eredmény nem halmaz lesz, annak ellenére, hogy mind az R, mind az S halmaz. Azaz, ha a t sor az R-ben és az S-ben is szerepel, akkor multihalmaz szabályt alkalmazva a t sor kétszer jelenik meg az $R \cup S$ eredményben, ha viszont halmaz szabályt alkalmazunk, akkor a t sor csak egyszerre jelenik meg az $R \cup S$ eredményben. Tehát, egyesítéskor különösen nagy figyelemmel kell megadnunk, hogy melyik szabályt kell alkalmazni, a multihalmaz szabályt, avagy a halmaz szabályt.

4.49. példa: Legyen az R a 4.28. ábrán látható reláció, ami valójában egy multihalmaz, amelyben az (1, 2) sor kétszer és a (3, 4) sor egyszer szerepel. Legyen az S a következő multihalmaz:

A	B
1	2
3	4
3	4
5	6

Ebben az esetben az $R \cup S$ egyesítés eredménye az a multihalmaz, amelyben az (1, 2) négyszer (háromszor az r miatt és egyszer az S miatt), a (3, 4) háromszor és az (5, 6) egyszer szerepel.

Az $R \cap S$ metszet eredménye a következő multihalmaz:

A	B
1	2
3	4

Mind az (1, 2), mind a (3, 4) sor csak egyszer szerepel az $R \cap S$ eredményben. Az (1, 2) háromszor szerepel az R-ben és egyszer az S-ben és $\min(1, 3) = 1$. Hasonló mó-

Algebrai azonosságok multihalmazok esetén

Az algebrai azonosság két relációs algebrai kifejezés közötti ekvivalencia. A kifejezések argumentumai relációk. Az ekvivalencia szerint mindegy, hogy milyen relációkat helyettesítünk be, a két kifejezés ugyanazt az eredményt adja. Egy jól ismert példa az egyesítésre vonatkozó azonosság: $R \cup S \equiv S \cup R$. Ez az azonosság fennáll, ha az R és S relációkat halmazként kezeljük, de akkor is fennáll, ha a relációkat multihalmazként kezeljük. Létezik azonban számos olyan azonosság, amely fennáll abban az esetben, ha a relációs algebrai műveleteket a hagyományos halmazelméleti módon értelmezzük, viszont nem érvényesek ha a relációkat multihalmazként kezeljük. Ilyen azonosság például a különbség és az egyesítés disztributív volta: $(R \cup S) - T \equiv (R - T) \cup (S - T)$. Ez az azonosság érvényes halmazokra, viszont nem érvényes multihalmazokra. Azért, hogy lássuk, miért is nem igaz multihalmazok esetén, tegyük fel, hogy a t sor egyszer szerepel mindhárom relációban, az R-ben, az S-ben és a T-ben. Ebben az esetben a bal oldali kifejezés eredményében egyszer szerepel a t sor, a jobb oldali kifejezés eredményében viszont nem szerepel a t sor. Ha halmaz szabályt alkalmazunk, akkor egyik oldal eredményében sem szerepel a t sor. A 4.6.4 és a 4.6.5. feladatokban megvizsgáljuk további algebrai azonosságok érvényességét multihalmazok esetén.

don a (3, 4) sorra min(1, 2) = 1. Az (5, 6) sor egyszer szerepel az S-ben és nem szerepel az R-ben, min(0, 1) = 0, tehát az $R \cap S$ eredményben ez a sor nem szerepel.

Az $R - S$ különbség eredménye a következő multihalmaz:

A	B
1	2
1	2

Az (1, 2) háromszor szerepel az R-ben és egyszer az S-ben, max(0, 3 – 1) = 2, ezért az $R - S$ különbségben ez a sor kétszer jelenik meg. A (3, 4) sor egyszer szerepel az R-ben és kétszer szerepel az S-ben, max(0, 1 – 2) = 0, tehát ez a sor nem szerepel az $R - S$ különbségben. Mivel az R-nek nincs más sora, ezért az $R - S$ eredmény sem tartalmazhat más sort.

A $S - R$ különbség a következő multihalmaz:

A	B
3	4
5	6

A (3, 4) sor egyszer jelenik meg, hiszen ki kell vonni az S-beli előfordulások számából az R-beli előfordulások számát. Az (5, 6) sor ugyanilyen okból szerepel egyszer az eredményben. Ebben az esetben a multihalmazként kezelt eredmény történetesen halmaz. □

4.6.3. Multihalmazok vetítése

A multihalmazok vetítéséről már volt szó. Amint a 4.48. példában láthattuk, a vetítés során mindegyik sort külön kezeljük. Ha az R reláció a 4.29. ábrán látható multihalmaz, akkor a $\pi_{A, B}(R)$ eredménye a 4.28. ábrán látható multihalmaz.

Ha vetítés során különböző sorokból egy vagy több attribútum elhagyásával egyforma sorokat kapunk, akkor ezeket az egyforma sorokat nem kell kiküszöbölni a multihalmaz vetítésének eredményéből. Ha a 4.29. ábra R relációjának (1, 2, 5), (1, 2, 7) és (1, 2, 8) sorait levetítjük az A és B attribútumokra, akkor mindhárom sor ugyanazt az (1, 2) sort eredményezi. Ez azt jelenti, hogy a multihalmaz-eredményben ez a sor háromszor szerepel, viszont a hagyományos vetítéssel csak egyszer szerepelne.

4.6.4. Multihalmazokon értelmezett kiválasztás

A multihalmazon végzett kiválasztás azt jelenti, hogy a kiválasztás feltételét minden egyes sorra alkalmazzuk. A multihalmazoknál megszokott módon, itt sem küszöböljük ki az azonos sorokat.

4.50. példa: Legyen az R a következő reláció:

A	B	C
0	2	5
3	4	6
1	2	7
1	2	7

A $\sigma_{C \geq 6}(R)$ multihalmazos kiválasztás eredménye:

A	B	C
0	4	6
1	2	7
1	2	7

Azaz, az első sor kivételével mindegyik sor teljesíti a feltételt. A két utolsó sor az R-ben is megegyezik, így mindkettő szerepel az eredményben is. □

4.6.5. Multihalmazok szorzata

A multihalmazok Descartes-szorzatára vonatkozó szabály pontosan az, amit vártunk: az első reláció minden egyes sorát össze kell párosítanunk a második reláció mindegyik sorával, függetlenül attól, hogy az adott sor hányszor szerepel az adott relációban. Következésképpen, ha az r sor m-szer szerepel az R relációban és az s sor n-szer szerepel az S relációban, akkor az rs sor (mn)-szer szerepel a szorzat eredményében.

4.51. példa: Legyen az R és S reláció a 4.30 ábrán látható két reláció. Ekkor az $R \times S$ szorzat hat sorból áll, amint az a 4.30.c) ábrán is látható. Megjegyezzük, hogy a hagyományos szorzatnál bevezetett, attribútum nevekre vonatkozó jelölés multihalmazokra is kitűnően alkalmazható. Ekképpen az R és S relációkban egyaránt megtalálható B attribútum kétszer jelenik meg a szorzatban, és előtagként a megfelelő reláció neve szolgál. □

A	B
1	2
1	2

a) Az R reláció

B	C
2	3
4	5
4	5

b) Az S reláció

A	$R.B$	$S.B$	C
1	2	2	3
1	2	2	3
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

c) A $R \times S$ szorzat

4.30. ábra. *Multihalmazok szorzatának kiszámítása*

4.6.6. Multihalmazok összekapcsolása

A multihalmazok összekapcsolása szintén nem szolgál különösebb meglepetéssel. Az első reláció minden egyes sorát összehasonlítjuk a második reláció valamennyi sorával, és eldöntjük, hogy az így kapott sorpárok sikeresen összekapcsolhatók-e vagy sem. Ha az összekapcsolás elvégezhető, akkor az eredményül kapott sort betesszük az eredménybe. Az eredményből nem kell kiküszöbölni a megegyező sorokat.

4.52. példa: A 4.30. ábrán látható R és S reláció természetes összekapcsolásának eredménye, $R \bowtie S$:

A	B	C
1	2	3
1	2	3

Az R reláció (1, 2) sora összekapcsolható az S(2, 3) sorával. Mivel az R relációban az (1, 2) sor kétszer szerepel és az S relációban a (2, 3) sor egyszer szerepel, ezért az eredményben az (1, 2, 3) kétszer jelenik meg. Az R és S egyéb sorai nem kapcsolhatók össze.

Ugyanezeken a relációkon végezzünk most egy théta-összekapcsolást:

$$R \bowtie_{R.B < S.B} S$$

Az eredményül kapott multihalmaz:

A	$R.B$	$S.B$	C
1	2	4	5
1	2	4	5
1	2	4	5
1	2	4	5

Az összekapcsolást a következő módon végezzük: az R(1, 2) sora és az S(4, 5) sora megfelel a feltételnek. Mivel mindkét sor kétszer szerepel a megfelelő relációkban, ezért az összekapcsolt sor $2 \times 2 = 4$ alkalommal jelenik meg az eredményben. Ezen kívül még összekapcsolhatnánk az R(1, 2) sorát az S(2, 3) sorával, de ez a sorpár nem teljesíti a feltételt, ily módon az eredményben sem jelenik meg. □

4.6.7. Datalog szabályok alkalmazása multihalmazokra

A kiválasztásra, vetítésre és összekapcsolásra vonatkozó módszerek a Datalog szabályokra is érvényesek, feltéve, hogy nem negált, relációs részcélokkal dolgozunk. Nagy vonalakban szólva vesszük a különböző részcéloknak megfelelő relációk összekapcsolását, az eredményre alkalmazzuk az aritmetikai részcéloknak megfelelő kiválasztást, és a kapott relációt végül levetítjük a fejben szereplő attribútumokra. Minden egyes lépésnél a multihalmazoknak megfelelő algoritmust alkalmazzuk.

A Datalog szabályok kiértékeléséhez egyszerűbb a 4.2.4. alfejezetben megadott megközelítést alkalmazni, az ún. második megközelítést. Emlékezzünk vissza, hogy ezzel a módszerrel mindegyik nem negált relációs részcélba behelyettesítjük a részcél predikátumának megfelelő reláció összes sorát. Ha az összes részcélba történő behelyettesítés a változók konzisztens értékeit eredményezte, és az aritmetikai részcélok is igazak[7], akkor megnézzük, hogy ez a megfeleltetés a fejben szereplő változókhoz milyen értékeket rendel. A kapott sort betesszük a fejnek megfelelő relációba.

Mivel most multihalmazokkal dolgozunk, ezért a fejben nem kell kiküszöbölnünk az azonos sorokat. Továbbá, mivel a részcélokhoz a sorok összes lehetséges kombinációit hozzárendeljük, ezért ha egy sor n-szer jelenik meg valamely részcélhoz tartozó relációban, akkor ezt a sort n-szer rendeljük hozzá az adott részcélhoz, mialatt a többi részcélhoz a többi lehetséges sorkombinációt rendeljük hozzá.

4.53. példa: Tekintsük a következő szabályt:

```
H(x,z) ← R(x,y) AND S(y,z)
```

Legyen az R és az S a 4.30. ábrán látható két reláció. A részcélokhoz egyetlen módon tudunk konzisztens módon sorokat megfeleltetni (úgy, hogy a két részcél y értéke megegyezzen), amikor az első részcélnak az (1, 2) sort feleltetjük meg, a második részcélnak pedig a (2, 3) sort. Mivel az (1, 2) sor kétszer jelenik meg az R-ben, a (2, 3) pedig egyszer az S-ben, ezért két olyan megfeleltetés van, ahol a változók értéke a következő: $x = 1$, $y = 2$ és $z = 3$. A fejhez tartozó (x, z) sor értéke mindkét alkalommal (1, 3). Ezért az (1, 3) sor kétszer jelenik meg a fejhez tartozó H relációban, és a H relációnak nincs is más sora. Tehát a szabály által definiált reláció:

$H1$	$H2$
1	3
1	3

A reláció attribútumait átneveztük, az új nevük $H1$ és $H2$. Általánosan fogalmazva, ha az (1, 2) sor n-szer jelenne meg az R-ben, a (2, 3) sor pedig m-szer az S-ben, akkor az (1, 3) sor (nm)-szer jelenne meg a H-ban. □

Ha egy relációt több szabállyal definiálunk, akkor az egyes szabályok eredménysorainak multihalmazos egyesítésével kapjuk meg a végeredményt.

4.54. példa: Tekintsük a következő két szabály által definiált H relációt:

---

[7] Meg kell jegyeznünk, hogy ez csak akkor helytálló, ha a szabály nem tartalmaz negált részcélt. A multihalmazos modellben még nincs letisztult fogalom a negációt, relációs részcélokat tartalmazó Datalog szabályok jelentését illetően.

```
H(x,y) ← S(x,y) AND x>1
H(x,y) ← S(x,y) AND y<5
```

Tegyük fel, hogy az S reláció értéke a 4.30.b) ábrán látható érték, azaz:

B	C
2	3
4	5
4	5

Az első szabály kiértékelésekor mindhárom sor bekerül az eredménybe, hiszen mindegyiknek az első komponense nagyobb, mint 1. A második szabály által csak a (2, 3) sor kerül be a H-ba, mert a (4, 5) sor nem teljesíti az $y<5$ feltételt. Tehát a H reláció végeredményben a (2, 3) sor két másolatát és a (4, 5) sor két másolatát tartalmazza. □

4.6.8. Feladatok

* **4.6.1. feladat:** Tekintsük a 4.10.a) ábrában szereplő PC relációt. Számítsuk ki a $\pi_{sebesség}$(PC) kifejezést. Mi lesz az értéke, ha a relációkat halmazokként kezeljük? És ha multihalmazokként? Mennyi lesz az értékek átlaga ebben a projekcióban, ha a relációkat halmazokként kezeljük? Mennyi, ha multihalmazokként?

4.6.2. feladat: Végezzük el a 4.6.1. feladatot a $\pi_{merevlemez}$(PC) kifejezéssel.

4.6.3. feladat: Tekintsük a 4.1.3. feladatban szereplő „Csatahajó" relációt.

a) Tekintsük a $\pi_{kaliber}$(Hajóosztályok) egyoszlopos relációt adó kifejezést, amely az egyes osztályok kaliberét adja meg. Mi lesz ennek az értéke a 4.1.3. feladat adataival, ha a relációkat halmazoknak tekintjük? Mi, ha multihalmazoknak?

! b) Készítsünk egy relációs algebrai kifejezést, amely az egyes hajók kaliberét adja (nem az egyes osztályokét). A kifejezéshez használjunk multihalmazokat, azaz annyiszor szerepeljen a b, mint kaliber, ahány b kaliberű hajó van.

! **4.6.4. feladat:** Bizonyos algebrai azonosságok, amelyek érvényesek a halmazokként kezelt relációkra, érvényesek a multihalmazokként kezelt relációkra is. Mutassuk meg, miért maradnak érvényben a következő azonosságok multihalmazokra is:

* a) Az egyesítés asszociatív tulajdonsága: $(R \cup S) \cup T \equiv R \cup (S \cup T)$

b) A metszet asszociatív tulajdonsága: $(R \cap S) \cap T \equiv R \cap (S \cap T)$

c) A természetes összekapcsolás asszociatív tulajdonsága: $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$

d) Az egyesítés kommutatív tulajdonsága: $(R \cup S) \equiv (S \cup R)$

e) A metszet kommutatív tulajdonsága: $(R \cap S) \equiv (S \cap R)$

f) Az egyesítés kommutatív tulajdonsága: $(R \bowtie S) \equiv (S \bowtie R)$

g) $\pi_L(R \cup S) \equiv \pi_L(R) \cup \pi_L(S)$, ahol L az attribútumok tetszőleges listája

* h) Az egyesítés és a metszet disztributív tulajdonsága:
$R \cup (S \cap T) \equiv (R \cup S) \cap (R \cup T)$

i) $\sigma_{C \text{ AND } D}(R) \equiv \sigma_C(R) \cap \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira.

!! **4.6.5. feladat:** A következő azonosságok érvényesek, ha a relációkat halmazokként kezeljük, de nem érvényesek, ha multihalmazokként. Mutassuk meg, hogy halmazokra miért érvényesek, és adjunk ellenpéldát multihalmazokra.

* a) $(R \cap S) - T \equiv R \cap (S - T)$

b) A metszet és az egyesítés disztributív tulajdonsága:
$R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$

c) $\sigma_{C \text{ OR } D}(R) \equiv \sigma_C(R) \cup \sigma_D(R)$, ahol C és D tetszőleges feltételek az R soraira

4.7. A relációs modell további kiterjesztései

Léteznek további fogalmak és műveletek, amelyek ugyan nem részei a relációs adatmodell formális leírásának, de mégis előfordulnak a gyakorlatban használt lekérdezőnyelvekben. Ebben a fejezetben olyan műveletekkel fogunk foglalkozni, amelyek módosítják a relációkat, „összesítéseket" számítanak (mint például az oszlopokban szereplő értékek összege), vagy „nézeteket", azaz névvel ellátott függvényeket definiálnak. Ezek mind megtalálhatók az 5. fejezetben tárgyalt SQL nyelvben, valamint néhány közülük szerepel az OQL nyelvben, amelyről a 8. fejezetben lesz szó.

4.7.1. Módosítások

A relációs algebra és a Datalog egyaránt lekérdezőnyelvek abban az értelemben, hogy mindkettő segítségével kiszámolhatunk olyan új relációkat, amelyek eredménye függ más, ismert reláció értékétől. Igaz ugyan, hogy a lekérdezések megfogalmazása fontos terület, de egy olyan adatbázis, amelyet nem lehetne módosítani, semmit nem érne. Éppen ezért, mindegyik valódi adatbázisnyelv a lekérdezést éppúgy magában foglalja,

mint a módosítások lehetőségét. A következő műveletek elvégzésére legalább kell lennie parancsnak:

1. Sorok beszúrása relációba.

2. Sorok törlése relációból.

3. Meglévő sorok módosítása, egy vagy több komponensének megváltoztatásával.

4.7.2. Összesítések

A relációs algebrai műveletek az ugyanabban a relációban található többi sor értékétől függetlenül dolgoznak valamely soron. Gyakran azonban szeretnénk egy adott reláció sorait összekombinálni, azért, hogy valamilyen összesített értéket kiszámoljunk, ami nem más, mint a valamilyen szempontból összeillő sorok függvénye. Például, az állandó filmes példánkban a következőket számolhatjuk ki ily módon:

- A `Film` relációban szereplő, különböző filmek száma.
- Valamennyi stúdióhoz az általa gyártott filmek hosszúságának az összege.
- A legnagyobb nettó bevétellel rendelkező gyártásirányító.

Ezért aztán a valódi adatbázis-lekérdezőnyelvek lehetővé teszik úgynevezett *összesítő műveletek* használatát, főleg a relációk oszlopaira vonatkozó leszámlálást (count), összegzést (sum), átlagot (average), minimumot és maximumot.

4.7.3. Nézetek

Egy relációs algebrai kifejezést elképzelhetünk úgy, mint egy programot, amelyik kiszámolja az R relációt és ki is nyomtatja, vagy úgy, mint amelyiknek a visszatérési értéke az R reláció. Azonban van egy másik értelmezési módja is a relációs algebrai kifejezéseknek. Tekinthetjük őket úgy is, mint formulákat, amelyeknek addig nincs eredményük amíg igazi relációkra nem alkalmazzuk őket. Adatbázis-terminológiával élve az ilyen formulákat *nézeteknek* nevezzük. Láthatjuk majd, hogy a nézeteknek gyakran nevet adunk, és ezeket a neveket ugyanúgy használjuk más relációs algebrai kifejezések argumentumaként, mint a valódi relációkat.

A Datalog szabályok jól szemléltetik a lekérdezések és a nézetek közötti különbségeket. Emlékezzünk vissza, hogy a Datalog szabályok által definiált predikátumokat, illetve relációkat intenzionálisnak neveztük, ami arra utal, hogy ezek olyan relációk, amelyeknek nem kell extenzionálisan, azaz külső tárolókon léteziük. A nézet egyenértékű az intenzionális predikátummal. Ugyanúgy, mint ahogy az intenzionális predikátumot használhatjuk a szabályok törzsében, a nézetet is használhatjuk algebrai kifejezések argumentumaként. Másrészről, éppúgy ahogyan a Datalog szabályok kollekcióját kiértékelhetjük igazi relációkból álló adatbázison, a nézet is kiértékelhető, amikor szükség van rá.

4.7.4. Nullértékek

Gyakran előfordul, hogy egy sor valamelyik komponensének értéket kell adnunk, de nem tudjuk, hogy mi is ez az érték. Például, azt tudjuk, hogy Kevin Costner egy filmsztár, de azt nem tudjuk, hogy mikor van a születésnapja. Viszont a `FilmSzínész` reláció minden sorának van `születésnap` komponense, kérdés, hogy ilyenkor mi a teendő. Az ilyen komponensek esetén használhatunk egy speciális `NULL` értéket, amit *nullértéknek* nevezünk. A `NULL` érték bizonyos szempontból ugyanolyan érték, mint a többi. Más szempontból viszont nem is érték. Két reláció összekapcsolásakor két `NULL` komponenst nem tekintünk egyenlőnek. Például, ha két filmsztár `születésnap` komponense `NULL`, az még nem jelenti azt, hogy ugyanakkor van a születésnapjuk.

A nullértékek értelmezésére több lehetőségünk is van. Lássuk ezek közül a leggyakoribbakat:

1. *Ismeretlen érték:* „Tudom, hogy valamilyen értéknek ott kell lenni, de nem tudom, melyik ez az érték." Ilyen például az előbbiekben tárgyalt ismeretlen születésnap.

2. *Alkalmazhatatlan érték:* „Nincs olyan érték, aminek itt értelme lenne." Például, ha a `FilmSzínész` relációnak lenne egy `hitves` attribútuma is, akkor az egyedülálló színészeknél ennek az attribútumnak nullértéke lenne, hiszen nem tudhatjuk valakinek a hitvesének a nevét, ha egyszer nincs is hitvese.

3. *Visszatartott érték:* „Nem vagyunk feljogosítva rá, hogy ismerjük a megfelelő értéket." Például a `telefonszám` attribútumhoz tartozó komponenshez `NULL` értéket írunk egy titkos telefonszám esetén.

4.8. Összefoglalás

- *Relációs algebra:* Ez az algebra egy fontos lekérdezőnyelv a relációs modellben. Alapvető műveletei az egyesítés, metszet, különbség, kiválasztás, vetítés, Descartes-szorzat, természetes összekapcsolás, théta-összekapcsolás és átnevezés.

- *Datalog:* A logikának ez a formája egy másik fontos lekérdezőnyelv a relációs modellben. A Datalogban szabályokat írunk, amelyekben egy predikátumot vagy relációt részcélokból álló törzzsel definiálunk. Mind a fej, mind a részcélok atomok,

egy atom pedig nem más, mint az argumentumaira alkalmazott (esetleg negált) predikátum. Minden lekérdezés, amely kifejezhető a relációs algebrában, kifejezhető a Datalogban is.

- *Rekurzív Datalog:* A Datalog szabályok lehetnek rekurzívak, azaz egy reláció definíciójában fel lehet használni önmagát is. Egy rekurzív Datalog szabály jelentésén a legkisebb fixpontot értjük, azaz a relációt felépítő soroknak azt a legszűkebb halmazát, amelyre a fej pont megegyezik azzal, amit a törzs ad.

- *Rétegzett negáció:* Ha negációt használunk a rekurzív Datalogban, a legkisebb fixpont nem biztos, hogy egyértelmű, és néha nincs elfogadható jelentése a Datalog szabálynak. Ezért a negáció használatát a rekurzív szabályokban meg kell tiltani, így merül fel az igény a rétegzett negációra. Az ilyen típusú szabályoknak van egy vagy több legkisebb fixpontja, amely az általánosan elfogadott jelentése a szabálynak.

- *Relációk mint multihalmazok:* A kereskedelmi adatbázisrendszerekben a relációk multihalmazok, amelyekben ugyanaz a sor többször is előfordulhat. A halmazokon vett relációs algebra műveletei kiterjeszthetők multihalmazokra, de az algebrai azonosságok közül némelyik nem marad érvényben.

- *Relációk a kereskedelmi rendszerekben:* A multihalmazmodellen túl a kereskedelmi rendszerek további műveleteket is bevezetnek a relációkra, amelyek nincsenek meg a relációs algebrában vagy a Datalogban. Ezek a műveletek a sorok beszúrása, törlése vagy módosítása, összesítések számítása és nullértékek használata.

4.9. Irodalomjegyzék

A relációs algebra volt az egyik fontos összetevője a relációs modellről szóló alapvető cikknek [4]. A logikai lekérdezőnyelvek eredete kevésbé tiszta. Codd bevezetett egy elsőrendű logikát, amelyet *relációs kalkulusnak* hívott a relációs modellről szóló egyik korai cikkben [5]. A relációs kalkulus egy kifejezésekből álló nyelv, amely hasonlít a relációs algebrára, és a kifejezőereje megegyezik a relációs algebráéval, ezt [5]-ben bizonyítja is.

A Datalog leginkább olyan logikai szabályokra hasonlít, amelyeket a Prolog programozási nyelv ihletett. Mivel rekurziót is megenged, a kifejezőereje nagyobb, mint a relációs kalkulusé. A logika használata lekérdezőnyelvként leginkább [6]-ból származik, de az elméletet [2] vezette be az adatbázisrendszerekbe. A lekérdezések használatát megszorítások kifejezésére [8] vezette be.

Az ötlet, hogy a rétegzett megközelítés a megfelelő választást adja a fixpontok közül [3]-ból származik, bár ennek Datalog szabályok kiértékelésére történő használata függetlenül jelent meg [1]-ben, [7]-ben és [10]-ben. Ennél többet a rétegzett negációról, ennek kapcsolatáról a relációs algebrával, a Dataloggal és a relációs kalkulussal, valamint a Datalog szabályok kiértékeléséről negációval és anélkül [9]-ben találunk.

1. Apt, K. R., H. Blair, and A. Walker, „Towards a theory of declarative knowledge", in *Foundation of Deductive Databases and Logic Programming* (J. Minker, ed.), pp. 89–148, Morgan-Kaufmann, San Francisco, 1988.

2. Bancilhon, F. and R. Ramakrishnan, „An amateur's introduction to recursive query-processing strategies", *ACM SIGMOD Intl. Conf. On Management of Data,* pp. 16–52, 1986.

3. Chandra, A. K. and D. Harel, „Structure and complexity of relational queries", *J. Computer and System Sciences* **25**:1, pp. 99–128.

4. Codd, E. F., „A relational model for large shared data banks", *Comm. ACM* **13**:6, pp. 377–387, 1970.

5. Codd, E. F., „Relational completeness of database sublanguages", in *Database Systems* (R. Rustin, ed.), Prentice Hall, Engelwood Cliffs, NJ, 1972.

6. Gallaire, H. and J. Minker, *Logic and Databeses,* Plenum Press, New York, 1978.

7. Naqvi, S., „Negation as failure for first-order queries", *Proc. Fifth ACM Symp. On Principles of Database Systems,* pp. 114–122, 1986.

8. Nicolas, J.-M., „Logic for improving integrity checking in relational databases", *Acta Informatica* **18**:3 pp. 227–253, 1982.

9. Ullman, J. D., *Principles of Database and Knowledge-Base Systems, Volume I,* Computer Science Press, New York, 1988.

10. Van Gelder, A, „Negation as failure using tight derivations for general logic programs", in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), pp. 149–176, Morgan-Kaufmann, San Francisco, 1988.

5. fejezet

Az SQL adatbázisnyelv

A legszélesebb körben használt relációs adatbázis-kezelő rendszerek egy SQL-nek nevezett nyelv segítségével kérdezik le és módosítják az adatbázist. Az SQL a „Structured Query Language" – „Strukturált LekérdezőNyelv" rövidítése. Az SQL központi magja ekvivalens a relációs algebrával, de számos olyan lényeges része is van, mely többet nyújt mint a relációs algebra, mint például az összesítések (pl. összegek, számosságok) és az adatbázis-módosítások.

Az SQL-nek számos különböző verziója van. Először is két fő szabvány ismert: az ANSI (American National Standards Institute – Amerikai Nemzeti Szabvány Intézet) SQL és egy 1992-ben elfogadott módosított szabvány, az SQL–92 vagy SQL2. Van egy feljövőben levő szabvány is, az SQL3, mely kiterjeszti az SQL2-t számos új részszel, mint a rekurzió, a triggerek és az objektumok. Ezenkívül léteznek az SQL-nek a nagy adatbázis-kezelő rendszereket forgalmazó cégek által készített verziói is. Ezek mindegyike az SQL2-nek egy kiterjesztése, mindegyiknek megvan a maga sajátossága, és mindegyik valamilyen irányban túllép az SQL2-n, esetleg az SQL3 néhány új javaslatát megvalósítva.

Ebben és a következő két fejezetben az SQL-t mint lekérdezőnyelvet fogjuk vizsgálni. Ez a fejezet az SQL általános lekérdezőfelületére koncentrál. Tehát az SQL-t egy különálló lekérdezőnyelvnek tekintjük, melyen keresztül egy terminálnál ülve adatbázis-lekérdezéseket vagy -módosításokat hajthatunk végre. A lekérdezések eredményei a terminálunkon jelennek meg. A következő három fejezetben az SQL2 szabványnak megfelelően vizsgáljuk az SQL-t, egyrészt bemutatva azokat a jellegzetességeit, melyek a legtöbb kereskedelmi rendszerben megtalálhatóak, másrészt vizsgáljuk a korábbi ANSI szabványt is. Néhány esetben, ha az SQL2 szabvány nem fedi le megfelelően az adott témát, az SQL3 legutóbbi verzióját vesszük alapul.

A következő három fejezetben a célunk az SQL megismertetése az olvasóval, mégpedig bevezetés, mintsem kézikönyv szintjén. Így csak a leggyakrabban használt részekre koncentrálunk. Az irodalomjegyzékbeli hivatkozások olyan publikációkat tartalmaznak, ahol megtalálható a nyelv részletesebb leírása és a különböző verziók.

5.1. Egyszerű lekérdezések az SQL-ben

Az SQL legegyszerűbb lekérdezései azon sorokra vonatkoznak, melyek egy bizonyos relációban eleget tesznek egy adott feltételnek. Egy ilyen lekérdezés a relációs algebra kiválasztási műveletének felel meg. Ez az egyszerű lekérdezés, mint ahogy a legtöbb SQL lekérdezés, az SQL három alapvető kulcsszavát használja, mégpedig a: `SELECT`, `FROM` és `WHERE` kulcsszavakat.

5.1. példa: A következő kérdésekben azt a 3.9. alfejezetben leírt adatbázissémát fogjuk alkalmazni, melyet az 5.1. ábrán ismét bemutatunk. Az 5.7. alfejezetben látni fogjuk, hogyan kell az adatbázisséma információt az SQL segítségével kifejezni, egyelőre azonban tekintsük úgy, hogy a 3.9. alfejezetbeli relációkat és értékkészleteket a megfelelő SQL utasításokkal kifejeztük.

Első lekérdezésként a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

relációból kérdezzük le az összes olyan filmet, melyet a Disney-stúdió készített 1990-ben. A megfelelő SQL utasítás:

```
SELECT *
FROM Film
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Ez a lekérdezés bemutatja az SQL lekérdezések jellegzetes select-from-where alakját.

- A `FROM` záradék azon relációt vagy relációkat adja meg, melyekre a lekérdezés vonatkozik. A mi példánkban a lekérdezés a `Film` relációra vonatkozik.
- A `WHERE` záradék egy feltétel, a relációs algebra kiválasztásának megfelelően, melyet a lekérdezésnek megfelelő sorok ki kell hogy elégítsenek. A példában a feltétel az, hogy a `stúdióNév` attribútumértéke 'Disney' és az `év` attribútumértéke 1990 legyen. Azon sorok, melyek mindkét feltételt kielégítik, megfelelnek a lekérdezésnek, a többi nem.
- A `SELECT` záradék megadja a feltételeknek megfelelő sorok azon attribútumait, melyeket a lekérdezésre adott válasz tartalmazni fog. A példabeli * azt jelzi, hogy a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

5.1. ábra. *Minta adatbázisséma, ismétlés*

teljes sort tartalmazni fogja a válasz. A lekérdezés eredménye az a reláció, mely tartalmazza az összes sort, melyeket ez az eljárás előállít.

A lekérdezés feldolgozásának egyik módja az, hogy a FROM záradékbeli relációnak az összes sorát egymás után megvizsgáljuk. A WHERE záradék feltételét alkalmazzuk a sorra. Pontosabban, a WHERE záradékban szereplő attribútumokba behelyettesítjük a sor megfelelő komponenseinek értékét. A feltételt kiértékeljük, és ha igaz, akkor a SELECT-ben szereplő attribútumok által alkotott sort az eredményhez hozzávesszük. Így a lekérdezés eredményei azon Film sorok lesznek, melyeknek megfelelő filmeket a Disney gyártott 1990-ben, például az eredményben lesz a *Micsoda nő!*.

Amikor az SQL-t feldolgozó processzor a következő sort vizsgálja meg:

cím	*év*	*hossz*	*színes*	*StúdióNév*	*producerAzon*
Micsoda nő!	1990	119	True	Disney	999

(ahol 999 a producer elképzelt azonosítója), a WHERE záradék feltételében a stúdióNév attribútum értéke 'Disney', míg az év attribútum értéke 1990 lesz, mert ezek lesznek a megfelelő attribútumok értékei a szóban forgó sor esetében. Így a WHERE feltétel alakja a következő lesz:

```
WHERE 'Disney' = 'Disney' AND 1990=1990.
```

Mivel ez a feltétel nyilván igaz, a *Micsoda nő!*-re vonatkozó sor megfelel a WHERE záradék feltételeinek, így a sor az eredményhez fog tartozni. □

5.1.1. Vetítés az SQL-ben

Ha azt szeretnénk, hogy a kiválasztott sorok bizonyos komponenseit kizárjuk az eredményből, akkor levetítjük az SQL lekérdezéssel előállított relációt néhány attribútumára. A SELECT záradékban a * helyett felsorolhatjuk a FROM záradékban adott reláció bármely attribútumát. Az eredményt levetítjük a felsorolt attribútumokra.[1]

5.2. példa: Tételezzük fel, hogy az 5.1. példa lekérdezését szeretnénk úgy módosítani, hogy csak a film címét és hosszát adja vissza. A megfelelő utasítás:

```
SELECT cím, hossz
FROM Film
WHERE stúdióNév = 'Disney' AND év = 1990;
```

[1] Így a SELECT kulcsszó főleg a relációs algebra vetítés műveletének felel meg, míg az algebra kiválasztás művelete az SQL lekérdezés WHERE záradékának felel meg.

Az eredmény egy kétoszlopos tábla, a cím és hossz oszlopokkal. A tábla sorai a filmek címeiből és hosszából álló azon párosok, amelyekben a filmet a Disney készítette 1990-ben. Például a relációséma és egy sora így néz ki:

cím	*hossz*
Micsoda nő!	119

□

Néha olyan relációt szeretnénk készíteni, melyben az oszlopnevek különböznek a FROM záradékban megadott reláció attribútumneveitől. Ezért az attribútumnév után az AS kulcsszót írhatjuk, utána pedig egy *másodnév* következik, azaz egy új név, mely az eredményrelációban az oszlopnév helyett fog szerepelni. Az AS nem kötelező (az SQL néhány régebbi verziója nem is használja), azonban a másodnév rögtön azon attribútum után következik, melynek neveként fog szerepelni.

5.3. példa: Az 5.2. példát módosíthatjuk úgy, hogy olyan relációt eredményezzen, melynek attribútumai név és időtartam, cím és hossz helyett:

```
SELECT cím AS név, hossz AS időtartam
FROM Film
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Az eredmény megegyezik az 5.2. példa eredményével, de az oszlopnevek név és időtartam. Például az eredményreláció kezdődhet így:

név	*időtartam*
Micsoda nő	119

□

Egy további lehetőség a SELECT záradékban kifejezés használata az attribútum helyett.

5.4. példa: Tételezzük fel, hogy ugyanazt az eredményt szeretnénk kapni mint az 5.3. példában, de a hossz órákban legyen kifejezve. A SELECT záradékot a következőkkel egészíthetnénk ki:

```
SELECT cím AS név, hossz*0.016667 AS hosszÓrákban
```

Így ugyanazokat a név-hossz párokat eredményezné, de az időtartamot órákban kapnánk meg, és a második oszlop neve hosszÓrákban. □

Kisbetű/nagybetű

Az SQL nem különbözteti meg a kis- és nagybetűket. Például, ugyan mi úgy döntöttünk, hogy az olyan kulcsszavakat mint a FROM nagybetűkkel írjuk, azonban ugyanúgy helyes a `From` vagy `from`, vagy akár `FrOm` is. Az attribútumnevek, a relációnevek, a másodnevek stb., mind függetlenek attól, hogy kis- vagy nagybetűvel írtuk őket. Az SQL csak az idézőjelek közé tett kifejezések esetén tesz különbséget kis- és nagybetűk között. Így, a 'FROM' és a 'from' különböző karaktersorok; melyek egyike sem egyezik meg a FROM kulcsszóval.

5.5. példa: A SELECT záradékban a tételek között konstansokat is megadhatunk. Ez értelmetlennek tűnik, de alkalmazható például arra, hogy fontos szavakat illesszünk az eredménybe. A következő lekérdezés:

```
SELECT cím, hossz*0.16667 AS hossz, 'óra' AS Órákban
FROM Film
WHERE stúdióNév = 'Disney' AND év = 1990;
```

olyan sorokat generál, mint:

cím	*hossz*	*Órákban*
Micsoda nő!	1,98334	Óra

A harmadik oszlop neve Órákban, mely a második oszlop címével összetartozik. A válasz minden sorában ott van az 'óra' konstans, mely így a második oszlopbeli érték egységeként szerepel. □

5.1.2. Kiválasztás az SQL-ben

Az SQL WHERE záradéka kiterjeszti a relációs algebra kiválasztás operátorát. A WHERE-t olyan feltételkifejezések követhetik, mint amelyeket a közismert C vagy Pascal nyelvek is használnak.

A hat alapvető összehasonlítási operátor segítségével =, <>, <, >, <=, >= értékeket hasonlíthatunk össze, s ezekkel építhetjük fel a feltételkifejezéseket. Ezek az operátorok megegyeznek a Pascal által használtakkal, és nyilvánvaló az értelmük (azok számára, akik nem nagy Pascal-rajongók, a „<>" értelme „nem egyenlő").

Az összehasonlítható értékek között lehetnek konstansok és a FROM utáni reláció vagy relációk attribútumai. Az értékekre alkalmazhatjuk a szokásos matematikai alapműveleteket is, mint például a +, * stb., mielőtt összehasonlítjuk őket. Például az (*év* – 1930)*(*év* – 1930) < 100 kifejezés igaz az 1930-tól legfeljebb 9 év távolságra levő évekre. Alkalmazhatjuk az összekapcsolás műveletet is: (||) karakterláncokra; például 'labda'||'rúgás' értéke 'labdarúgás'.

Egy példát láthattunk az összehasonlításra az 5.1. példában:

```
stúdióNév = 'Disney'
```

A Film reláció stúdióNév attribútumának értékét összehasonlítjuk a 'Disney' konstanssal. Ennek a konstansnak karakterlánc értéke van, melyet az SQL-ben egy idézőjellel jelölünk. Megengedettek numerikus konstansok is, mint az egész számok vagy a valós számok; az SQL a szokásos jelöléseket alkalmazza a valós számokra: 13.35 vagy 1.23E45.

Az összehasonlítás eredményének van egy igazságértéke: TRUE (igaz) vagy FALSE (hamis). A logikai értékeket kombinálhatjuk az AND, OR és NOT logikai műveletek segítségével, melyeknek a szokásos jelentésük van, mint például a Pascal nyelvben. Az 5.1. példában láthattuk, hogyan lehet két feltételt az AND művelettel összekapcsolni. Ebben a példában a WHERE feltétel igazságértéke akkor és csak akkor lesz igaz, ha mindkét összehasonlítás eredménye igaz, azaz a stúdió neve 'Disney' és az év 1990. A következőkben még néhány összetett WHERE feltételű példát mutatunk be.

5.6. példa: A következő lekérdezés azon fekete-fehér filmek címeit eredményezi, melyeket 1970 után gyártottak:

```
SELECT cím
FROM Film
WHERE év > 1970 AND NOT színes;
```

Ebben a feltételben ismét az AND segítségével kapcsolunk össze két logikai értéket. Az első egy közönséges összehasonlítás, a második viszont a színes attribútum, tagadva. Az attribútum ilyen jellegű használatára az ad jogot, hogy az attribútum típusa logikai érték.

Tekintsük a következő lekérdezést:

```
SELECT cím
FROM Film
WHERE (év > 1970 OR hossz < 90) AND stúdióNév = 'MGM';
```

Ez az utasítás azon filmcímeket adja meg, melyeket az MGM Stúdió készített, és melyek vagy 1970 után készültek vagy 90 percnél rövidebbek. Figyeljük meg, hogy az összehasonlításokat zárójelek segítségével csoportosíthatjuk. A zárójelekre azért van szükség, mert az SQL-ben a logikai műveletek megelőzési sorrendje ugyanaz mint a legtöbb programozási nyelvben: az AND megelőzi az OR műveletet, a NOT pedig megelőzi mindkettőjüket. □

Logikai értékek és bitláncok tárolása

A logikai értékeket az SQL-ben a *bitláncok* speciális eseteként tekinthetjük. Egy bitláncot egy B karakter és két idézőjel közötti 0 és 1 karakterek sorozata jelképez. Így a `B'011'` egy hárombites láncot jelképez, melyek közül az első 0, a másik kettő 1. Hexadecimális jelölést is alkalmazhatunk, melyben egy X-et követ egy idézőjelek közötti hexadecimális szám (a számjegyek 0-tól 9-ig és *a*-tól *f*-ig tartanak, melyek a 10-től 15-ig terjedő „számjegyeket" jelképezik). Például `X'7ff'` egy tizenkét bites láncot jelképez, az első bit 0, utána pedig tizenegy 1-es következik. Minden hexadecimális számjegy négy bitet jelképez, és a számkezdő 0-kat nem hagyhatjuk el.

A TRUE (igaz) logikai értéket 1 biten tárolhatjuk, `B'1'` formában. Hasonlóképpen a FALSE (hamis) értéket `B'0'`-ként tárolhatjuk.

5.1.3. Karakterláncok összehasonlítása

Két karakterlánc egyenlő, ha a karaktereknek ugyanabból az egymás után következő sorozatából áll. Az SQL több típusú karakterláncot is használ, mint például a rögzített hosszúságú karaktertömböt, vagy a változó hosszúságú karakterlistát.[2] A különböző karakterlánctípusok között elfogadható rugalmasságot várhatunk el. Például egy olyan karakterlánctípusok között elfogadható rugalmasságot várhatunk el. Például egy olyan karakterláncot, mint a `'gól'`, tárolhatunk egy rögzített 10 hosszúságú karakterláncként, 7 üres karakterrel a végén, vagy egy változó hosszúságú karakterláncként. Elvárnánk, hogy a két érték egyenlő legyen és hogy mindkettő egyenlő legyen a `'gól'` konstans karakterlánccal.

Amikor két karakterláncot összehasonlítunk egy aritmetikai összehasonlító operátorral, mint a < vagy a >=, azt szeretnénk tudni, hogy az egyik megelőzi-e a másikat lexikografikus rendezésben (azaz ábécérend szerint). Ha $a_1a_2...a_n$ és $b_1b_2...b_m$ két karakterlánc, akkor az első „kisebb mint" a második, ha $a_1 < b_1$, vagy $a_1 = b_1$ és $a_2 < b_2$, vagy $a_1 = b_1$, $a_2 = b_2$ és $a_3 < b_3$ stb. $a_1a_2...a_n < b_1b_2...b_m$ akkor is, ha $n < m$ és $a_1a_2...a_n = b_1b_2...b_n$, azaz az első karakterlánc egy valódi prefixe, eleje a másodiknak. Például `'labda'<'lap'`, mivel az első két karakter azonos a két karakterláncban, a harmadik karaktere az elsőnek pedig megelőzi a második karakterlánc harmadik karakterét. `'kar'<'kard'`, mert az első a második valódi prefixe. Egyenlőség vizsgálatakor elvárunk bizonyos fokú rugalmasságot a különböző típusok között.

Az SQL lehetővé teszi, hogy mintákkal is összehasonlíthassunk karakterláncokat. Az összehasonlítás ilyen jellegű formája:

```
s LIKE p
```

[2] Tulajdonképpen ezeket a típusokat csak úgy tekintjük, mintha tömbként vagy listaként lennének tárolva, valójában azonban az SQL szabványok nem rögzítik le, hogyan kell tárolni és a tárolás implementációfüggő.

Escape karakterek a LIKE típusú kifejezésekben

Mi történik, ha az a karaktersor típus, melyre a LIKE kifejezéssel szeretnénk keresni, tartalmazza a % vagy _ karaktereket? Szükségünk van egy *escape* karakterre, mely semlegesíti a % vagy _ karakterek speciális jellegét. Ez az escape karakter nincs rögzítve, mint a UNIX esetében a \, hanem az SQL utasításban meg lehet adni mi legyen az, mégpedig úgy, hogy a minta után az ESCAPE kulcsszót írjuk, majd idézőjelek közé a kiválasztott karaktert. A % és _ karakterek, ha egy escape karakter áll előttük, csak önmaguknak felelnek meg a karaktersorban, és nem egy bármilyen karakterekből álló karaktersorozatnak, illetve egy bármilyen karakternek. Például az:

```
s LIKE 'x%%x%' ESCAPE 'x'
```

kifejezésben x az escape karakter az `'x%%x%'` mintában. Az x% karaktersor egyetlen %-nak fog megfelelni. Így a minta minden olyan karaktersornak megfelel, mely a % karakterrel kezdődik és fejeződik be.

ahol *s* egy karakterlánc, *p* pedig egy *minta*. A minta egy karakterlánc, melyben használhatjuk a speciális % és _ karaktereket. A *p* többi karaktere csak önmagának felel meg az *s*-ben. A % jel a *p*-ben megfelel az *s*-ben bármilyen karakterek 0 vagy nagyobb hosszúságú sorozatának, míg az _ jel a *p*-ben megfelel az *s*-ben egy bármilyen karakternek. Ez a feltétel akkor és csak akkor igaz, ha az *s* karakterlánc megfelel a *p* mintának. Hasonlóképpen *s* NOT LIKE *p* akkor és csak akkor igaz, ha az *s* karakterlánc nem felel meg a *p* mintának.

5.7. példa: Egy olyan filmet keresünk, mely úgy kezdődik, hogy `'Halálos'` és tudjuk hogy a címben lévő második szó 7 karakterből áll. Mi lehet a film címe? Az ilyen típusú címeket a következő lekérdezéssel tudhatjuk meg:

```
SELECT cím
FROM Film
WHERE cím LIKE 'Halálos _______';
```

Ez a programrész olyan filmcímeket keres, melyek 15 karakter hosszúak, az első 7 karakter: `'Halálos'`, a nyolcadik üres, az utolsó 7 karakter pedig bármi lehet. A lekérdezés eredménye a feltételnek megfelelő filmcímek, mint például a *Halálos fegyver* vagy a *Halálos játszma*.

5.8. példa: Azokat a filmeket keressük, melyeknek a címében megtalálható az *'s* karaktersor. A megfelelő utasítás:

```
SELECT cím
FROM Film
WHERE cím LIKE '%''s%';
```

Ahhoz hogy megértsük a példát, először is vegyük észre, hogy mivel az idézőjel a karaktersorok határoló karaktere az SQL-ben, nem képviselheti önmagát. Az SQL azt a szabályt alkalmazza, hogy két egymás utáni idézőjel egy karaktersorban egy idézőjelet jelképez és nem zárja le a sort. Így az ' karaktersor egy idézőjelnek felel meg, s pedig önmagának.

A minta két végén a % karakterek bármilyen karaktersornak megfelelnek, így a válasz olyan filmeket fog tartalmazni, mint például a *Wayne's World* (Wayne világa). □

5.1.4. Dátumok és időpontok összehasonlítása

Az SQL különböző megvalósításai általában speciális típusként kezelik a dátum és idő típusokat. Ezek az adatok aztán sokféleképpen tárolhatók, mint például `5/14/1948` vagy `14 May 1948`. Ebben a részben csak az SQL2 szabványt adjuk meg, amely aprólékosan leírja ezeket a formátumokat.

Egy *dátumot* a `DATE` kulcsszóval, utána pedig egy idézőjelek közötti speciális karaktersorral írhatunk le. Például, a `DATE '1948-05-14'` megfelel a leírásnak. Az első négy karakter az év számjegyeit jelképezi. Utána következik egy gondolatjel, majd két számjegy mely a hónapot jelképezi. Figyeljük meg, hogy egy egyszámjegyű hónap ki van egészítve elöl egy 0 karakterrel. Végül következik egy újabb gondolatjel és két számjegy, mely a napot jelképezi. Ugyanúgy mint a hónapok esetében, a napot is kiegészítjük egy kezdő 0-val hogy kétszámjegyű számot kapjunk.

Egy *idő* értéket hasonlóan a `TIME` kulcsszó és egy idézőjelek közötti karakterlánc segítségével lehet ábrázolni. A karaktersorban az órának két számjegy felel meg a 24 órás órán. Ezután kettőspont következik, két számjegy a perceknek, újabb kettőspont, majd két számjegy a másodperceknek. Ha a másodperc törtrészeit is szeretnénk használni, következhet egy pont és annyi számjegy, amennyire szükség van. Így, a `TIME '15:00:02.5'` azt az időt jelképezi, amikor már az összes diák elhagyta az osztálytermet egy olyan óra után, mely délután 3-kor ért véget, tehát két és fél másodperccel három után.

A dátum és idő értékeket ugyanúgy összehasonlíthatjuk mint a karakterlánc vagy numerikus értékeket. A < jel dátumok esetén azt jelenti, hogy az első korábbi dátum mint az utóbbi, idő értékek esetében pedig azt jelenti, hogy az első korábbi időpont (egy napon belül), mint a második.

5.1.5. Az eredmény rendezése

Szükség lehet arra, hogy egy lekérdezés eredménye bizonyos sorrendbe legyen rendezve. A sorrend valamely attribútum értékén alapulhat, egyenlőség esetén egy második attribútum értékén, további egyenlőség esetén egy harmadik attribútum értékén

stb. Az eredmény rendezése érdekében a select-from-where utasításhoz a következő záradékot adjuk hozzá:

```
ORDER BY <attribútumlista>
```

A rendezés alapértelmezésben növekvő sorrendben történik, de csökkenően is lekérhetjük az adatokat, a `DESC` kulcsszó megadásával (a „descending" – „csökkenő" rövidítése). Hasonlóképpen megadhatjuk azt is, hogy a rendezés növekvően történjen az `ASC` kulcsszóval, de ez felesleges, mert ha nem adjuk meg hogyan történjen, automatikusan így rendeződik.

5.9. példa: Írjuk át az 5.1. példát, mely az 1990-es Disney-filmeket kérdezte le a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

relációból. A filmeket hossz szerint szeretnénk lekérdezni növekvően, és az egyenlő hosszúakat ábécérendben. A megfelelő utasítás:

```
SELECT *
FROM Film
WHERE stúdióNév = 'Disney' AND év=1990
ORDER BY hossz, cím;
```

Ha az attribútumoknak létezik egy sorrendje (aminek léteznie kellene, mivel az SQL-ben, ahogy az 5.7.2. alfejezetben látni fogjuk, a relációkat egy attribútumlistán keresztül építjük fel), akkor az attribútumnevek helyett használhatjuk a sorszámaikat. Így a fenti `ORDER BY` záradék így nézne ki:

```
ORDER BY 3, 1;
```

annak a sorrendnek megfelelően, ahogy a `Film` reláció attribútumait felsoroltuk. □

5.1.6. Feladatok

5.1.1. feladat: Ha egy lekérdezésben megtalálható a következő `SELECT` záradék:

```
SELECT A B
```

honnan tudjuk, hogy *A* és *B* két különböző attribútum, vagy *B* másodneve az *A*-nak?

5.1.2. feladat: Adjuk meg SQL-ben a következő lekérdezéseket, a fejezet elején említett film adatbázisra vonatkozóan:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
FilmSzínész(név, cím, nem, születésnap)
Gyártásirányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

* a) Keressük meg az MGM stúdió címét.

b) Keressük meg Sandra Bullock születésnapját.

* c) Keressük meg az összes olyan filmsztárt, akik vagy egy 1980-as filmben szerepeltek, vagy egy olyanban, melynek a címében szerepel a 'Szerelem' szó.

d) Keressük meg az összes olyan gyártásirányítót, akiknek a nettó bevétele legalább 10 000 000 $.

e) Keressük meg az összes olyan filmsztárt, akik vagy férfiak, vagy Malibuban laknak (a címük tartalmazza a Malibu szót).

5.1.3. feladat: Adjuk meg a következő SQL lekérdezéseket, melyek a 4.1.1. feladat adatbázisára vonatkoznak:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A lekérdezések eredményeit szemléltessük a 4.1.1. feladat adataival.

* a) Keressük meg a modellszámát, sebességét, merevlemez-kapacitását azoknak a PC-knek, melyek ára 1600 $ alatt van.

* b) Ugyanaz, mint az a) pontban, de nevezzük át a *sebesség* oszlopot *megaherzre*, a *merevlemez* oszlopot pedig *gigabájtra*.

c) Keressük meg a nyomtatók gyártóit.

d) Keressük meg azon laptopok modellszámát, memóriakapacitását és képernyőnagyságát, melyek több mint 2000 $-ba kerülnek.

* e) Keressük meg a `Nyomtató` reláció azon sorait, melyek a színes nyomtatókra vonatkoznak. Vegyük figyelembe, hogy a `színes` attribútum logikai típusú.

f) Keressük meg azon PC-k modellszámát, sebességét, memórianagyságát, melyeknek 6× vagy 8× CD-jük van, és az áruk 2000 $ alatt van. A `cd` attribútum karakterlánc típusú.

5.1.4. feladat: Adjuk meg SQL-ben a következő lekérdezéseket, melyek a 4.1.3. feladat adatbázissémájára alapulnak:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

és a lekérdezések eredményeit szemléltessük a 4.1.3. példa adataival.

a) Keressük meg az osztálynevét és országát azon hajóosztályoknak, melyek legalább 10 ágyúval rendelkeznek.

b) Keressük meg az 1918 előtt felavatott hajókat, de az eredmény oszlop neve legyen `hajóNév`.

c) Keressük meg a csatákban elsüllyesztett hajók nevét és azon csaták nevét, melyben elsüllyedtek.

d) Keressük meg azon hajókat, melyek neve megegyezik a hajóosztályuk nevével.

e) Keressük meg az „R" betűvel kezdődő hajók nevét.

! f) Keressük meg az olyan hajóneveket, melyek három vagy több szóból állnak (például *Nagy Lajos király*).

5.2. Több relációra vonatkozó lekérdezések

A relációs algebra legfőbb erőssége az, hogy két vagy több relációt tud kombinálni összekapcsolásokon, szorzatokon, egyesítéseken, metszeteken és különbségeken keresztül. Ennek az öt műveletnek bármelyikét alkalmazhatjuk az SQL-ben. A halmaz műveletek – egyesítés, metszet és különbség – direkt módon jelennek meg az SQL-ben, ahogy az 5.2.5. alfejezetben szemléltetni fogjuk. Először azonban vizsgáljuk meg, hogy a select-from-where utasítás milyen módon teszi lehetővé szorzatok és összekapcsolások alkalmazását.

5.2.1. Szorzat és összekapcsolás az SQL-ben

Több reláció összekapcsolása nagyon egyszerűen valósítható meg az SQL-ben: fel kell sorolni az összes relációt a `FROM` záradékban. Ezután a `SELECT` és `WHERE` záradékok a `FROM` záradék minden relációjának minden attribútumát felhasználhatják.

5.10. példa: Szeretnénk megtudni, hogy *George Lucas* mely években gyártott filmet. A következő két relációra van ehhez szükség:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A GyártásIrányító relációból egy egyszerű lekérdezéssel megtudhatjuk George Lucas azonosítóját. Az azonosító segítségével a Film relációból egy újabb lekérdezésen keresztül megtudhatjuk azokat az évszámokat, amikor George Lucas filmet gyártott. Ezt a két lépést összevonhatjuk egyetlen lekérdezésbe, mely a Film és a GyártásIrányító relációkra vonatkozik:

```
SELECT év
FROM Film, GyártásIrányító
WHERE név = 'George Lucas' AND producerAzon = azonosító;
```

Ez a lekérdezés az összes Film-GyártásIrányító sorpárt megvizsgálja és ezekre ellenőrzi a WHERE feltételt:

1. A GyártásIrányító reláció sorának név attribútuma 'George Lucas' értékű kell legyen.
2. A Film sor producerAzon attribútumának ugyanaz kell legyen az értéke, mint a GyártásIrányító sor azonosító attribútumának, azaz a két sor ugyanarra a gyártásirányítóra vonatkozik.

Valahányszor találunk egy sorpárt, mely mindkét feltételnek eleget tesz, a Film sor év attribútumértéke belekerül az eredménybe. Az összes sorpárban, melyek a feltételeket kielégítik, a gyártásirányító neve *George Lucas* lesz és a filmek az általa gyártott filmek. Az eredményben lesz például az 1977-es év, amikor a *Csillagok háborúja* készült. □

5.2.2. Attribútumok megkülönböztetése

Előfordulhatnak olyan lekérdezések, melyekben két vagy több reláció szerepel, és ezekben a relációkban két vagy több attribútumnak ugyanaz a neve. Ilyen esetben valamilyen módon jeleznünk kell, hogy melyik attribútumról van szó, amikor a közös név szerepel a lekérdezésben. A problémát az SQL úgy oldja meg, hogy megengedi egy relációnévnek és egy pontnak a használatát egy attribútum előtt. Így *R.A* az *R* reláció *A* nevű attribútumát jelképezi.

5.11. példa: Tekintsük a következő két relációt:

```
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```


Mindkettőben szerepelnek a név és cím attribútumok. Tételezzük fel, hogy olyan filmszínész-gyártásirányító párosokat keresünk, melyek ugyanazon a címen laknak. A megfelelő lekérdezés a következő:

```
SELECT FilmSzínész.név, GyártásIrányító.név
FROM FilmSzínész, GyártásIrányító
WHERE FilmSzínész.cím = GyártásIrányító.cím;
```

Ebben a lekérdezésben olyan FilmSzínész-GyártásIrányító sorpárokat keresünk, melyekben a cím attribútumérték megegyezik. A WHERE záradék fejezi ki a cím attribútumokra vonatkozó feltételt. Mindegyik olyan sorpár esetén, mely megfelel a feltételnek, mindkét név attribútumot hozzáadjuk az eredményhez. Így az eredményben olyan névpárok lesznek, mint:

FilmSzínész.név	*GyártásIrányító.név*
Jane Fonda	Ted Turner

□

A relációnevet és a pontot olyan esetben is az attribútum elé írhatjuk, amikor nincs névazonosságból származó kétértelműség. Például az 5.10. példa lekérdezését így is írhatnánk:

```
SELECT Film.év
FROM Film, GyártásIrányító
WHERE GyártásIrányító.név = 'George Lucas'
    AND Film.producerAzon = GyártásIrányító.azonosító;
```

A relációnév és pont kombinációt a lekérdezésben szereplő bármely attribútumra alkalmazhatjuk.

5.2.3. Sorváltozók

A relációnév előtagként történő alkalmazása jó megoldás az attribútumok névütközésének elkerülésére addig, amíg több különböző reláció szerepel a lekérdezésben. Néha azonban olyan lekérdezésre van szükségünk, mely ugyanazon reláció két vagy több sorát kombinálja össze. Az *R* relációt annyiszor sorolhatjuk fel a WHERE záradékban, ahányszor szükségünk van rá, de valamilyen módon meg kell tudnunk különböztetni az előfordulásait. Az SQL-ben lehetőségünk van arra, hogy a FROM záradékban *R* minden előfordulásához hozzárendeljünk egy másodnevet, melyet *sorváltozónak* is nevezünk. A FROM záradékban *R* minden előfordulása után következhet az AS kulcsszó és a sorváltozó neve.

R attribútumait a SELECT és WHERE záradékokban megkülönböztethetjük egy

Sorváltozók és relációnevek

Gyakorlatilag a SELECT és WHERE záradékokban az attribútumokra *mindig* úgy utalunk mint egy sorváltozó komponenseire. Viszont ha egy reláció csak egyszer szerepel a FROM záradékban, akkor a relációnevet használhatjuk mint sorváltozót. Így a FROM záradékbeli *R* relációnevet tekinthetjük úgy, mint egy rövidítést az *R* AS *R* helyett.

előtag segítségével, mely a megfelelő sorváltozóból és egy pontból áll. Tehát a sorváltozó az *R* reláció másodneveként szerepel és bárhol használható az *R* helyett.

5.12. példa: Míg az 5.11. példában olyan filmszínészeket és gyártásirányítókat kerestünk, akiknek azonos a lakcímük, hasonlóképpen kereshetünk olyan színészpárokat is, akik egy helyen laknak. A lekérdezés lényegében ugyanaz, de most a FilmSzínész reláció két sorát kell párosítani és nem a FilmSzínész reláció egy sorát a GyártásIrányító reláció egy sorával. A FilmSzínész reláció két előfordulására sorváltozókat használunk másodnévként, így a lekérdezés alakja a következő lesz:

```
SELECT Színész1.név, Színész2.név
FROM FilmSzínész AS Színész1, FilmSzínész AS Színész2
WHERE Színész1.cím = Színész2.cím AND
    Színész1.név < Színész2.név;
```

A FROM záradékban a Színész1 és Színész2 sorváltozókat használjuk a FilmSzínész reláció másodneveiként. A SELECT záradékban a sorváltozók segítségével különböztetjük meg a két sor azonos komponenseit. A másodneveket a WHERE záradékban is használjuk, annak kifejezésére, hogy a két FilmSzínész sor cím attribútumának ugyanaz az értéke.

A WHERE záradék második feltétele, Színész1.név < Színész2.név, azt fejezi ki, hogy az első színész neve ábécérendben megelőzi a második színész nevét. Ha ezt a feltételt elhagynánk, akkor Színész1 és Színész2 vonatkozhatnának ugyanarra a sorra. Ebben az esetben azt találnánk, hogy a két sorban a címek természetesen megegyeznek, tehát az eredménybe belekerülne minden színész neve önmagával párosítva.[3] A második feltétel azt is kikényszeríti, hogy az eredményben minden közös lakcímű színészpár csak egyszer szerepeljen, ábécérendben. Ha a <> (nem egyenlő) jelet használnánk összehasonlítási operátorként, akkor az eredmény minden színész házaspárt kétszer tartalmazna, például:

[3] Ugyanaz a probléma előfordulhat az 5.11. példában is, ha egy személy filmszínész és gyártásirányító is. A problémát hasonlóan tudjuk megoldani, ha feltételként szabjuk, hogy a két név legyen különböző.

Színész1.név	*Színész2.név*
Alec Baldwin	Kim Basinger
Kim Basinger	Alec Baldwin

□

```
A from záradék sorváltozói tartozzanak az R1, R2,...,Rn
   relációkhoz
MINDEN t1 sorra az R1 relációban
   MINDEN t2 sorra az R2 relációban
          ...
      MINDEN tn sorra az Rn relációban
         HA a where záradék igaz, amikor az attribútumokban
      t1, t2, ..., tn megfelelő értékei találhatóak AKKOR
            t1, t2, ..., tn-nek megfelelően kiértékeljük a
         select záradék attribútumait és az értékekből
         alkotott sort az eredményhez adjuk
```

5.2. ábra. *Egy egyszerű SQL lekérdezés kiértékelése*

5.2.4. Lekérdezések értelmezése

Az előzőekben bemutatott select-from-where kifejezések értelmezésére számos lehetőség van. Ezek az értelmezések mind *egyenértékűek*, abban az értelemben, hogy ugyanazt a választ eredményezik, ha ugyanarra az adatbázis-előfordulásra alkalmazzuk ugyanazt a lekérdezést. Vizsgáljuk meg őket sorban.

Beágyazott ciklusok

A példákban eddig a sorváltozók szemantikáját alkalmaztuk. Idézzük fel, hogy egy reláció másodneve egy sorváltozó, amely a relációhoz tartozó összes sor értékét felveszi. Egy másodnév nélküli relációnév értelmezhető saját másodneveként. Ha több sorváltozónk van, mindegyikhez rendelhetünk egy-egy ciklust, melyben a változó végigmegy a neki megfelelő reláció összes során. A ciklusok egymásba vannak ágyazva. A sorváltozók mindegyik értékadásakor leellenőrizzük, hogy a WHERE feltétel igaz-e vagy sem. Ha igaz, akkor a SELECT záradékbeli komponenseknek megfelelő értékekből összeállított sort az eredménybe illesztjük. A lekérdezést feldolgozó algoritmust az 5.2. ábra érzékelteti.

Párhuzamos értékadás

Ebben az esetben nem kell explicit módon beágyazott ciklusokat létrehozni. Ehelyett tekintsük tetszőleges sorrendben vagy párhuzamosan a sorváltozók összes lehetséges értékadását a megfelelő relációkból. Minden egyes értékadásra megvizsgáljuk, hogy a

Datalog értelmezés és SQL értelmezés

Figyeljük meg a hasonlóságot a Datalog szabályok második értelmezése, melyet a 4.2.4. alfejezetben mutattunk be, és az SQL select-from-where utasításainak második értelmezése között. A Datalog esetében a törzs relációs részcéljaihoz hozzárendeltük az összes lehetséges sort a megfelelő relációból. Az SQL-ben a sorváltozókhoz rendeljük hozzá az összes lehetséges sort. Mindkét esetben, az aritmetikai predikátumok (SQL-ben a WHERE záradék részei) korlátozzák az értékadásokat, míg az eredménysorokat a szabály fejének kiértékeléséből kapjuk (SQL-ben a SELECT záradékából).

WHERE feltétel igaz-e. Minden egyes olyan értékadás, melyre a WHERE igaz, egy sort ad a válaszhoz. A válaszbeli sorok a SELECT záradék attribútumaiból épülnek fel, az aktuális értékadásnak megfelelően.

Konverzió a relációs algebrába

Egy harmadik megközelítés az SQL lekérdezést a relációs algebrával köti össze. A FROM záradék sorváltozóiból indulunk ki és tekintjük a Descartes-szorzatukat. Ha két változó ugyanarra a relációra vonatkozik, akkor a reláció kétszer szerepel a szorzatban, és az attribútumait átnevezzük úgy, hogy minden attribútumnak egyedi neve legyen. Hasonlóan a különböző relációkban szereplő azonos attribútumneveket is átnevezzük, hogy elkerüljük a kétértelműséget.

A WHERE záradékot átalakítjuk egy kiválasztási feltétellé, melyet alkalmazunk az elkészített szorzatra. A WHERE záradékbeli mindegyik attribútumutalást kicseréljük arra a szorzatbeli attribútumra, melynek megfelel. Végül a SELECT záradék alapján létrehozunk egy attribútumlistát. A vetítési művelethez az attribútumokat ugyanúgy határozzuk meg, mint a kiválasztáshoz: a SELECT záradékban minden attribútumot kicserélünk a szorzat neki megfelelő attribútumára.[4]

5.13. példa: Írjuk át az 5.12. példát relációs algebrára. Először is a FROM záradékban két sorváltozó van, mindkettő a FilmSzínész relációra vonatkozik. Ezért az algebrai kifejezés így kezdődik:

FilmSzínész × FilmSzínész

[4] Tulajdonképpen a relációs algebra nem engedi meg az aritmetikai műveleteket a SELECT záradékban, míg az SQL, ahogy az 5.4. példában láthattuk, megengedi ezen műveletek használatát. A relációs algebra vetítési műveletét nyugodtan kiterjeszthetnénk, hiszen a vetítés csak történelmi okokból van korlátozott módon definiálva.

Az SQL szemantika egy meglepő következménye

Tételezzük fel, hogy R, S és T unáris (egyoszlopos) relációk és mindegyik egyedüli attribútuma az A. Szeretnénk megkeresni azokat az elemeket, melyek az R-ben és vagy az S-ben vagy a T-ben (vagy mindkettőben) megtalálhatóak. Tehát az $R \cap (S \cup T)$ halmazt szeretnénk meghatározni. Azt hihetnénk, hogy a következő lekérdezés a megfelelő választ eredményezi:

```
SELECT R.A
FROM R, S, T
WHERE R.A = S.A OR R.A = T.A
```

Vizsgáljuk meg azt a speciális esetet, amikor T üres. Mivel az $R.A=T.A$ egyenlőség nem teljesülhet, az „OR" művelettel kapcsolatos ismereteink alapján azt várnánk, hogy az eredmény $R \cap S$ legyen. Mégis, az 5.2.4. alfejezet három értelmezése közül bármelyiket is választjuk ki, a lekérdezés eredménye az üres halmaz lesz, függetlenül attól, hogy R-nek és S-nek hány közös eleme van. Ha az 5.2. ábrán bemutatott beágyazott ciklus szemantikát használjuk, észrevehető, hogy a T változó ciklusa 0 hosszúságú, mivel a relációnak, mely fölött a T változik, nincs egy sora sem. Így a legbelső ciklusbeli HA utasítás egyszer sem kerül végrehajtásra, tehát az eredmény üres lesz. Hasonlóképpen, ha a sorváltozók lehetséges értékadásait tekintjük, kiderül, hogy a T változóhoz nem rendelhető érték, tehát nincs egy lehetőség sem a sorváltozók érték-hozzárendelésére. Végül, ha a Descartes-szorzatos megközelítést alkalmazzuk, akkor a kiindulópontunk az $R \times S \times T$ szorzat, amely üres, mivel T is üres.

Az eredményrelációnak nyolc attribútuma van, az első négy a FilmSzínész reláció első másolatának név, cím, nem és születésnap attribútumainak felel meg, a következő négy pedig a FilmSzínész reláció második másolatának ugyanazon attribútumainak. Az attribútumokat jelölhetnénk a sorváltozóval és egy ponttal – például Színész1.nem –, de a tömörség kedvéért nevezzük át az attribútumokat A_1, A_2, ..., A_8-nak. Így A_1 megfelel Színész1.név-nek, A_5 pedig Színész2.név-nek stb.

Ezzel az attribútumátnevezési stratégiával a WHERE záradék kiválasztási feltétele így alakul: $A_2 = A_6$ és $A_1 < A_5$. A vetítési lista A_1, A_5. Így a következő kifejezés adja meg a lekérdezésnek megfelelő algebrai kifejezést:

$$\pi_{A_1,A_5}(\sigma_{A_2 = A_6 \text{ AND } A_1<A_5}(\rho_{M(A_1,A_2,A_3,A_4)}(\text{FilmSzínész}) \times \rho_{M(A_5,A_6,A_7,A_8)}(\text{FilmSzínész})))$$

□

5.2.5. Egyesítés, metszet és különbség az SQL-ben

Néha a relációs algebra halmazműveleteit: az egyesítést, a metszetet és a különbséget szeretnénk használni relációk kombinálására. Az SQL biztosítja a megfelelő operátorokat, melyeket lekérdezések eredményeire lehet alkalmazni, feltéve, hogy a lekérdezések ugyanolyan attribútumhalmazú relációkat eredményeznek. Az ∩, ∪ és – műveleteknek megfelelő kulcsszavak: `UNION`, `INTERSECT` és `EXCEPT`. Ezeket a kulcsszavakat két zárójel közé írt lekérdezés közé kell helyezni.

5.14. példa: Tételezzük fel, hogy azokat a színésznőket keressük, akik gyártásirányítók is, 10 000 000 $ feletti nettó bevétellel. A következő két relációra van szükségünk:

```
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

A lekérdezést az 5.3. ábra mutatja be. Az 1)–3) sorok egy olyan relációt eredményeznek, melynek attribútumai `név` és `cím`, és a színésznőket tartalmazza.

Hasonlóképpen, az 5)–7) sorok a „gazdag" gyártásirányítókat eredményezik, akiknek nettó bevétele meghaladja a 10 000 000 $-t. Ez a lekérdezés szintén egy olyan relációt eredményez, melynek attribútumai `név` és `cím`. Mivel a két relációséma megegyezik, alkalmazhatjuk rá a metszet műveletet, melyet a 4) sor ad meg. □

```
1) ( SELECT név, cím
2)   FROM FilmSzínész
3)   WHERE nem = 'N' )
4)        INTERSECT
5) ( SELECT név, cím
6)   FROM GyártásIrányító
7)   WHERE nettóBevétel > 10000000);
```

5.3. ábra. *Színésznők és gazdag gyártásirányítók metszete*

5.15. példa: Hasonló módon meghatározhatjuk a két személyhalmaz különbségét. A következő lekérdezés:

SQL lekérdezések olvashatósága

Általában az SQL lekérdezéseket úgy írjuk, hogy az olyan fontos kulcsszavak, mint a `FROM` vagy `WHERE` a sor elején álljanak. Ez a stílus sokkal jobban láthatóvá teszi a lekérdezés struktúráját. Egy rövid lekérdezést azonban egy sorba is írhatunk, mint az 5.15. példában. A lekérdezés tömörségét megtartva, ez a stílus is jó olvashatóságot biztosít.


```
(SELECT név, cím FROM FilmSzínész)
  EXCEPT
(SELECT név, cím FROM GyártásIrányító);
```

azon színészek nevét és címét adja meg, akik nem gyártásirányítók, függetlenül a nemtől és a nettó bevételtől. □

A fenti két példában a két halmaz attribútumai szerencsére megegyeztek. Szükség esetén azonban átnevezhetjük az attribútumokat, hogy azonos attribútumhalmazt kapjunk, mint az 5.3. példában.

5.16. példa: Keressük meg azon filmcímeket és éveket, melyek a `Film` vagy a `SzerepelBenne` relációkban megtalálhatók:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
```

Elvileg a két relációban megtalálható filmhalmazoknak meg kellene egyezniük. Gyakorlatilag azonban könnyen megtörténhet, hogy van olyan film, melyhez nincsenek felsorolva a benne szereplő színészek vagy olyan `SzerepelBenne` sor is létezhet, melynek megfelelő sor nincs a film relációban.[5] Így a lekérdezés:

```
(SELECT cím, év FROM Film)
  UNION
(SELECT filmCím AS cím, év FROM SzerepelBenne);
```

Az eredményreláció attribútumai: `cím` és `év`, azokat a filmeket fogja tartalmazni, melyek legalább az egyik relációban szerepelnek. □

5.2.6. Feladatok

5.2.1. feladat: Az aktuális adatbázist használva:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

adjuk meg SQL-ben a következő lekérdezéseket:

* a) Keressük meg az *Elemi ösztön* férfi szereplőit.

[5] Léteznek módszerek az ilyen eltérések megakadályozására, lásd a 6. fejezetet.

b) Kik szerepeltek az MGM által 1995-ben gyártott filmekben?

c) Ki az MGM stúdió elnöke?

*! d) Mely filmek hosszabbak az *Elfújta a szél* című filmnél?

! e) Kik azok a gyártásirányítók, akiknek több a nettó bevételük, mint Merv Griffinnek?

5.2.2. feladat: A 4.1.1. feladatbeli adatbázis alapján adjuk meg a megfelelő lekérdezéseket:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez,
       képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

és a 4.1.1. feladat adataival szemléltessük az eredményt.

* a) Keressük meg azon laptopok gyártóit és sebességét, melyeknek legalább egy gigabájtos merevlemeze van.

* b) Keressük meg a *B* gyártó által készített bármilyen típusú termékek számát és árát.

c) Keressük meg azon gyártókat, akik laptopokat gyártanak, de PC-ket nem.

! d) Keressük meg azon merevlemezméreteket, melyek legalább két PC-ben előfordulnak.

! e) Keressük meg azon PC modellpárokat, melyeknek ugyanakkora a sebessége és a memóriája. Egy párt csak egyszer listázzunk ki, azaz ha (i, j) az eredményben van, akkor (j, i) nincs benne.

!! f) Keressük meg azokat a gyártókat, akik legalább két különböző, 133-nál nagyobb sebességű számítógépet gyártanak (PC-t vagy laptopot).

5.2.3. feladat: A 4.1.3. feladat adatbázisát és adatait felhasználva adjuk meg a következő lekérdezéseket és eredményeiket:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```


a) Keressük meg a 35 000 tonnánál súlyosabb hajókat.

b) Keressük meg a Guadalcanal csatában részt vevő hajók nevét, vízkiszorítását és ágyúszámát.

c) Keressük meg az adatbázisban megtalálható összes hajót. (Vegyük figyelembe, hogy nem biztos, hogy az összes hajót tároltuk a `Hajók` relációban.)

! d) Keressük meg azokat az országokat, melyeknek csatahajóik és cirkálóik is vannak.

! e) Keressük meg azokat a hajókat, melyek megsérültek egy csatában, de részt vettek később egy másikban.

! f) Keressük meg azokat a csatákat, melyekben legalább három azonos országbeli hajó vett részt.

*! **5.2.4. feladat:** A relációs algebra egyik leggyakoribb kifejezése a következő:

$$\pi_L(\sigma_C(R_1 \times R_2 \ldots R_n)),$$

ahol L az attribútumok tetszőleges listája, míg C egy tetszőleges feltétel. Az R_1, R_2, ..., R_n lista tartalmazhatja ugyanazt a relációt többször is, ilyenkor az attribútumokat átnevezzük. Mutassuk meg, hogyan lehet kifejezni egy ilyen lekérdezést az SQL-ben.

! **5.2.5. feladat:** A relációs algebra egy másik tipikus lekérdezése:

$$\pi_L(\sigma_C(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n))$$

Hasonló feltevésekkel élünk, mint az 5.2.4. feladatban, azzal a különbséggel, hogy szorzat helyett természetes összekapcsolást használunk. Mutassuk meg, hogyan lehet egy ilyen lekérdezést SQL-ben megfogalmazni.

5.3. Alkérdések

Ebben a fejezetben kibővítjük ismereteinket azokkal a kifejezésekkel kapcsolatban, melyek megjelenhetnek a `WHERE` záradékban. Eddigi ismereteink szerint a feltételekben összehasonlíthatunk *skaláris* értékeket (olyan elemi értékek, mint az egészek, valós számok, karakterláncok, dátumok, vagy ezeket az értékeket eredményező kifejezések). A továbbiakban kiterjesztjük ezt azzal, hogy egész sorokat vagy éppen relációkat hasonlítunk össze. Első lépésként megismerjük, hogy hogyan kell alkérdéseket használni a feltételekben. Egy *alkérdés* egy olyan kifejezés, mely egy relációt eredményez.

Például alkérdés lehet egy select-from-where kifejezés. Miután bemutattuk, hogyan kell relációkat létrehozni alkérdések segítségével, ismertetünk néhány operátort, melyek segítségével a WHERE záradékban sorokat és relációkat tudunk összehasonlítani.

5.3.1. Skalár értéket adó alkérdések

Egy select-from-where kifejezés egy bármilyen oszlopszámú relációt eredményezhet és bármennyi sor lehet a relációban. Néha viszont csak egy bizonyos attribútum értékeire van szükségünk. Ezenkívül, a kulcsok alapján arra következtethetünk, hogy annak az attribútumnak csak egy értéke lesz az eredményben.

Ha így van, a zárójelek közé tett select-from-where kifejezést konstansként használhatjuk. Minden olyan helyen megjelenhet a WHERE záradékban, ahol egy konstans vagy egy sor egy attribútuma szerepelhet. Például az alkérdés eredményét összehasonlíthatjuk egy konstanssal vagy attribútummal.

5.17. példa: Keressük meg a *Csillagok háborújá*nak gyártásirányítóját. A következő két relációra van szükség:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

mert csak az első tartalmazza a filmcím információkat és csak a második a gyártásirányító neveket. Az információk az azonosítószámon kapcsolódnak össze. Ezek a számok egyértelműen azonosítják a gyártásirányítókat. A megfelelő lekérdezés:

```
SELECT név
FROM Film, GyártásIrányító
WHERE Film.cím = 'Csillagok háborúja'
                AND producerAzon = azonosító;
```

A WHERE záradékban a cím attribútum elé írtuk a reláció nevét és a pontot, mivel az attribútum mindkét relációban megtalálható.

Másképpen is vizsgálhatjuk ezt a lekérdezést. A Film relációból megtudhatjuk a *Csillagok háborúja* gyártásirányítójának azonosítóját. A GyártásIrányító relációból az azonosító alapján megtudhatjuk a megfelelő személy nevét. Az azonosító meghatározása egy alkérdésként fogalmazható meg. Az alkérdés eredményét, mely várhatóan egy érték lesz, a „fő" lekérdezésben használhatjuk fel. A lekérdezést az 5.4. ábra szemlélteti.

```
1) SELECT név
2) FROM GyártásIrányító
3) WHERE azonosító =
4)    (SELECT producerAzon
5)    FROM Film
6)    WHERE cím = 'Csillagok háborúja'
7)    );
```

5.4. ábra. *A* Csillagok háborúja *producereinek megkeresése beágyazott alkérdéssel*

A 4)–6) sorok tartalmazzák az alkérdést. Ha csak ezt a lekérdezést vizsgáljuk, észrevehető, hogy az eredmény egy egyoszlopos reláció lesz, mely várhatóan csak egy sort fog tartalmazni. A sor (12345) alakú lesz, azaz egyetlen oszlop, melyben az az egész szám található, mely George Lucas azonosítója. Ha nulla, vagy egynél több sort eredményez a 4)–6) sorok közötti lekérdezés, akkor a lekérdezés futás közbeni hibát fog jelezni.

Az alkérdés lefutása után az 1)–3) sorok közötti lekérdezés is végrehajtható, mintha az egész alkérdés helyett az 12345 konstans lenne behelyettesítve. Azaz a „fő" lekérdezés úgy hajtódik végre, mint a következő:

```
SELECT név
FROM GyártásIrányító
WHERE azonosító = 12345;
```

A lekérdezés eredménye George Lucas kell legyen.

Megfigyelhető, hogy az 5.4. ábrán látható lekérdezésben a cím attribútum előtt már nem kell megjelölni a relációt, mely őt tartalmazza, mert a lekérdezést szétbontottuk két külön lekérdezésre. □

5.3.2. Relációkat tartalmazó feltételek

Léteznek olyan SQL operátorok, melyeket alkalmazhatunk egy R relációra, és az eredmény logikai érték lesz. Az esetek többségében ilyenkor R egy select-from-where lekérdezés eredménye. Az operátorok közül néhánynak – IN, ALL és ANY – szüksége van még egy s skaláris értékre is, ebben az esetben R egyoszlopos reláció kell legyen. Az operátorok definíciói:

1. EXISTS R egy olyan feltétel, mely akkor és csak akkor igaz, ha R nem üres.

2. s IN R akkor és csak akkor igaz, ha s egyenlő valamelyik R-beli értékkel. Hasonlóképpen, s NOT IN R akkor és csak akkor igaz, ha s egyetlen R-beli értékkel sem egyenlő. Itt feltételeztük, hogy R egyoszlopos reláció. Az 5.3.3. alfejezetben vizsgálni fogjuk az IN és NOT IN operátorok kiterjesztését, amikor R-nek több attribútuma van és s egy sor.

3. s > ALL R akkor és csak akkor igaz, ha s nagyobb mint az R egyoszlopos reláció minden értéke. Hasonlóképpen, a > operátort analóg módon kicserélhetjük bármelyikkel a többi öt összehasonlítási operátor közül. Például, s <> ALL R ugyanazt eredményezi, mint az s NOT IN R.

4. s > ANY R akkor és csak akkor igaz, ha s nagyobb az R egyoszlopos reláció legalább egy értékénél. Hasonlóképpen, bármelyiket használhatjuk a többi öt összehasonlítási operátorból a > helyett. Például, s = ANY R ugyanaz mint az s IN R.

Hasonlóan más logikai kifejezésekhez, az EXISTS, ALL és ANY operátorokat tagadhatjuk, ha a NOT kulcsszót az egész kifejezés elé helyezzük. Így a NOT EXISTS R akkor és csak akkor igaz, ha R üres. NOT s > ALL R akkor és csak akkor igaz, ha s nem nagyobb minden R-beli értéknél, míg a NOT s > ANY R akkor és csak akkor igaz, ha s nem nagyobb egyetlen R-beli értéknél sem. Hamarosan példákon keresztül fogjuk bemutatni az operátorok használatát.

5.3.3. Sorokat tartalmazó feltételek

Az SQL-ben egy sor zárójelek közötti skalárértékek felsorolása. Ilyen például az (123, 'labda') és a (név, cím, nettóBevétel). Az első példában a sor komponensei konstansok, a másodikban attribútumok. Megengedett a konstansok és attribútumok kombinálása.

Ha egy t sornak és R relációnak ugyanannyi komponense van, akkor az 5.3.2. alfejezetben leírt kifejezésekben hasonlíthatjuk össze őket. Ilyen lehet például a t IN R vagy a t <> ANY R. A második kifejezés azt jelenti, hogy létezik az R-ben t-től különböző sor. Jegyezzük meg, hogy amikor egy sort összehasonlítunk egy R reláció soraival, a komponenseket az attribútumok R-beli sorrendjének megfelelően kell összehasonlítani.

```
 1) SELECT név
 2) FROM GyártásIrányító
 3) WHERE azonosító IN
 4)    (SELECT producerAzon
 5)    FROM Film
 6)    WHERE (cím, év) In
 7)       (SELECT filmCím, év
 8)       FROM SzerepelBenne
 9)       WHERE színész = 'Harrison Ford'
10)       )
11)    );
```

5.5. ábra. *Harrison Ford filmjeinek gyártásirányítói*

5.18. példa: Az 5.5. ábrán látható SQL lekérdezés a következő három relációra vonatkozik:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```


A lekérdezés azon filmek gyártásirányítóit keresi, amelyekben Harrison Ford játszott. Egy „fő" lekérdezésből áll, azon belül két beágyazott lekérdezésből.

A beágyazott lekérdezéseket belülről kifelé értékeljük ki. Induljunk ki a legbelső beágyazott lekérdezésből, amely a 7)–9) sorok között található. A lekérdezés a SzerepelBenne reláció sorait vizsgálja és azokat találja meg, melyekben a színészNév komponens értéke 'Harrison Ford'. Ezeknek a filmeknek a címét és gyártási évét tartalmazza az eredmény. Emlékezzünk vissza, hogy a cím és az év együtt alkotja a Film kulcsát, mely alapján pontosan azonosítani lehet. Így a 7)–9) sorok közötti lekérdezés eredménye olyan sorokat fog tartalmazni, mint a következők:

cím	*év*
Csillagok háborúja	1977
Az elveszett frigyláda fosztogatói	1981
A szökevény	1993
...	...

5.6. ábra. *Cím-év párok, melyeket a legbelső alkérdés eredményez*

Tekintsük a 4)–6) sorok közötti középső alkérdést. Olyan Film sorokat keres, melyek címe és gyártási éve az 5.6. ábrán szemléltetett relációban találhatók. Minden egyes megtalált sor esetén az eredménybe kerül a gyártásirányító azonosítója. Így a középső alkérdés eredménye a Harrison Ford-filmek gyártásirányítóinak azonosítóit fogja tartalmazni.

Végül vizsgáljuk meg az 1)–3) sorok közötti „fő" lekérdezést. Megvizsgálja a GyártásIrányító reláció sorait, hogy megtalálja azokat, melyek azonosító komponense a középső lekérdezés által eredményezett halmazban van. Minden egyes megfelelő sor esetén a gyártásirányító neve az eredménybe kerül, mely így azon filmek gyártásirányítóit fogja tartalmazni, melyekben Harrison Ford szerepelt. □

Az 5.5. ábra beágyazott lekérdezését, ugyanúgy mint sok más beágyazott lekérdezést, átírhatjuk egy egyszerű select-from-where lekérdezésre. A lekérdezés FROM záradékában szerepelni fog az összes olyan reláció, mely az 5.5. ábra fő lekérdezésében, vagy valamelyik alkérdésben megjelenik. Az IN kapcsolat helyett egyenlőségek lesznek a WHERE záradékban. Így lényegében az 5.7. ábrán látható lekérdezés megegyezik az 5.5. ábra lekérdezésével. Különbségek csak a gyártásirányítók ismétlődésének kezelésében vannak, ahogy az 5.4.1. alfejezetben látni fogjuk.

```
SELECT név
FROM GyártásIrányító, Film, SzerepelBenne
WHERE azonosító = producerAzon AND
   Film.cím = filmCím AND
   Film.év = SzerepelBenne.év AND
   Színész = 'Harrison Ford';
```

5.7. ábra. *Ford gyártásirányítói beágyazott lekérdezések nélkül*

5.3.4. Korrelált alkérdések

A legegyszerűbb alkérdéseket csak egyszer kerülnek kiértékelésre, majd az eredményt egy magasabb rendű lekérdezés hasznosíthatja. A beágyazott alkérdéseket bonyolultabb módon is lehet használni, úgy hogy az alkérdés többször legyen kiértékelve. Minden egyes kiértékelés megfelel egy olyan értékadásnak, mely az alkérdésen kívüli sorváltozóból származik. Egy ilyen típusú alkérdést *korrelált* alkérdésnek nevezünk. Vizsgáljuk meg egy példán keresztül.

5.19. példa: Keressük meg azokat a filmcímeket, melyek két vagy több filmhez is tartoznak. Kezdjük a megoldást egy külső lekérdezéssel, mely a következő reláció minden sorát megvizsgálja:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

Minden egyes sor esetén egy alkérdésen keresztül megvizsgáljuk, hogy létezik-e egy ugyanolyan című film, melyet később gyártottak (feltesszük, hogy azonos című filmek nem készültek ugyanabban az évben). Az egész lekérdezést az 5.8. ábra szemlélteti.

Mint a többi beágyazott lekérdezés esetén is, kezdjük a vizsgálatot a 4)–6) sorok közötti belső lekérdezéssel. Ha a `Régi.cím` komponenst a 6) sorban egy konstanssal cserélnénk ki, mint például a `'King Kong'`, a lekérdezés a *'King Kong'* című filmek gyártási évét keresné. Az aktuális lekérdezés ettől kicsit különbözik. Az egyedüli probléma az, hogy nem tudjuk, hogy mi a `Régi.cím` értéke. Amikor azonban az 1)–3) sorok közötti külső lekérdezés folyamán a `Film` sorokon haladunk végig, a `Régi.cím`-nek mindig van valamilyen értéke. Ezt az értéket használva végrehajtjuk a 4)–6) sorok közötti lekérdezést, azért, hogy a 3)–6) sorok közötti `WHERE` feltétel igazságértékét meghatározzuk.

A 3) sorbeli feltétel akkor lesz igaz, ha létezik olyan film, melynek a címe megegyezik a `Régi.cím` értékével és később gyártották, mint a `Régi` sorváltozó aktuális értékéhez tartozó filmet. Ez a feltétel igaz lesz mindaddig, amíg a `Régi` sorváltozó év értéke nem az utolsó év, amelyben egy olyan című filmet gyártottak. Tehát az 1)–3) sorok közötti lekérdezés eggyel kevesebbszer fog kiírni egy filmcímet, mint ahányszor ilyen című filmet gyártottak. Egy kétszer gyártott filmet egyszer fog az eredmény tartalmazni, egy háromszor gyártottat kétszer stb.[6] □

A korrelált lekérdezések használata közben figyelembe kell vennünk a nevek *érvényességi körére* vonatkozó szabályokat. Általában egy lekérdezésben szereplő attribútum a lekérdezés `FROM` záradékában szereplő valamelyik sorváltozóhoz tartozik, ha a sorváltozó relációja tartalmazza azt az attribútumot. Ha nem tartalmazza egyik sem,

[6] Ez a példa az első olyan alkalom, amikor egy éles különbséget tapasztalhatunk a relációs algebra relációi és az SQL-beli relációk között. Az SQL relációban lehetnek ismétlődések, az eredmény multihalmaz, nem pedig halmaz. Több módon is eltávolíthatók az ismétlődések, ahogy azt az 5.4. fejezetben vizsgálni fogjuk.


```
1) SELECT cím
2) FROM Film AS Régi
3) WHERE év < ANY
4)    (SELECT év
5)    FROM Film
6)    WHERE cím = Régi.cím
7)    );
```

5.8. ábra. *A több mint egyszer szereplő filmcímek megkeresése*

akkor megvizsgáljuk az őt körbevevő lekérdezést, azután az azt körbevevőt stb. Így a 4) sorban szereplő `év` és a 6) sorban szereplő `cím` annak a sorváltozónak az attribútumai, amely az 5) sorban bevezetett `Film` reláció sorain megy végig – azaz annak a `Film` relációnak a másolatán, amelyet a 4)–6) sorok közötti lekérdezés vizsgál.

Egy attribútumot azonban egy másik sorváltozóhoz is kapcsolhatunk, ha a sorváltozó nevét és egy pontot írunk az attribútum elé. Ezért vezettük be a külső lekérdezésben a `Film` reláció `Régi` másodnevét, és ezért hivatkoztunk a 6) sorban a `Régi.cím` komponensre. Jegyezzük meg, hogy ha a 2) és 5) sorokban található `FROM` záradékok relációi nem egyeztek volna meg, akkor nem lett volna szükség egy másodnévre. Ehelyett az alkérdésben nyugodtan használhattuk volna a 2) sorbeli reláció attribútumait.

5.3.5. Feladatok

5.3.1. feladat: A 4.1.1. feladat adatbázissémáját használva, adjuk meg a következő lekérdezéseket:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

A válaszokban használjunk legalább egy alkérdést, és minden lekérdezésre adjunk két különböző megoldást (azaz használjunk különböző operátorokat az `EXISTS`, `IN`, `ALL` és `ANY` közül).

* a) Keressük meg a legalább 160-as sebességű PC-k gyártóit.

b) Keressük meg a legdrágább nyomtatókat.

! c) Keressük meg azokat a laptopokat, amelyek minden PC-nél lassúbbak.

! d) Keressük meg a modellszámát a legdrágább terméknek (PC, laptop vagy nyomtató).

! e) Keressük meg a legolcsóbb színes nyomtató gyártóját.

!! f) Keressük meg annak a nyomtatónak a gyártóját, amely a leggyorsabb processzorú PC-t gyártja a legkisebb memóriájú PC-k között.

5.3.2. feladat: Adjuk meg a következő lekérdezéseket, a 4.1.3. feladat adatbázissémájára vonatkozóan:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

A lekérdezés tartalmazzon legalább egy alkérdést és mindegyik feladathoz adjunk meg két megoldást (azaz különböző operátorokat használjunk az EXISTS, IN, ALL és ANY közül).

a) Keressük meg a legtöbb ágyúval rendelkező hajók országait.

*! b) Keressük meg azokat a hajóosztályokat, melyekből legalább egy hajót elsüllyesztettek egy csatában.

c) Keressük meg azokat a hajókat, melyek ágyúinak kalibere 16 hüvelyk.

d) Keressük meg azokat a csatákat, melyekben a Kongó hajóosztályba tartozó hajók vettek részt.

!! e) Keressük meg azokat a hajókat, melyeknek az ágyúszáma a legnagyobb a velük megegyező kaliberű ágyúkat tartalmazó hajók között.

! **5.3.3. feladat:** Adjuk meg az 5.8. ábra lekérdezését alkérdések nélkül.

! **5.3.4. feladat:** Tekintsük a $\pi_L (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)$ relációs algebrai kifejezést, ahol L csak az R_1 attribútumait tartalmazza. Mutassuk meg, hogy ezt a lekérdezést át lehet írni SQL-be csak alkérdések felhasználásával. Pontosabban, adjunk meg egy olyan SQL kifejezést amelyben mindegyik FROM záradékban csak egy sorváltozó található.

! **5.3.5. feladat:** Adjuk meg a következő lekérdezéseket metszet és különbség használata nélkül:

* a) Az 5.3. ábra lekérdezését.

b) Az 5.15. ábra lekérdezését.

!! **5.3.6. feladat:** Észrevehettük, hogy néhány SQL operátor felesleges, abban az értelemben, hogy más operátorokkal helyettesíthető. Például láthattuk, hogy s IN R helyettesíthető az s = ANY R kifejezéssel. Mutassuk meg, hogy az EXISTS és NOT EXISTS operátorok feleslegesek, azaz olyan kifejezéssel helyettesíthetőek, melyben nincs EXISTS. *Tanács:* A SELECT záradékban konstans is használható.

5.4. Ismétlődő sorok

Az eddigi relációs műveletek a relációkat soronként vizsgálták, kivéve az 5.2.5. alfejezetben bemutatott egyesítés, metszet és különbség operátorokat. A következő két fejezetben olyan operátorokat vizsgálunk, melyek a relációt mint egy egységes egészt kezelik. Figyelembe kell vennünk azt a tényt, hogy az SQL a relációkat nem tekinti halmaznak, azaz ugyanaz a sor többször is előfordulhat. Az 5.4.1. alfejezetben bemutatjuk, hogyan kell az ismétlődéseket megszüntetni, míg az 5.4.2. alfejezetben ismertetjük, hogyan kell az ismétlődések megőrzését biztosítani.

5.4.1. Ismétlődések megszüntetése

Ahogy azt az 5.3.4. alfejezetben is említettük, az SQL relációi különböznek a 3. fejezetben absztrakt módon definiált relációtól. A reláció, halmazként tekintve, nem tartalmazhatja ugyanazt a sort többször. Amikor azonban az SQL létrehoz egy relációt, nem szünteti meg automatikusan az ismétlődéseket. Így az SQL egy lekérdezésre adott válasza tartalmazhatja ugyanazt a sort többször is.

Elevenítsük fel az 5.2.4. alfejezetben említetteket: az SQL select-from-where műveletének egyik értelmezése szerint a FROM záradékban szereplő relációk Descartes-szorzatát számoljuk ki először. Minden egyes így megalkotott sorra ellenőrizzük a WHERE feltételt, majd a megfelelő sorok a SELECT záradéknak megfelelően levetítve az eredménybe kerülnek. Ez a vetítés azt eredményezheti, hogy több különböző sorból ugyanaz az eredménysor keletkezik. Ezenkívül, mivel a FROM záradékban lévő SQL relációk is tartalmazhatnak ismétlődéseket, ezeket az ismétlődéseket párosítva a többi reláció soraival, újabb ismétlődéseket kaphatunk az eredményben.

Ha szeretnénk megszüntetni az ismétlődéseket az eredményben, a SELECT kulcsszó után a DISTINCT kulcsszót kell írnunk. Ez a kulcsszó jelzi az SQL-nek, hogy minden egyes sor csak egyszer szerepeljen az eredményben. Az SQL biztosítja, hogy az eredmény ismétlődésmentes lesz.

5.20. példa: Tekintsük az 5.7. ábra lekérdezését, melyben a Harrison Ford-filmek gyártásirányítóit kerestük, alkérdések nélkül. A lekérdezés jelenlegi formájában, George Lucas többször is szerepelni fog az eredményben, mivel több olyan filmet is gyártott, melyben Harrison Ford szerepelt. Ha azt szeretnénk, hogy minden gyártásirányító csak egyszer szerepeljen, a lekérdezés 1) sorát a következőre kell kicserélni:

```
1) SELECT DISTINCT név
```

Így a gyártásirányítók listájából az eredmény kiírása előtt az ismétlődések kitörlődnek.

Valószínűleg az 5.5. ábra lekérdezése nem fog az eredményben ismétlődéseket tartalmazni. Igaz ugyan, hogy a 4) sor lekérdezése többször is tartalmazhatja George Lucas azonosítószámát, de az 1) sor „fő" lekérdezésében minden `GyártásIrányító` sort csak egyszer vizsgálunk. Elvileg ebben a relációban George Lucas csak egyszer szerepel, és ez az egyedüli sor, amely a 3) sor `WHERE` feltételének eleget tesz. Így George Lucas csak egyszer fog szerepelni az eredményben. □

5.4.2. Ismétlődések kezelése halmazműveletek során

A `SELECT` utasítással ellentétben, amely csak akkor szünteti meg az ismétlődéseket, ha jelen van a `DISTINCT` kulcsszó, az egyesítés, metszet és különbség operátorok normális esetben megszüntetik az ismétlődéseket. Azért, hogy az ismétlődések megmaradjanak, az `UNION, EXCEPT` vagy `INTERSECT` kulcsszavak után az `ALL` kulcsszót kell írni. Ebben az esetben ezek az operátorok a 4.6.2. alfejezetben tárgyalt multihalmaz szemantika szerint fognak működni.

5.21. példa: Tekintsük ismét az 5.16. példa lekérdezését, de adjuk hozzá az `ALL` kulcsszót:

```
(SELECT cím, év FROM Film)
   UNION ALL
(SELECT filmCím AS cím, év AS év FROM SzerepelBenne);
```

Így minden egyes filmcím és gyártási év annyiszor fog szerepelni, ahányszor a `Film` és a `SzerepelBenne` relációkban összesen megjelenik. Például, ha egy film egyszer található meg a `Film` relációban és három színész tartozik hozzá a `SzerepelBenne` relációban, akkor az egyesítés eredményében a film négyszer fog szerepelni. □

Hasonlóan az egyesítéshez, az `INTERSECT ALL` és az `EXCEPT ALL` műveletek is multihalmazok felett dolgoznak. Tehát, ha R és S két reláció, akkor a

Az ismétlődések megszüntetésének költsége

Hajlamosak lennénk minden `SELECT` után a `DISTINCT` kulcsszót írni, mivel hiba nem keletkezhet belőle. Valójában azonban nagyon költséges az ismétlődések megszüntetése egy relációban. Általában a relációt sorba kell rendezni, hogy a megegyező sorok egymás mellé kerüljenek. Csak a sorok ilyen jellegű csoportosításán keresztül tudjuk azt eldönteni, hogy egy adott sor többször szerepel-e. Sokszor az az idő, amíg a relációt rendezzük, hosszabb mint magának a lekérdezésnek a végrehajtása. Tehát az ismétlődések megszüntetését megfelelő módon kell használni, ha azt szeretnénk, hogy a lekérdezéseink gyorsak legyenek.

R `INTERSECT ALL` S

kifejezés eredménye az a reláció, amelyben egy t sor előfordulásainak száma egyenlő a t sor R-beli és S-beli előfordulásai számának a minimumával.

Továbbá, az

R `EXCEPT ALL` S

kifejezés eredménye az a reláció, amelyben egy t sor előfordulásainak száma egyenlő a t sor R-beli előfordulásainak számából kivonva a t sor S-beli előfordulásainak számát, ha az eredmény pozitív. Ezek a definíciók megegyeznek a 4.6.2. alfejezetben elméletileg tárgyalt definíciókkal.

5.4.3. Feladatok

5.4.1. feladat: Adjuk meg a 4.1.1. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

5.4.2. feladat: Adjuk meg a 4.1.3. feladatban szereplő lekérdezéseket SQL-ben, biztosítva az ismétlődések megszüntetését.

! **5.4.3. feladat:** Az 5.3.1. feladat mindegyik lekérdezésénél döntsük el, hogy az eredmény tartalmazhat-e ismétlődéseket, és ha igen, írjuk át a lekérdezést úgy, hogy ismétlődések ne szerepeljenek benne. Ha nem tartalmaznak ismétlődéseket, akkor adjunk meg egy alkérdések nélküli olyan lekérdezést, mely ugyanazt az ismétlődésmentes eredményt adja.

! **5.4.4. feladat:** Ugyanaz a feladat, mint az 5.4.3. feladatban, ezúttal azonban az 5.3.2. feladat lekérdezéseire vonatkozóan.

5.5. Összesítések

Az egységként tekintett relációk fölött elvégzett műveleteknek egy másik osztálya az, amelyek egy oszlopban összesítéseket eredményeznek. *Összesítés* alatt egy olyan műveletet értünk, amely egy oszlop értékeiből egy értéket hoz létre. Ilyen például egy oszlop értékeinek átlaga vagy összege. Az SQL nemcsak annak a lehetőségét biztosítja, hogy több oszlopot összesítsünk, hanem csoportosíthatjuk a reláció egyes sorait bizonyos feltétel szerint, mint például egy oszlop értéke, és csoporton belül is végezhetünk összesítéseket.

5.5.1. Összesítő függvények

Az SQL öt olyan operátort biztosít, amelyek egy oszlop értékeinek bizonyos fajta összesítését végzik el. Ezek az operátorok a következők:

1. SUM, az oszlop értékeinek összegét határozza meg.
2. AVG, az oszlop értékeinek átlagát határozza meg.
3. MIN, az oszlop értékeinek minimumát határozza meg.
4. MAX, az oszlop értékeinek maximumát határozza meg.
5. COUNT, az oszlopban található elemek számát határozza meg (beleértve az ismétlődéseket is, ha azok nincsenek megszüntetve a DISTINCT kulcsszóval).

Ezeket az operátorokat egy skalár értékre alkalmazzuk, általában egy SELECT záradékbeli oszlopra.

5.22. példa: A következő lekérdezés megadja az összes gyártásirányító átlagos nettó bevételét:

```
SELECT AVG(nettóBevétel)
FROM GyártásIrányító;
```

Figyeljük meg, hogy nincs feltétel a sorokra, így a WHERE záradékot el lehet hagyni. A lekérdezés a következő reláció nettóBevétel oszlopát vizsgálja:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Összeadja az oszlopban található értékeket, mindegyik sor esetén egy értéket (akkor is, ha a sor egy másik sor ismétlődése), majd az összeget elosztja a sorok számával. Ha nincsenek ismétlődések, a lekérdezés az átlagos nettó bevételt eredményezi, ahogy szerettük volna. Ha vannak ismétlődések, akkor annak a gyártásirányítónak a nettó bevételét, aki *n*-szer szerepel a relációban, *n*-szer fogja az összeghez hozzáadni. □

5.23. példa: A következő lekérdezés a GyártásIrányító reláció sorainak számát határozza meg:

```
SELECT COUNT(*)
FROM GyártásIrányító;
```

Feltéve, hogy a név kulcs a GyártásIrányító relációban és a relációban nincsenek ismétlődések, a lekérdezés az adatbázisban lévő gyártásirányítók számát adja meg.

A * csak a COUNT függvényben használható, a többi operátor esetében nincs is értelme alkalmazni őket teljes sorokra.

Ha biztosak akarunk lenni benne, hogy minden sort csak egyszer számolunk meg, akkor használhatjuk a DISTINCT kulcsszót:

```
SELECT COUNT(DISTINCT név)
FROM GyártásIrányító;
```

Ez akkor is jó, ha a név nem kulcs (azaz két különböző sor ugyanarra a gyártásirányítóra vonatkozik, vagy két gyártásirányítónak azonos neve van), mivel ez a lekérdezés minden nevet csak egyszer vesz figyelembe. □

5.5.2. Csoportosítás

Néha nem egyszerűen egy oszlop összesítésére van szükségünk, hanem a reláció sorait kell csoportosítanunk egy vagy több oszlop értékei szerint. Például, szeretnénk meghatározni mindegyik stúdió által előállított filmpercek összegét, stúdiónként. Ezért csoportosítanunk kell a Film reláció sorait a stúdióNév szerint, majd csoportonként ki kell számolni a hossz összegét. Azt is szeretnénk, hogy az eredmény táblázat alakjában adja meg a stúdióneveket és a hozzájuk rendelt percösszegeket, ilyen formában: □

stúdióNév	*SUM(hossz)*
Disney	12345
MGM	54321
...	...

Az eredmény elérése érdekében egy GROUP BY záradékot használunk, a WHERE záradék után. A GROUP BY kulcsszavakat *csoportosító* attribútumok listája követi. A legegyszerűbb esetben a FROM záradék csak egy sorváltozót tartalmaz, és a reláció sorait csoportosítjuk a csoportosító attribútumoknak megfelelően. A SELECT záradékban szereplő összesítési operátorokat a csoportokra kell alkalmazni.

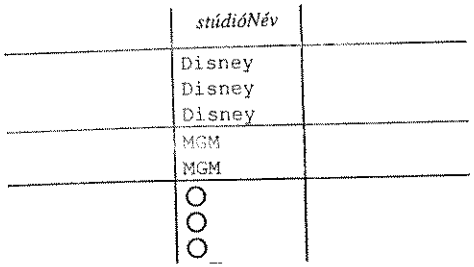
5.24. példa: A Film relációból:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

szeretnénk megtudni stúdiónként a gyártott percek összegét. Ezt a következő lekérdezés fejezi ki:

```
SELECT stúdióNév, SUM(hossz)
FROM Film
GROUP BY stúdióNév;
```

A lekérdezés értelmezése úgy képzelhető el, hogy a Film reláció sorait átszervezzük úgy, hogy az összes sor, amely a Disney-stúdióra vonatkozik, egymás mellett van,



5.9. ábra. *Egy reláció elképzelt sorcsoportosítása*

ugyanígy egymás mellett vannak az MGM-re vonatkozó sorok stb. A `hossz` komponensek összegét csoportonként kiszámítjuk és minden csoportra az eredménybe kerül a stúdió neve és a hozzá tartozó összeg. □

Figyeljük meg, hogy az 5.24. példában a SELECT záradékban kétféle elem található:

1. Összesítések, melyekben egy összesítési operátort alkalmazunk egy attribútumra vagy egy attribútumot tartalmazó kifejezésre. Ezek a kifejezéseket csoportonként kerülnek kiértékelésre.

2. Attribútumok, melyek a GROUP BY záradékban szerepelnek, mint a példában `stúdióNév`. Egy összesítéseket tartalmazó SELECT záradékban csak a GROUP BY záradékban is megtalálható attribútumok jelenhetnek meg összesítési operátor nélkül.

A GROUP BY záradékot tartalmazó lekérdezésekben általában csoportosító attribútumok és összesítések is vannak a SELECT záradékban, de elvileg nincs akadálya olyan lekérdezést írni, melyben valamelyik ezek közül hiányzik. Például a következő lekérdezés is helyes:

```
SELECT stúdióNév
FROM Film
GROUP BY stúdióNév;
```

Ez a lekérdezés csoportosítja a sorokat `stúdióNév` szerint, majd minden csoport esetén kiírja a `stúdióNév` értékét. Azaz a lekérdezés ugyanazt eredményezi, mint a következő:

```
SELECT DISTINCT stúdióNév
FROM Film;
```

A GROUP BY záradékot többrelációs lekérdezésben is használhatjuk. Egy ilyen lekérdezés kiértékelése a következő módon történik:

1. A FROM és WHERE záradékokból kialakul egy R reláció. Az R relációt úgy kapjuk, hogy a FROM záradékban lévő relációk Descartes-szorzatára alkalmazzuk a WHERE feltételt.

2. R sorait csoportosítjuk a GROUP BY attribútumainak megfelelően.

3. Eredményként a SELECT záradék attribútumai és összesítései jelennek meg csoportonként úgy, mintha a lekérdezés az R relációra történt volna.

5.25. példa: Tételezzük fel, hogy szeretnénk meghatározni mindegyik gyártásirányító által gyártott filmek összes hosszát. A következő két relációra van szükségünk:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Tekintsük a théta-összekapcsolásukat, egyenlővé téve az azonosítószámokat. Ez a lépés egy olyan relációt hoz létre, melyben minden `GyártásIrányító` sort összepárosítja az összes olyan `Film` sorral, amelyet az a gyártásirányító készített. Ezután a relációt csoportosíthatjuk a gyártásirányító azonosítószáma szerint, és minden csoportban meghatározhatjuk a filmek hosszainak összegét. A lekérdezést az 5.10. ábra szemlélteti: □

```
1) SELECT név, SUM(hossz)
2) FROM GyártásIrányító, Film
3) WHERE producerAzon = azonosító
4) GROUP BY név;
```

5.10. ábra. *Gyártásirányítónként a filmek hosszainak összege*

5.5.3. HAVING záradék

Tételezzük fel, hogy az 5.25. példában nem szeretnénk mindegyik gyártásirányítót figyelembe venni. A sorokra olyan feltételt tehetnénk előzőleg, hogy a csoportosítás során egyes csoportok üresek legyenek. Például, ha olyan gyártásirányítókat szeretnénk vizsgálni, akiknek a nettó bevétele nagyobb, mint 10 000 000 $, akkor a 3) sort kicserélhetnénk:

```
3) WHERE producerAzon = azonosító AND
                        nettóBevétel >= 10000000
```

Néha azonban a csoportokat a csoport bizonyos összesített tulajdonsága alapján szeretnénk kiválogatni. Ilyenkor a GROUP BY záradék után a HAVING záradékot írhatjuk. A záradék a HAVING kulcsszóból és egy feltételből áll, mely a csoportra vonatkozik.

```
SELECT név, SUM(hossz)
FROM GyártásIrányító, Film
WHERE producerAzon = azonosító
GROUP BY név
HAVING MIN(év) < 1930;
```

5.11. ábra. *Az első gyártásirányítók filmhosszainak összege*

5.26. példa: Keressük meg azokat a gyártásirányítókat és filmhosszaik összegét, akik legalább egy 1930 előtti filmet is készítettek. Az 5.10. ábrához csatolhatjuk a következő sort:

```
HAVING MIN(év) < 1930
```

A lekérdezés csak azokat a csoportokat veszi majd figyelembe, melyekben legalább egy 1930 előtti film található. □

5.5.4. Feladatok

5.5.1. feladat: Adjuk meg a következő lekérdezéseket, a 4.1.1. feladat adatbázisára vonatkozóan:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

és szemléltessük az eredményt a 4.1.1. feladat adataival.

* a) Keressük meg a PC-k átlagos sebességét.

b) Keressük meg a 2500 $-nál drágább laptopok átlagos sebességét.

c) Keressük meg az „A" gyártó által gyártott PC-k átlagos árát.

A záradékok sorrendje az SQL lekérdezésekben

Most már megismertük az összes záradékot melyek az SQL „select-from-where" lekérdezésében szerepelhetnek: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` és `ORDER BY`. Ezek közül csak az első kettő kötelező. A többi közül bármelyik szerepelhet, de csak a fenti sorrendben.

! d) Keressük meg a „D" gyártó által gyártott PC-k és laptopok átlagos árát.

e) Keressük meg minden egyes PC-sebességhez az ilyen sebességű PC-k átlagos árát.

*! f) Keressük meg minden gyártó esetén a laptopok átlagos képernyőméretét.

! g) Keressük meg azokat a gyártókat, akik legalább háromfajta PC-t gyártanak.

! h) Keressük meg minden gyártó esetén a maximális PC-árat.

*! i) Keressük meg minden 150-nél nagyobb sebességű PC átlagos árát.

!! j) Keressük meg minden olyan gyártóhoz, akik nyomtatót gyártanak, a PC-k átlagos merevlemezméretét.

5.5.2. feladat: Adjuk meg a következő SQL lekérdezéseket, melyek a 4.1.3. feladat adatbázissémájára vonatkoznak:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

és a lekérdezések eredményeit szemléltessük a 4.1.3. példa adataival.

a) Keressük meg a hajóosztályok számát.

b) Keressük meg a hajóosztályok átlagos ágyúszámát.

! c) Keressük meg a hajók átlagos ágyúszámát. Figyeljük meg a különbséget b) és c) között: melyik esetben súlyoztuk a hajóosztályokat a hajóik számával?

! d) Keressük meg minden hajóosztály esetén azt az évet, amikor az ebbe az osztályba tartozó első hajót felavattak.

! e) Keressük meg minden hajóosztály esetén azon hozzá tartozó hajók számát, amelyeket csatában elsüllyesztettek.

!! f) Keressük meg a legalább három hajóból álló osztályokra a csatában elsüllyesztett hajók számát.

!! g) Egy ágyú által kilőtt golyó súlya (fontokban) körülbelül akkora, mint a kaliber (hüvelykben) köbének a fele. Keressük meg minden országra a hozzá tartozó hajók ágyúgolyóinak átlagos súlyát.

5.6. Változtatások az adatbázisban

Eddig a pontig a select-from-where SQL utasításra koncentráltunk. Léteznek további SQL utasítások is, melyek nem egy eredményt adnak vissza, hanem változtatásokat hoznak létre, megváltoztatják az adatbázis állapotát. Ebben e fejezetben három utasítástípust ismertetünk, melyek a következő hatással járnak:

1. Sorok beszúrása egy relációba.
2. Bizonyos sorok törlése egy relációból.
3. Bizonyos létező sorok meghatározott komponenseinek módosítása.

Az ilyen típusú műveleteket általában *változtatásoknak* nevezzük.

5.6.1. Beszúrás

A beszúrási utasítás legegyszerűbb formája a következő részekből áll:

1. Az `INSERT INTO` kulcsszavak,
2. Az R reláció neve,
3. Az R reláció attribútumainak listája zárójelek között,
4. A `VALUES` kulcsszó és
5. Egy sorkifejezés, amely a 3) sorban található lista minden eleméhez megad egy konkrét értéket.

Tehát, a beszúrási művelet legegyszerűbb alakja:

`INSERT INTO` $R(A_1, \ldots, A_n)$ `VALUES` $(v_1, \ldots, v_n)$;

A sor úgy keletkezik, hogy az A_i attribútumhoz v_i értéket rendelünk, $i = 1, 2, \ldots, n$. Ha az attribútumlista nem tartalmazza R összes attribútumát, akkor a hiányzó attribútumok az alapértelmezés szerinti értéket kapják. Az alapértelmezés szerinti értékeket az 5.7.5. alfejezetben mutatjuk be. A leggyakrabban használt alapértelmezett érték a nullérték (`NULL`), melyet a 4.7.4. alfejezetben vezettünk be és az 5.9. alfejezetben az SQL környezetben is vizsgálni fogjuk. Egyelőre a `NULL`-t tekinthetjük úgy, mint annak a jelölését, hogy a komponens helyes értéke ismeretlen.

5.27. példa: Tételezzük fel, hogy *Kate Winslet*-et szeretnénk az *Titanic* szereplőinek listájába felvenni. A megfelelő utasítás:


```
1) INSERT INTO SzerepelBenne(filmCím, év, színészNév)
2) VALUES('Titanic', 1998, 'Kate Winslet');
```

Az utasítás hatása, hogy egy sort, melynek három komponense a 2) sorban található, beszúrunk a `SzerepelBenne` relációba. Mivel a `SzerepelBenne` reláció minden attribútumát felsoroltuk az 1) sorban, nincs szükség alapértelmezett értékekre. A (2) sorban található értékeket összepárosítjuk az 1) sorban felsorolt megfelelő attribútumokkal, így például a `filmCím` értéke `'Titanic'` lesz stb. □

Ha az 5.27. példához hasonlóan a reláció minden attribútumának megadjuk az új értékét, az attribútumlistát elhagyhatjuk. Azaz, az utasítás így alakul:

```
INSERT INTO SzerepelBenne
VALUES('Titanic', 1998, 'Kate Winslet');
```

Ebben az esetben azonban nagyon kell vigyáznunk arra, hogy az értékek sorrendje megegyezzen az attribútumok relációbeli sorrendjével. Az 5.7. alfejezetben bemutatjuk majd, hogy hogyan lehet relációsémát definiálni, és látni fogjuk, hogy a definiálás közben megadunk egy bizonyos sorrendet az attribútumokra. Ezt a sorrendet kell figyelembe vennünk, amikor az `INSERT` utasításban nincs megadva az attribútumlista. Ha nem vagyunk biztosak az attribútumok sorrendjében, jobb, ha felsoroljuk őket a nekünk megfelelő sorrendben.

A feljebb ismertetett legegyszerűbb `INSERT` utasítás csak egy sort szúr be a relációba. A sorhoz megadott értéklista helyett egy alkérdés segítségével meghatározhatunk több beszúrandó sort is. Ez az alkérdés helyettesíti a `VALUES` kulcsszót és az `INSERT` utáni sorkifejezést.

5.28. példa: Tételezzük fel, hogy a következő relációba

```
Stúdió(név, cím, elnökAzon)
```

be szeretnénk szúrni a `Film` relációban megtalálható összes olyan stúdiót, melyek azonban a `Stúdió` relációban nem szerepelnek. A `Film` reláció sémája:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

Mivel nem tudjuk ezen stúdiók címét és elnökének azonosítóját, ezek az attribútumok nullértéket kapnak. Az utasítást az 5.12. ábra szemlélteti:

Mint a legtöbb SQL utasítást, az 5.12. ábra utasítását is legkönnyebb belülről kifelé értelmezni. Az 5)–6) sorok az összes `Stúdió` relációbeli stúdiónevet eredményezik. Így a 4) sor azt ellenőrzi, hogy a `Film` sor stúdióneve ne legyen ezen stúdiók között.

Tehát a 2)–6) sorok azokat a stúdióneveket eredményezik, melyek a `Film`-ben benne vannak, de a `Stúdió`-ban nincsenek benne. A 2) sorban a `DISTINCT` azt biztosítja, hogy ebben a halmazban mindegyik stúdió csak egyszer szerepel, függetlenül

```
1) INSERT INTO Stúdió(név)
2)    SELECT DISTINCT stúdióNév
3)    FROM Film
4)    WHERE stúdióNév NOT IN
5)       (SELECT név
6)       FROM Stúdió);
```

5.12. ábra. *Új stúdiók beszúrása*

attól, hogy hány filmet gyártottak. Az 1) sor beszúrja ezeket a stúdiókat, nullértéket ad a `cím` és `elnökAzon` komponenseknek. □

5.6.2. Törlés

Egy törlési utasítás a következő részekből áll:

1. A `DELETE FROM` kulcsszavak,
2. A reláció neve, például *R*,
3. A `WHERE` kulcsszó, és
4. Egy feltétel.

Tehát a törlés alakja:

`DELETE FROM` *R* `WHERE` <feltétel>;

Az utasítás azt eredményezi, hogy az *R* relációból kitörlődik minden olyan sor, amely megfelel a feltételnek.

5.29. példa: A következő relációból:

```
SzerepelBenne(filmCím, év, színészNév)
```

kitörölhetjük azt a tényt, hogy Kate Winslet játszott a *Titanic*ban a következő SQL utasítás segítségével:

```
DELETE FROM SzerepelBenne
WHERE filmCím = 'Titanic' AND
   év = 1998 AND
   színészNév = 'Kate Winslet';
```

Figyeljük meg, hogy az 5.27. példa beszúrási műveletével ellentétben csak egy `WHERE` feltétel segítségével tudjuk megadni, hogy melyik sort akarjuk törölni. □

A beszúrások időzítése

Az 5.12. ábra alapján az SQL utasítások különböző kiértékelését is szemléltetni tudjuk. Elvileg, a 2)–6) sorok közötti lekérdezést az 1) sor beszúrása előtt kellene elvégezni. Így az 1) sorban beszúrt új stúdiók nem lehetnek hatással a 4) sor feltételére. Elképzelhető azonban olyan megvalósítása az SQL-nek, mely a hatékonyság érdekében rögtön beszúrja az új stúdiókat, amint megtalálta őket.

Ebben a példában nincs jelentősége, hogy a beszúrások késleltetve vannak-e addig, amíg a lekérdezés teljesen kiértékelése megtörténik. Lehetnek azonban olyan utasítások, melyeknek az eredménye különbözne a beszúrások időzítésétől függően. Például tételezzük fel, hogy az 5.12. ábra 2) sorából kivesszük a `DISTINCT` kulcsszót. Ha a 2)–6) sorok közötti lekérdezést teljesen kiértékeljük a beszúrások előtt, akkor egy új stúdiónév, amely többször szerepel a `Film` relációban, az eredményben is többször szerepelne, tehát a `Stúdió` relációba is többször kerülne bele. Viszont ha minden egyes megtalált stúdiót rögtön beszúrnánk, egyik sem lenne többször beszúrva a `Stúdió` relációba. Ennek az az oka, hogy mivel az új stúdiónevet már beszúrtuk, a név nem fogja kielégíteni a 4)–6) sorok közötti feltételt, tehát a 2)–6) sorok közötti lekérdezés nem fogja ismét eredményként jelezni.

5.30. példa: Egy újabb példa a törlésre. Ezúttal töröljünk ki a következő relációból

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

több olyan sort egyszerre, melyek megfelelnek egy bizonyos feltételnek:

```
DELETE FROM GyártásIrányító
WHERE nettóBevétel < 10000000;
```

Ezzel az utasítással kitöröljük a relációból a kis bevételű – tízmillió dollár alatti – gyártásirányítókat. □

5.6.3. Módosítás

A beszúrást és a törlést tulajdonképpen tekinthetjük módosításoknak, de valójában a *módosítás* az SQL-ben az adatbázis egy speciális változtatása, melyben egy vagy több létező sor bizonyos komponenseinek értékét megváltoztatjuk. A módosítás általános formája a következő részekből áll:

1. Az `UPDATE` kulcsszó,

Beszúrások, törlések és ismétlődések

Van néhány kényes pont azzal kapcsolatban, hogy a törlések és beszúrások hogyan viszonyulnak az SQL-ben megengedett ismétlődésekhez. Először is, egy beszúrás hozzáad a relációhoz új sorokat, függetlenül attól, hogy a sorok előtte léteztek-e a relációban vagy sem. Tehát ha például a `SzerepelBenne` relációban létezett egy sor, amely a *Titanic*ra és Kate Winsletre vonatkozott, akkor az 5.27. utasítás ennek a sornak egy ismétlődését szúrná be. Ha a beszúrás után törölnénk az 5.29. utasítás segítségével, akkor mindkét sor megfelelne a feltételeknek, tehát mindkettő kitörlődne. Egy érdekes következtetés tehát, hogy ez a beszúrás-törlés kombináció a `SzerepelBenne` relációt nem az utasítások előtti állapotban hagyja. Továbbá a törlés, mely látszólag egy sort töröl, tulajdonképpen kettőt fog kitörölni. Tulajdonképpen az SQL-ben nincs lehetőség arra, hogy két megegyező sor közül csak az egyiket töröljük ki.

2. A reláció neve, például *R*,
3. A `SET` kulcsszó,
4. Egy kifejezéslista, melyek közül *R* mindegyik attribútumát egyenlővé teszi egy kifejezéssel vagy egy konstanssal,
5. A `WHERE` kulcsszó, és
6. Egy feltétel.

Tehát a módosítás általános alakja:

`UPDATE` *R* `SET` <értékadások> `WHERE` <feltétel>;

Mindegyik értékadás egy attribútumból, az egyenlőségjelből és egy kifejezésből áll. Ha több mint egy értékadás van, vesszővel kell őket elválasztani.

Az utasítás eredményeképpen az összes olyan *R*-beli sorban, melyek megfelelnek a 6) feltételnek, a 4) értékadáslistának megfelelően a komponensek módosulnak.

5.31. példa: Módosítsuk a következő relációt:

```
GyártásIrányító(név, cím, azonosító, nettóBevétel),
```

a név elé téve az `'Ig.'` rövidítést minden olyan gyártásirányító esetén, aki elnöke (igazgatója) egy stúdiónak. A feltétel az, hogy az azonosító szerepeljen valamelyik `Stúdió` sorban az `elnökAzon` komponensben. A megfelelő utasítás:


```
1) UPDATE GyártásIrányító
2) SET név = 'Ig.' || név
3) WHERE azonosító IN (SELECT elnökAzon FROM
                       Stúdió);
```

A 3) sor azt ellenőrzi, hogy a gyártásirányító azonosítója megegyezik-e a `Stúdió` reláció valamelyik sorának `elnökAzon` komponensével.

A 2) sor végrehajtja a módosítást a megfelelő sorokon. Emlékezzünk vissza, hogy a `||` operátor karaktersorok összeillesztését jelenti, tehát az = utáni kifejezés az `'Ig. '` karaktersort helyezi a `név` komponens régi értéke elé. Az így kapott karaktersor lesz a `név` komponens új értéke. □

5.6.4. Feladatok

5.6.1. feladat: Adjuk meg a következő adatbázis-módosításokat a 4.1.1. feladat adatbázissémájára vonatkozóan, és a feladat adataival szemléltessük a változást az adatbázisban:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez,
       képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

a) Két `INSERT` utasítás segítségével tároljuk az adatbázisban azt a tényt, hogy az 1100-as PC modellt a C gyártó gyártja, 240 a sebessége, a memóriája 32, merevlemeze 2,5, a CD 12× és az ára 2499 $.

! b) Szúrjuk be az adatbázisba azt, hogy minden egyes PC esetén létezik egy vele megegyező sebességű, memóriájú, merevlemezű laptop, melynek 11 hüvelykes képernyője van, modellszáma 1100-zal nagyobb és 500 $-ral drágább.

c) Töröljük ki a 2 gigabájtnál kisebb merevlemezű PC-ket.

d) Töröljük ki az összes olyan laptopot, melyeket olyan cég gyárt, amely nem gyárt nyomtatókat.

e) Az A cég megveszi a B céget. Módosítsuk a B által gyártott termékeket olyan módon, hogy ezentúl az A gyártja őket.

f) Minden egyes PC esetén kétszerezzük meg a memória nagyságát, és adjunk egy gigabájtot a merevlemez méretéhez. (Ne feledjük, hogy egyetlen `UPDATE` utasítás segítségével több attribútum is módosítható.)

! g) Az E cég által gyártott laptopok esetén adjunk egy hüvelyket a képernyő méretéhez és vonjunk ki 100 $-t az árból.

5.6.2. feladat: Adjuk meg a következő adatbázis-módosításokat, a 4.1.3. feladat adatbázissémájára vonatkozóan, és a feladatnak az adataival szemléltessük a változást az adatbázisban:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

* a) A Nelson osztályba tartozó két brit csatahajót – a Nelson és a Rodney – 1927-ben avatták fel, mindkettőnek 16 hüvelykes kaliberű ágyúja van, és 34 000 tonna a vízkiszorításuk. Szúrjuk be ezeket az adatokat az adatbázisba.

b) A Vittorio Veneto hajóosztályba tartozó három olasz csatahajóból – a Vittorio Veneto és az Italia – 1940-ben avattak fel kettőt; a Rómát pedig, a harmadik hajót, 1942-ben. Mind a háromnak 15 hüvelykes kaliberű ágyúja van és 41 000 tonnás. Szúrjuk be ezeket az adatokat az adatbázisba.

* c) Töröljük ki a `Hajók` relációból az összes elsüllyesztett hajót.

* d) Módosítsuk a `Hajóosztályok` relációt úgy, hogy a kalibert centiméterben (1 hüvelyk=2,5 centiméter), és vízkiszorítást metrikus tonnában (1 metrikus tonna = 1,1 tonna) adjuk meg.

e) Töröljük ki a háromnál kevesebb hajóból álló osztályokat.

5.7. Relációsémák definiálása SQL-ben

A következő részben az *adatdefinícióval* foglalkozunk majd, azaz az SQL-nek azon részeivel, melyek az adatbázisbeli információk struktúráját írják le. Az eddig leírt SQL utasítások – lekérdezések és változtatások – az *adatok kezelésével* foglalkoztak.

A fejezet témája a relációsémák definiálása. Bemutatjuk majd hogyan kell egy új relációt – SQL terminológiával *táblát* – leírni. Definiálni tudjuk az attribútumneveket, az attribútumok adattípusát és néhány korlátozott függőséget, mint például a kulcsokat. Az 5.8. alfejezet bemutatja a „nézettáblákat", amelyek virtuális relációk, melyek valójában nincsenek tárolva az adatbázisban. Néhány bonyolultabb problémát a függőségekkel kapcsolatban a 6. fejezet fog tárgyalni.

5.7.1. Adattípusok

Bevezetésképpen bemutatjuk az SQL rendszerek által használt fő adattípusokat. Minden attribútumhoz kötelező megadni egy adattípust.

1. Rögzített vagy változó hosszúságú karaktersorok. A `CHAR(n)` típus egy rögzített hosszúságú karaktersort jelöl. Tehát, ha egy attribútum adattípusa `CHAR(n)`, akkor minden sorban az attribútumnak megfelelő komponens egy *n* hosszúságú karaktersor. A `VARCHAR(n)` egy legfeljebb *n* hosszúságú karaktersort jelöl. Az ilyen típusú attribútumoknak megfelelő komponensek olyan karaktersorok lesznek, melyeknek hossza 0 és *n* között van. Az SQL rugalmasságot biztosít a különböző karaktertípusok között. Ha egy *k* hosszúságú karaktersor egy olyan komponens értékévé válik, amelynek a típusa egy rögzített hosszúságú karaktersor, melynek a hossza több mint *k*, akkor a karaktersor végére üres karaktereket illesztünk, hogy a megfelelő hosszúságot kitöltsék. Például, ha a `'labda'` karaktersor egy `CHAR(8)` típusú komponens értékévé válik, akkor az érték `'labda  '` lesz (azaz három üres karakter kerül be a végére). A kitöltő üres karaktereket figyelmen kívül maradnak, amikor a karaktersort összehasonlítjuk (lásd 5.1.3. alfejezet) egy másik karaktersorral.

2. Rögzített vagy változó hosszúságú bitsorok. Ezek hasonlóak a rögzített és a változó hosszúságú karaktersorokhoz, csak nem karakterekből, hanem bitekből állnak. A `BIT(n)` típus egy *n* hosszúságú bitsort, míg a `BIT VARYING(n)` egy legfeljebb *n* bitből álló bitsort jelképez.

3. Az `INT` vagy más néven `INTEGER` típusok, egész számokat jelképeznek. A `SHORTINT` típus is egész számot jelképez, de kevesebb biten tároljuk, ez a bitszám megvalósítástól függő (ugyanúgy mint az `int` és `short int` a C-ben).

4. A lebegőpontos értékeket többféleképpen is tárolhatjuk. Használhatjuk a `FLOAT` és a `REAL` típusokat (amelyek megegyeznek). Nagyobb pontossággal tárolhatók az értékek a `DOUBLE PRECISION` típusban; a különbségek ismét a C-hez hasonlóak. Az SQL-ben vannak olyan típusok is, amelyek fixpontos valós számok. Például, a `DECIMAL(n,d)` olyan értékeket tárol, amelyek *n* számjegyből állnak, és a tizedesponttól jobbra *d* számú tizedesjegy áll. Így a 0123,45 érték típusa `DECIMAL(6,2)`.

5. Dátumokat és időket a `DATE` és `TIME` típusok jelképeznek. Emlékezzünk vissza ezen típusok vizsgálatára az 5.1.4. alfejezetben. Ezek az értékek tulajdonképpen speciális alakú karaktersorok. A dátum és idő értékeket átkonvertálhatjuk karaktersorrá, és visszafelé is, ha a karaktersor értelmes idő, illetve dátum értéket képvisel.

5.7.2. Táblák létrehozása

A legegyszerűbb módja egy tábla létrehozásának a `CREATE TABLE` kulcsszavakból, a relációnévből, az attribútumoknak és típusaiknak zárójelek közé tett listájából áll.

5.32. példa: A 3.9. alfejezet `FilmSzínész` reláció sémáját az 5.13. ábrán látható SQL utasítással hozhatjuk létre.

```
1) CREATE TABLE FilmSzínész (
2)    név CHAR(30),
3)    cím VARCHAR(255),
4)    nem CHAR(1),
5)    születésnap DATE
   );
```

5.13. ábra. *A `FilmSzínész` reláció sémájának definiálása*

Az első két attribútum, a `név` és a `cím` típusa karaktersor. A különbség abban áll, hogy a `név` attribútum egy 30 karakterből álló rögzített hosszúságú karaktersor, melyet szükség esetén üres karakterekkel töltünk ki, illetve lecsonkítjuk a végét, ha hosszabb. Ezzel szemben a címet egy változó hosszúságú karaktersorba helyeztük, melynek maximális hossza 255 karakter.[7] Nem biztos, hogy ez a két választás a legjobb, csak azért használtuk ezeket hogy bemutassuk a kétféle karaktersor használatát.

A `nem` attribútum értékei lehetnek 'N' és 'F', tehát egyetlen karakter. Így nyugodtan használhatjuk az egy hosszúságú karaktersort típusként. Végül a `születésnap` attribútum értéke természetesen `DATE`. Ha ez a típus nem létezne a rendszerben, mert a rendszerünk nem felel meg az SQL2 szabványnak, akkor nyugodtan használhatnánk a `CHAR(10)` típust, hiszen egy angol dátum 10 karakterből áll: nyolc számjegy és két gondolatjel. □

5.7.3. Táblák megszüntetése

Egy hosszú életű adatbázis esetén a létrehozott táblákat éveken keresztül töltjük fel adatokkal, melyeket lehet, hogy később többször is módosítunk. A relációt *nagy tömegben is feltölthetjük,* felhasználva olyan adatokat, melyeket nem egy SQL tábla alakjában tároltak; elképzelhető, hogy az adatbázis-kezelőnk biztosít ilyen lehetőséget. Ezután a relációban naponta gyűlhetnek a sorok. Például a `Film` relációt feltölthetjük egy korábbi adatbázisból, majd `INSERT` utasítások segítségével beszúrjuk az új adatokat, minden egyes új film megjelenésekor.

[7] A 255-ös szám nem egy valamilyen bűvös szám, minden címnek ilyen hosszúnak kell lennie. Egy bájton 0 és 255 közötti számokat lehet tárolni, tehát egy változó hosszúságú karaktersort, melynek maximális hossza 255, úgy lehet tárolni, hogy egy bájton tároljuk a karakterek számát és magát a karaktersort annyi bájton, amennyi szükséges. A kereskedelmi rendszerek általában hosszabb változó hosszúságú karaktersorokat is megengednek.

Néha azonban szükségünk van arra, hogy megszüntessünk egy táblát az adatbázissémában. A feltételek megváltozása miatt lehet, hogy felesleges az adatok karbantartása ebben a táblában. Az is lehetséges, hogy a reláció csak ideiglenes, valamely bonyolult közbülső lekérdezés eredményeképpen. Ilyen esetben az R relációt a következő SQL utasítással szüntethetjük meg:

```
DROP R;
```

5.7.4. Relációsémák módosítása

Egy hosszú életű adatbázis esetében sokkal többször kell egy tábla struktúráját módosítani, mint egy táblát megszüntetni. A megfelelő utasítások az `ALTER TABLE` kulcsszavakkal és a reláció nevével kezdődnek. Ezután több lehetőségünk is van, a legjelentősebb azonban a következő kettő:

1. `ADD`, egy oszlopnév és adattípus.

2. `DROP` és egy oszlopnév.

5.33. példa: Módosítsuk a `FilmSzínész` relációt, hozzáadva egy `telefon` attribútumot:

```
ALTER TABLE FilmSzínész ADD telefon CHAR(16);
```

Az utasítás eredményeképpen, a `FilmSzínész` relációnak öt attribútuma lesz, az első négyet az 5.13. ábra bemutatta, az ötödik pedig a `telefon` attribútum, amely egy rögzített 16 hosszúságú karaktersor. A megváltozatott relációban minden sorban szerepelni fog a `telefon` komponens, de mivel még nincsenek telefonszámok megadva, ezért ezen komponensek értéke nullérték lesz. Az 5.7.5. alfejezetben ismertetni fogjuk, hogyan lehet egy másik alapértelmezett értéket beállítani a nullérték helyett.

Egy másik példaként töröljük ki a `születésnap` attribútumot:

```
ALTER TABLE FilmSzínész DROP születésnap;
```

□

5.7.5. Alapértelmezés szerinti értékek

Amikor létrehozunk vagy módosítunk sorokat, néha egyes komponensek értékét nem ismerjük. Az előző példában a relációsémához hozzáadtunk egy új oszlopot, és a létező sorokban nincs meg az új oszlop értéke. Ott azt javasoltuk, hogy a komponens értéke nullérték legyen. Az 5.28. példában új sorokat szúrtunk be a `Stúdió` relációba,

anélkül hogy a cím és az elnökAzon attribútumok értékét ismertük volna. Ismét szükség lenne egy olyan értékre, mely kifejezi azt, hogy „nem tudom" a komponens valódi értékét.

A probléma megoldására az SQL bevezeti a NULL értéket. Ez az érték kerül bele azokba a komponensekbe, melyeknél nincs érték megadva, kivéve olyan speciális helyzeteket, amikor a NULL érték nem megengedett (lásd a 6.2. alfejezetet). Néha azonban szeretnénk egy másik alapértelmezett értéket használni úgy, hogy ezt az értéket kapja a komponens, ha nem tudjuk a valódi értékét.

Általában, amikor egy attribútumot és az adattípusát definiáljuk, hozzáadhatjuk a DEFAULT kulcsszót és egy megfelelő értéket. Az érték vagy nullérték vagy egy konstans. Egyéb olyan értékeket is használhatunk, melyeket a rendszer rendelkezésünkre bocsát, mint például az aktuális idő.

5.34. példa: Tekintsük az 5.32. példát. Szeretnénk alapértelmezésként a '?' karaktert használni a nem attribútum esetében és a '0000-00-00' dátumot a születésnap esetében. Így az 5.13. ábra 4)–5) sorát a következőkre kell kicserélni:

```
4)   nem CHAR(1) DEFAULT '?',
5)   születésnap DATE DEFAULT DATE '0000-00-00'
```

Az 5.33. példában a telefon attribútum alapértelmezett értéke legyen 'nem ismert' szöveg:

```
ALTER TABLE FilmSzínész ADD telefon CHAR(16)
      DEFAULT 'nem ismert';
```

□

5.7.6. Értéktartományok

Eddig minden attribútum számára definiáltunk egy adattípust. Egy másik lehetőség az *értéktartomány* definiálása, amely új neve lesz egy adattípusnak. Az értéktartomány definíciójába belekerülhetnek további információk, mint az alapértelmezett érték és a függőségek, amiket a 6.3.3. alfejezetben fogunk tárgyalni. Attribútum definiálásakor később használhatjuk az értéktartományt típusként. Több attribútumnak is lehet ugyanaz az értéktartománya, ami több szempontból is hasznosan összeköti őket. Például, ha két attribútumnak ugyanaz az értéktartománya, akkor tudjuk, hogy az egyik bármikor felveheti a másik értékét.

Az értéktartomány definíciójának általános alakja CREATE DOMAIN kulcsszavakból, az értéktartomány nevéből, az AS kulcsszóból és az adattípusból áll. Tehát az értéktartomány definiálásának alakja:

```
CREATE DOMAIN <név> AS <típusleírás>;
```


Ezután következhet az alapértelmezés, majd a függőségek leírása, amelyeket a 6.3.3. alfejezetben fogunk tárgyalni.

5.35. példa: Definiáljunk egy új értelmezési tartományt a Film reláció cím attribútuma számára: legyen a neve FilmÉrtelmezésiTartomány. Ezt az értelmezési tartományt használhatjuk a Film reláció cím attribútuma számára, de ugyanúgy használhatjuk a SzerepelBenne reláció filmCím attribútuma számára. A következő módon lehet definiálni az értelmezési tartományt:

```
CREATE DOMAIN FilmÉrtelmezésiTartomány
AS VARCHAR(50) DEFAULT 'ismeretlen';
```

Így, a FilmÉrtelmezésiTartomány értékei változó hosszúságú karaktersorok, melyek hossza maximum 50 karakter, és az alapértelmezett értéke 'ismeretlen'.

Amikor létrehozzuk a Film reláció sémáját, a cím attribútumot így definiálhatjuk:

```
cím FilmÉrtelmezésiTartomány
```

amely ekvivalens a következővel:

```
cím VARCHAR(50) DEFAULT 'ismeretlen'
```

Hasonlóképpen, a SzerepelBenne reláció filmCím attribútumát így definiálhatjuk:

```
filmCím FilmÉrtelmezésiTartomány
```

□

Egy értelmezési tartomány alapértelmezett értékét a következő utasítással lehet módosítani:

```
ALTER DOMAIN FilmÉrtelmezésiTartomány
SET DEFAULT 'nincs ilyen cím';
```

Ez az utasítás az alapértelmezett 'ismeretlen' értéket kicseréli a 'nincs ilyen cím' karaktersorra. További módosításokat is végre lehet hajtani, például a függőségekkel kapcsolatban, melyekkel majd a 6. fejezetben foglalkozunk.

A következő utasítással törölhetünk egy értelmezési tartományt:

```
DROP DOMAIN FilmÉrtelmezésiTartomány;
```

Az utasítás hatásaként az értelmezési tartomány a továbbiakban nem használható attribútumok definiálására. Az eddig ilyen típusúra definiált attribútumok azonban megtartják típusukat és alapértelmezett értéküket.

5.7.7. Indexek

Egy reláció A attribútumára vonatkozó *indexe* egy olyan adatstruktúra, mely lehetővé teszi azon sorok gyors keresését, melyeknek az A attribútumon bizonyos rögzített értéke van. Az indexek általában olyan lekérdezésekben segítenek, melyekben az A attribútumot egy konstanssal hasonlítjuk össze, például $A = 3$ vagy $A \leq 3$. Amikor egy relációban már nagyon sok sor van, túl költségessé válik az összes sor végigvizsgálása bizonyos feltételt kielégítő egy (vagy nagyon kevés) sor megtalálása érdekében. Például tekintsük az első lekérdezést, melyet vizsgáltunk:

```
SELECT *
FROM Film
WHERE stúdióNév = 'Disney' AND év = 1990;
```

Elképzelhető, hogy van 10 000 `Film` sor, amelyből csak 200-at készítettek 1990-ben.

A legegyszerűbb végrehajtása a lekérdezésnek megvizsgálná mind a 10 000 sort és leellenőrizné a `WHERE` feltételt mindegyik sorra. Sokkal jobb lenne a lekérdezés hatásfoka, ha valamilyen módon ki tudnánk szűrni azt a 200 sort, amely az 1990-ben készített filmekre vonatkozik, majd ezek közül néznénk meg, hogy melyik készült a Disney-stúdióban. Még gyorsabb lenne, ha egyből meg tudnánk határozni azt a 10 körüli sort, amelyek mind a két feltételt kielégítik – azaz a stúdió Disney és a készítési év 1990 – de ez több, mint amit elvárhatunk az általában használt adatstruktúráktól.

Annak ellenére, hogy az indexlétrehozás nem része az SQL szabványnak, az SQL2-t is beleértve, azonban a legtöbb kereskedelmi rendszer biztosít eszközöket az indexek létrehozására. Tételezzük fel, hogy a `Film` relációra szeretnénk egy indexet definiálni az `év` attribútum szerint. Ezt általában a következő utasítással adhatjuk meg:

```
CREATE INDEX ÉvIndex ON Film(év);
```

Ennek eredményeképpen egy `ÉvIndex` nevű indexet hoz létre a rendszer a `Film` relációra az `év` attribútum szerint. Tehát azokat az SQL lekérdezéseket, melyek megadják az év értékét, az SQL processzor úgy fogja végrehajtani, hogy csak a megfelelő évben készített filmekre vonatkozó sorokat vizsgáljuk meg, így a végrehajtási idő lényegesen csökkenni fog.

Használhatunk *többattribútumos indexeket* is. Az ilyen index több attribútum értékét is vizsgálva keresi meg a megfelelő sorokat. Első látásra úgy tűnhet, hogy a többattribútumos indexek kevésbé használhatók, mint az egy attribútumra alapozottak, mivel csak akkor lehet használni őket, ha mindegyik attribútum értéke ismert. Mégis, egyes lekérdezések esetében a többattribútumos indexek sokkal nagyobb hatásfokot eredményeznek.

5.36. példa: Mivel a `cím` és az `év` jelentik a kulcsot a `Film` relációban, valószínű, hogy a lekérdezésekben vagy mindkettőt megadják, vagy egyiket sem. Így a következő utasítás segítségével definiálhatunk egy indexet a `Film` relációra a két attribútum szerint:


```
CREATE INDEX KulcsIndex ON Film(cím, év);
```

Mivel a (`cím`, `év`) páros kulcs, amikor megadjuk az értékeket, tudjuk hogy az index csak egy sort fog találni, és az lesz a keresett sor. Ha azonban csak az `ÉvIndex` létezik, akkor a rendszer mást nem tehet, mint azt, hogy az összes 1990-ben gyártott filmet végigvizsgálja, figyelve, hogy melyiket gyártotta a Disney-stúdió.

Ha az `ÉvIndex` helyett egy olyan indexünk lenne, amely `cím` szerint van, jobb helyzetben lennénk, mivel egy adott évben sok filmet gyártottak, de egy adott című filmből csak egy pár létezik. Ebben a speciális esetben, az adott című filmek lekérdezése, majd azok vizsgálata, hogy melyiket készítették az adott évben, csak kevéssel több időbe telik, mint a többattribútumos index használata esetén. □

Ha meg szeretnénk szüntetni egy indexet, a következő egyszerű utasítással tudjuk ezt elérni:

```
DROP INDEX ÉvIndex;
```

Az indexek használatakor az adatbázis tervezőjének több szempontot is figyelembe kell vennie:

- Egy bizonyos attribútum szerinti index nagymértékben növeli azon lekérdezések sebességét, melyekben az attribútum egy értékét ismerjük.
- Másrészt problémát jelent az, hogy minden egyes index növeli a beszúrások, törlések és módosítások bonyolultságát és időigényességét.

Az indexek kiválasztása az egyik legnehezebb feladat az adatbázis megtervezése során, ugyanis körülbelül ismerni kell, hogy az adatbázison milyen jellegű lekérdezéseket és változtatásokat fogunk végrehajtani. Ha egy relációt sokkal többször fogunk lekérdezni, mint módosítani, javasolt az indexek használata. Ha viszont a reláció gyakran lesz változtatva, akkor nagyon megfontoltan kell bánnunk az indexekkel. Ebben az esetben is lehet hatásfokot növelni egy olyan index definiálásával, mely egy gyakran használt attribútumra vonatkozik. Mivel egyes változtatási műveletek lekérdezéseket is jelentenek (például olyan `INSERT` esetén, amely select-from-where feltételt használ, vagy feltételt tartalmazó `DELETE` esetén), nagyon nehéz megjósolni, hogy milyen adatbázis-műveletek lesznek túlsúlyban.

5.7.8. Feladatok

*** 5.7.1. feladat:** A fejezetben megadtuk a `FilmSzínész` reláció definícióját. Adjuk meg az aktuális adatbázis többi relációjának definícióját:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

5.7.2. feladat: Tekintsük ismét a 4.1.1. feladat adatbázisát:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell,sebesség, memória, merevlemez, képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

Adjuk meg a következő definíciókat:

a) A `Termék` reláció megfelelő sémája.

b) A `PC` reláció sémája.

* c) A `Laptop` reláció sémája.

d) A `Nyomtató` reláció sémája.

e) Egy `ModellTípus` értelmezési tartomány definíciója, melynek értékei modellszámok. Mutassuk meg ennek az értelmezési tartománynak a használatát az a)–d) definíciókban.

* f) A c) pontbeli sémát módosítsuk úgy, hogy tartalmazzon egy `cd` mezőt, melynek alapértelmezett értéke legyen '`nincs`', abban az esetben ha a laptop nem tartalmaz CD olvasót.

g) A d) pontbeli `Nyomtató` séma módosítása úgy, hogy a `színes` attribútumot töröljük.

5.7.3. feladat: Tekintsük a 4.1.3. feladat adatbázisát:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
Csaták(név, dátum)
Kimenetelek(hajó, csata, eredmény)
```

Adjuk meg a következő definíciókat:

a) Egy megfelelő sémát a `Hajóosztályok` reláció számára.

b) A `Hajók` reláció sémája.

c) A `Csaták` reláció sémája.

d) A `Kimenetelek` reláció sémája.

e) Egy `HajóNevek` értelmezési tartomány definíciója, mely alkalmazható a hajók és a hajóosztályok neveként is. Módosítsuk az a), b), c) sémákat úgy, hogy tartalmazzák ezt az értelmezési tartományt.

f) A b) pontban megadott `Hajók` reláció sémájának módosítása úgy, hogy tartalmazza a `készült` attribútumot, mely megadja azt a helyet, ahol a hajót készítették.

g) Az a) pontban megadott `Hajók` reláció módosítása úgy, hogy töröljük ki a `kaliber` attribútumot.

! **5.7.4. feladat:** Magyarázzuk meg a különbséget a következő két utasítás között:

```
DROP R;
DELETE FROM R;
```

5.8. Nézettáblák

A `CREATE TABLE` segítségével definiált táblák fizikailag léteznek az adatbázisban, azaz egy SQL utasítás valamilyen fizikai struktúrában tárolja a táblákat. Megtartják az állapotukat, azaz nem változnak addig, amíg egy `INSERT` vagy valamilyen változtatási utasítás nem kényszeríti őket állapotuk megváltoztatására.

Létezik az SQL relációknak egy másik típusa, amit *nézettáblának* nevezünk, amelyek nem léteznek fizikailag az adatbázisban. Ehelyett egy lekérdezéshez hasonló kifejezés segítségével definiáljuk. A nézettáblákat lekérdezhetjük ugyanúgy, mint a fizikailag létező táblákat, és egyes esetekben még módosíthatjuk is őket.

5.8.1. Nézettáblák létrehozása

A legegyszerűbb mód egy nézettábla létrehozására a következő:

1. A `CREATE VIEW` kulcsszavak,
2. A nézettábla neve,
3. Az `AS` kulcsszó, és

Relációk, táblák és nézettáblák

Az SQL programozók gyakran használják a „tábla" szót a „reláció" szó helyett. Az ok abban rejlik, hogy fontos különbséget tenni a tárolt relációk, azaz a „táblák" és a virtuális relációk, a „nézettáblák" között. Most, hogy már ismerjük a különbséget a tábla és a nézettábla között, a „reláció" megnevezést csak ott fogjuk használni, ahol a táblát és a nézettáblát is használhatnánk. Amikor hangsúlyozni akarjuk, hogy egy reláció tárolt, akkor az „alapreláció" vagy „alaptábla" kifejezéseket fogjuk használni.

Létezik egy harmadik típusú reláció is, amely nem nézettábla és nincs fizikailag tárolva. Ezek az ideiglenes relációk, melyek valamely lekérdezés eredményeképpen jöttek létre. Ezekről is „relációként" fogunk említést tenni.

4. Egy *Q* lekérdezés. Ez a lekérdezés definiálja a nézettáblát. Amikor a nézettáblát lekérdezzük, az SQL úgy viselkedik, mintha a *Q* lekérdezést hajtanánk végre, és annak eredményét adja vissza.

Tehát a nézettábla definiálásának alakja:

```
CREATE VIEW <név> AS <definíció>;
```

5.37. példa: Tételezzük fel, hogy egy olyan nézettáblát szeretnénk, mely a

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

reláció egy részét jelképezi, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét:

```
1) CREATE VIEW ParamountFilm AS
2)    SELECT cím, év
3)    FROM Film
4)    WHERE stúdióNév = 'Paramount';
```

Az 1) sorban látható, hogy a nézettábla neve `ParamountFilm`. A nézettábla attribútumait a 2) sor tartalmazza: `cím` és `év`. A nézettábla definíciója a 2)–4) sorok között található meg. □

5.8.2. Nézettáblák lekérdezése

A `ParamountFilm` reláció nem tartalmaz sorokat a szokásos értelemben. Amikor a `ParamountFilm` relációt lekérdezzük, akkor a sorokat a `Film` relációból olvassuk ki, hogy a lekérdezést meg lehessen válaszolni. Tehát, ha egymás után kétszer lekérdezzük a `ParamountFilm` relációt, akkor lehet, hogy különböző eredményt kapunk, annak ellenére hogy a `ParamountFilm` relációban nem változtattunk semmit, ugyanis az alaptábla adatai megváltozhattak közben.

5.38. példa: Úgy kérdezhetjük le a `ParamountFilm` relációt, mintha egy tárolt reláció lenne:

```
SELECT cím
FROM ParamountFilm
WHERE év = 1979;
```

A `ParamountFilm` nézettábla definíciója segítségével a fenti lekérdezés átalakul egy olyan lekérdezéssé, amely csak a `Film` táblára vonatkozik. Az 5.8.5. alfejezetben bemutatjuk, hogyan lehet átalakítani egy nézettábla lekérdezését az alaptábla lekérdezésévé. Ebben az egyszerű példában azonban könnyű belátni mit jelent ennek a nézettáblának a lekérdezése. A `ParamountFilm` a `Film` relációtól csak két dologban különbözik:

1. A `ParamountFilm` csak a `cím` és az `év` attribútumokat tartalmazza.

2. A `stúdióNév = 'Paramount'` feltétel része minden olyan `WHERE` feltételnek, mely egy `ParamountFilm`mel kapcsolatos lekérdezésben található.

Mivel a mi lekérdezésünk csak a `cím` attribútumban érdekelt, 1) nem jelent problémát. A 2) érdekében csak a `stúdióNév = 'Paramount'` feltételt kell a lekérdezésünk `WHERE` feltételéhez hozzáadni. Ezután a `Film` relációt használhatjuk a `FROM` feltételben a `ParamountFilm` helyett, mivel biztosítottuk, hogy a lekérdezésünk értelme ugyanaz lesz. Így a következő lekérdezés:

```
SELECT cím
FROM Film
WHERE stúdióNév = 'Paramount' AND év = 1979;
```

ekvivalens a `ParamountFilm`re vonatkozó lekérdezésünkkel. Jegyezzük meg, hogy az SQL rendszer feladata ez az átalakítás, csak az illusztráció kedvéért mutattuk meg, hogy hogyan történik. □

5.39. példa: Írhatunk olyan lekérdezéseket is, melyek nézettáblákat és alaptáblákat is tartalmaznak. Ilyen például a következő:

```
SELECT DISTINCT színészNév
FROM ParamountFilm, SzerepelBenne
WHERE cím = filmCím AND
        ParamountFilm.év = SzerepelBenne.év;
```

Ez a lekérdezés a Paramount által gyártott filmekben szereplő összes színészt megadja. Figyeljük meg, hogy a DISTINCT kulcsszó biztosítja azt, hogy mindegyik színész csak egyszer szerepeljen eredményben, még akkor is, ha több Paramount által gyártott filmben is szerepelt. □

5.40. példa: Tekintsünk egy sokkal bonyolultabb nézettábla-definíciót. A célunk egy olyan FilmProd reláció definiálása, melyben a filmcímek és gyártásirányítóik találhatók. A nézetet definiáló lekérdezés használja a következő két relációt:

```
Film(év, cím, színes, stúdióNév, producerAzon)
```

amely a gyártásirányító azonosítószámát tartalmazza, és a

```
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

relációt, ahol összekapcsoljuk az azonosítót a névvel. A megfelelő utasítás:

```
1) CREATE VIEW FilmProd AS
2)   SELECT Film.cím, név
3)   FROM Film, GyártásIrányító
4)   WHERE producerAzon = azonosító;
```

Ezt a nézettáblát ugyanúgy lekérdezhetjük, mintha egy tárolt reláció lenne. Például, ha az *Elfújta a szél* gyártásirányítóját keressük:

```
SELECT név
FROM FilmProd
WHERE cím = 'Elfújta a szél';
```

A lekérdezés úgy kerül kiértékelésre, mintha egy vele ekvivalens lekérdezés lenne a megfelelő táblákra:

```
SELECT név
FROM Film, GyártásIrányító
WHERE Film.cím = 'Elfújta a szél'
                 AND producerAzon = azonosító;
```

□

5.8.3. Attribútumok átnevezése

Néha szeretnénk más attribútumneveket használni a nézettábla definíciójában, mint amelyek a lekérdezésből adódnak. A nézettábla attribútumait megadhatjuk egy zárójelek közé írt lista formájában, melyet a nézettábla neve után kell írni. Például az 5.40. példa nézettáblájának definícióját a következőképpen lehetne átírni:


```
CREATE VIEW FilmProd(filmCím, producerNév) AS
  SELECT Film.Cím, név
  FROM Film, GyártásIrányító
  WHERE producerAzon = azonosító;
```

A nézettábla ugyanaz lesz, attól eltekintve, hogy az oszlopnevek nem cím és név, hanem filmCím és producerNév lesz.

5.8.4. Adatok módosítása nézettáblákon keresztül

Korlátozott módon lehetséges beszúrni, törölni vagy változtatni az adatokat egy nézettáblán keresztül. Első látásra teljesen értelmetlen ötletnek tűnik, mivel a nézettábla nem létezik abban a formában, amelyben egy alaptábla (tárolt reláció) létezik. Mit jelent például egy új sor beszúrása egy nézettáblába? Hol tároljuk és honnan emlékezzen majd arra az adatbázis-kezelő, hogy a sor a nézettáblába került?

A legtöbb nézettábla esetén nem engedjük meg az ilyen beszúrást. Eléggé egyszerű nézettáblák esetén azonban a nézettábla módosítását átalakíthatjuk egy alaptábla módosításává, mely ugyanolyan hatású lesz, mintha a nézettáblát módosítanánk. Az ilyen nézettáblákat *módosítható nézettábláknak* nevezzük. Az SQL2 szabvány formálisan leírja, mikor lehet egy nézettáblát módosítani és mikor nem. Az SQL2 szabályok bonyolultak, de nagyjából azt engedik meg, hogy egy *R* reláción (amely szintén lehet módosítható nézettábla) definiált nézettáblát módosíthassunk, ha a SELECT után DISTINCT nem szerepel. Két fontos kikötés van:

- A WHERE záradékban *R* nem szerepelhet egy alkérdésben sem.
- A SELECT záradék elég attribútumot kell tartalmazzon ahhoz, hogy egy beszúrás esetén a többi attribútumot nullértékkel, vagy az alapértelmezett értékkel tölthessük fel az alaptáblában.

5.41. példa: Tételezzük fel, hogy a ParamountFilm nézettáblába szeretnénk egy sort beszúrni a következő módon:

```
INSERT INTO ParamountFilm
VALUES ('Csillagok háborúja');
```

A ParamountFilm nézettábla majdnem megfelel az SQL2 módosíthatósági feltételeinek, mivel a következő alaptábla néhány attribútumára vonatkozik:

```
Film(cím, év, színes, hossz, stúdióNév, producerAzon)
```

Az egyedüli probléma az, hogy mivel a stúdióNév attribútum nem része a nézettáblának, a Film táblába beszúrandó sorban az értéke NULL lesz, 'Paramount' helyett.

Így azért, hogy a nézettáblát módosíthatóvá alakítsuk át, a `stúdióNév` attribútumot hozzáadjuk a `SELECT` záradékhoz, annak ellenére, hogy az értéke nyilvánvalóan Paramount lesz. A `ParamountFilm` nézettábla új definíciója tehát:

```
1) CREATE VIEW ParamountFilm AS
2)   SELECT stúdióNév, cím, év
3)   FROM Film
4)   WHERE stúdióNév = 'Paramount';
```

Ezután a módosítható `ParamountFilm` nézettáblába beszúrjuk a kívánt sort:

```
INSERT INTO ParamountFilm
VALUES ('Paramount', 'Csillagok háborúja', 1979);
```

Hogy a beszúrást végrehajthassuk, létrehozunk egy olyan `Film` sort, amelyből a kívánt nézettábla sora keletkezik, amikor a nézettábla definíciójának megfelelő lekérdezés végrehajtásra kerül. A fenti sajátos beszúrás esetén a `stúdióNév` komponens értéke `'Paramount'`, a `cím` komponens értéke `'Csillagok háborúja'`, és az `év` komponens értéke 1979.

A nézettáblában nem szereplő attribútumoknak – `hossz`, `színes` és `producerAzon` – is léteznünk kell a beszúrt `Film` sorban. Az értéküket azonban nem tudjuk meghatározni. Tehát, az új `Film` sor ezen attribútumaiban vagy nullérték lesz benne, vagy a megfelelő alapértelmezett érték, melyet az attribútum vagy az értelmezési tartománya számára definiáltunk. Például, ha a `hossz` attribútum alapértelmezett értéke 0, a másik két attribútum pedig a `NULL` értéket használja alapértelmezettként, az elkészített `Film` sor ilyen alakú lesz:

cím	*év*	*hossz*	*színes*	*stúdióNév*	*producerAzon*
`'Csillagok háborúja'`	1979	0	NULL	`'Paramount'`	NULL

□

Törölhetünk is egy módosítható nézettáblából. A módosításhoz hasonlóan a törlés is az alaptáblán keresztül történik, olyan sorok kitörlése által, melyek megjelennek a nézettáblában.

5.42. példa: Tételezzük fel, hogy a módosítható `ParamountFilm` nézettáblából szeretnénk kitörölni az összes olyan filmet, melyek címe tartalmazza a `'háború'` szót. Ezt a következő utasítás segítségével érhetjük el:

```
DELETE FROM ParamountFilm
WHERE cím LIKE '%háború%';
```


A törlés átalakul egy `Film` táblára vonatkozó törlésre, azzal a különbséggel, hogy a `ParamountFilm` nézettáblát definiáló lekérdezés `WHERE` feltétele hozzáadódik a törlés `WHERE` feltételéhez:

```
DELETE FROM Film
WHERE cím LIKE '%háború%' AND stúdióNév = 'Paramount';
```

□

Hasonlóképpen, a módosítások is az alaptáblán keresztül történnek. A nézettábla módosításainak tehát az a hatása, hogy az alaptábla azon sorait módosítja, melyek a nézettáblában sort eredményeznek.

5.43. példa: A következő módosítás:

```
UPDATE ParamountFilm
SET év = 1979
WHERE cím = 'Csillagok háborúja film';
```

ilyen alakban hajtjuk végre:

```
UPDATE Film
SET év = 1979
WHERE cím = 'Csillagok háborúja film' AND
  stúdióNév = 'Paramount';
```

□

Létezik a nézettáblának még egy fajta módosítása, a megszüntetés. Ezt a változtatást akkor is végre lehet hajtani, ha a nézettábla nem módosítható. A megszüntetési utasítás:

```
DROP VIEW ParamountFilm;
```

Ez az utasítás kitörli a nézettábla definícióját, tehát utána már nem kérdezhetjük le és nem módosíthatjuk a nézettáblát. Ugyanakkor a nézettábla megszüntetése nincs kihatással az alaptábla soraira. Ellenben, a

```
DROP TABLE Film;
```

nem csak a `Film` táblát szünteti meg, hanem a `ParamountFilm` nézettáblát is használhatatlanná teszi, mivel egy nem létező `Film` relációra vonatkozik.

Miért nem módosíthatóak egyes nézettáblák

Tekintsük az 5.40. példában előforduló `FilmProd` nézettáblát, amely a filmcímeket párosítja a gyártásirányítókkal. Ez a nézettábla nem módosítható az SQL2 szabvány szabályai szerint, mivel két relációra vonatkozik: a `Film` és a `GyártásIrányító` relációkra. Tételezzük fel, hogy szeretnénk beszúrni a következő sort:

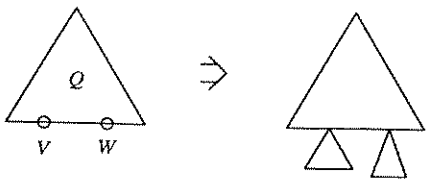
```
('Indiana Jones és a haláltemplom', 'George Lucas')
```

A `Film` és a `GyártásIrányító` relációkba is be kellene szúrni egy-egy sort. Az olyan attribútumokra, mint például a `hossz` vagy a `színes` használhatjuk az alapértelmezett értéket, de mit csináljunk a két egyenlővé tett attribútummal, a `producerAzon` és az `azonosító` attribútumokkal? Használhatnánk a `NULL` értéket számukra. Így azonban nem tudnánk összekapcsolni a két relációt, mert az SQL nem tekint két `NULL` értéket egyenlőnek (lásd az 5.9.1. alfejezetet). Tehát az `'Indiana Jones és a haláltemplom'` és a `'George Lucas'` nem kapcsolódna össze a `FilmProd` nézettáblában, tehát a beszúrás nem lenne helyesen végrehajtva.

5.8.5. Nézettáblákat érintő lekérdezések értelmezése

Könyvünknek nem célja annak bemutatása, hogyan hajtja végre az SQL rendszer a nézettábla lekérdezéseit, de jobban megérthetjük, mit jelent egy nézettábla, ha végigkövetjük az értelmezési folyamatot. Vizsgálatunkat olyan lekérdezésekre és nézettáblákra fogjuk korlátozni, melyeket ki lehet fejezni relációs algebrai kifejezésekkel, bár az SQL összes lehetőségeit használhatnánk, mint a csoportosítás és az összesítés.

Az alapötletet az 5.14. ábra szemlélteti. A *Q* lekérdezést a relációs algebrai kifejezésfa jelképezi. A kifejezésfa levelei nézettáblák. A rajzon két ilyen levelet adtunk meg, ezek a *V* és a *W*. Hogy *Q* értelmezését meghatározzuk az alaptáblák függvényében, meg kell határozzuk először a *V* és *W* nézettáblák definícióját. Ezeket a definíciókat szintén relációs algebrai kifejezésfaként adjuk meg.



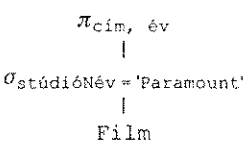
5.14. ábra. *Nézettáblák definícióinak kicserélése*

Az alaptáblákra vonatkozó lekérdezés meghatározásához a *Q* lekérdezés fájában minden egyes olyan levelet, ami nézettáblát jelképez, kicserélünk a nézettáblát definiáló fa gyökerére. Ezt szemléltettük az 5.14. ábrán, amikor a *V* és *W* nézettáblákat kicseréltük a definícióikkal. A kapott lekérdezés az alaptáblákra fog vonatkozni, és ekvivalens lesz az eredeti, nézettáblákra vonatkozó lekérdezéssel.

5.44. példa: Tekintsük az 5.38. példa nézettábláját és lekérdezését. A `ParamountFilm` nézettábla definíciója a következő:

```
1) CREATE VIEW ParamountFilm AS
2)   SELECT stúdióNév, cím, év
3)   FROM Film
4)   WHERE stúdióNév = 'Paramount';
```

Az 5.15. ábra bemutatja a nézettáblának a kifejezésfáját.



5.15. ábra. *A ParamountFilm nézettábla kifejezésfája*

Az 5.38. feladat lekérdezése:

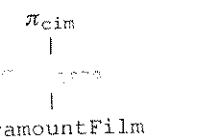
```
SELECT cím
FROM ParamountFilm
WHERE év = 1979;
```

mely az 1979-ben készült Paramount filmeket keresi. A lekérdezés kifejezésfáját az 5.16. ábra mutatja be. Figyeljük meg, hogy a fa egyik levele a `ParamountFilm` nézettábla.

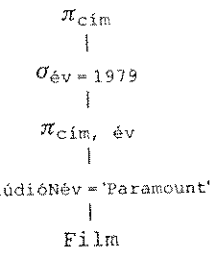
A lekérdezést tehát úgy értékeljük ki, hogy az 5.16. ábrában a `ParamountFilm`-re vonatkozó levelet kicseréljük az 5.15. ábra fájával. Az eredményt az 5.17. ábra szemlélteti.

Az 5.17. ábra egy lehetséges értelmezése a lekérdezésnek. Azonban feleslegesen bonyolult. Egy SQL rendszer normális esetben át tudja alakítani ezt a fát egy egyszerűbbre, mint például a következő:

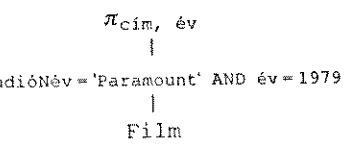
```
SELECT cím
FROM Film
WHERE stúdióNév = 'Paramount' AND év = 1979;
```



5.16. ábra. *A lekérdezés kifejezésfája*



5.17. ábra. *A lekérdezés kifejezése az alaptábla szerint*



5.18. ábra. *Az alaptáblára vonatkozó lekérdezés egyszerűsítése*

Például a $\pi_{\texttt{cím,év}}$ vetítést a $\sigma_{\texttt{év}=1979}$ kiválasztás fölé helyezhetjük. Ezt azért tehetjük meg, mert egy vetítés késleltetése nem változtathatja meg a kifejezés értelmét. Így egymás után két vetítésünk van, először a `cím` és az `év` szerint, majd csak a `cím` szerint. Nyilván az első vetítés teljesen felesleges, tehát kihagyható. A két vetítés helyett marad csupán a második, a `cím` szerinti.

A két kiválasztást szintén összevonhatjuk. Általában véve, két egymás utáni kiválasztást helyettesíteni lehet egyetlen kiválasztással, melyben a két feltételt AND operátorral kötjük össze. Az így kialakult kifejezésfát az 5.18. ábra szemlélteti. Ez a fa a következő lekérdezésnek felel meg:

```
SELECT cím
FROM Film
WHERE stúdióNév = 'Paramount' AND év = 1979;
```

□

5.8.6. Feladatok

5.8.1. **feladat:** A következő alaptáblákból kiindulva:

```
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

építsük fel a következő nézettáblákat:

* a) Egy `GazdagProducer` nézettábla, mely a legalább 10 000 000 $ nettó bevételű gyártásirányítók nevét, címét, azonosítószámát és bevételét adja meg.

b) Egy `StúdióElnök` nézettábla, mely azon gyártásirányítók nevét, címét és azonosítószámát tartalmazza, akik stúdióelnökök is.

c) Egy `ProdSzínész` nézettábla, mely azon személyek nevét, címét, nemét, születésnapját, azonosítószámát és nettó bevételét tartalmazza, akik egyben gyártásirányítók és színészek is.

5.8.2. feladat: Melyek módosíthatók az 5.8.1. feladat nézettáblái közül?

5.8.3. feladat: Alaptáblák használata nélkül adjuk meg a következő lekérdezéseket (használva az 5.8.1. feladat) nézettábláit:

a) Keressük meg azoknak a nőknek a neveit, akik gyártásirányítók és színészek is.

* b) Keressük meg azon gyártásirányítók neveit, akik egyben stúdióelnökök is és nettó bevételük legalább 10 000 000 $.

! c) Keressük meg azon stúdióelnökök nevét, akik egyben színészek is és nettó bevételük legalább 50 000 000 $.

*! **5.8.4. feladat:** Az 5.40. példa nézettáblájára vonatkozóan:

a) Ábrázoljuk a `FilmProd` nézettábla kifejezésfáját.

b) Ábrázoljuk a példabeli lekérdezés kifejezésfáját.

c) Az a) és b) pontban adott válaszok alapján építsük fel a lekérdezésnek egy olyan kiértékelését, mely alaptáblákat használ.

d) Mutassuk meg, hogyan lehet a c)-ben megadott kifejezést átalakítani egy olyan lekérdezéssé, mely ekvivalens az 5.40. példában javasolt megoldással.

! **5.8.5. feladat:** Az 5.8.3. feladat lekérdezéseire vonatkozóan fejezzük ki a nézettáblákat és a lekérdezést relációs algebrai kifejezéssel, helyettesítsük be a lekérdezés algebrai kifejezésébe a nézettábláknak megfelelő algebrai kifejezéseket, majd az eredményt egyszerűsítsük amennyire csak lehet. A kapott kifejezésnek megfelelően adjunk meg SQL lekérdezéseket az alaptáblákra vonatkozóan.

5.8.6. feladat: Felhasználva a következő alaptáblákat

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
```

a 4.1.3. feladatból.

a) Definiáljunk egy `BritHajók` nézettáblát, mely a Nagy-Britanniához tartozó hajók osztályát, típusát, ágyúszámát, kaliberét, vízkiszorítását és felavatási idejét adja meg.

b) Adjunk meg egy lekérdezést az a) pontban megadott nézettáblára vonatkozóan, mely tartalmazza azon brit csatahajók ágyúszámát és vízkiszorítását, melyeket 1919 előtt avattak fel.

! c) Fejezzük ki a b) pontbeli nézettáblát és az a) pontbeli lekérdezést relációs algebrai kifejezésekként, helyettesítsünk be a nézettábla helyett, majd egyszerűsítsük a kifejezést amennyire tudjuk.

! d) Adjunk meg egy SQL lekérdezést, amely a c) pontbeli kifejezésnek felel meg és a `Hajóosztályok` és `Hajók` alaptáblákra vonatkozik.

5.9. Nullértékek és külső összekapcsolások

Ebben a fejezetben további ismeretekre teszünk szert az SQL sorokban a `NULL` érték használatával kapcsolatban. Egy különösen fontos használati területe van a `NULL` értékeknek az olyan összekapcsolási művelet definiálásában, amelyben nem tűnnek el az olyan sorokra vonatkozó információk, amelyek nem kapcsolódnak össze egy másik sorral sem. Az ilyen típusú összekapcsolást „külső" összekapcsolásnak nevezzük. A külső összekapcsolásnak számos változatát fogjuk bemutatni ebben a fejezetben. Annak ellenére hogy a `NULL` megtalálható a korábbi SQL szabványokban is, a külső összekapcsolási műveletet általában csak az SQL2-nek megfelelő rendszerek tartalmazzák.

5.9.1. Nullértékeken értelmezett műveletek

Többször is láthattuk, hogy az SQL megenged egy speciális értéket, melyet `NULL`-nak nevezünk. A 4.7.4. alfejezet bemutatja a `NULL` néhány felhasználási területét, mint például egy ismeretlen vagy nem létező érték jelképezését. `NULL` értékek keletkeznek

Nullértékekkel kapcsolatos csapdák

Feltételezhetnénk, hogy a `NULL` egy olyan érték, amit nem ismerünk, de amely biztos létezik. A valóságban azonban ez a feltételezés nem helytálló. Például tételezzük fel, hogy x egy sor egy bizonyos komponense, melynek az értelmezési tartománya az egész számok. Úgy érvelhetnék, hogy $0 * x$ értéke biztos 0, mivel függetlenül attól, hogy mennyi az x értéke, a 0-val való szorzás eredménye 0. Mégis, ha x értéke `NULL`, akkor az 5.9.1. alfejezet 1) szabálya lép életbe, azaz 0 és `NULL` szorzata `NULL`. Hasonlóképpen azt hihetnénk, hogy $x - x$ értéke 0, függetlenül attól, hogy mennyi az x értéke. Mégis, ebben az esetben is az 1) szabály alapján az eredmény `NULL`.

például akkor, amikor egy új sort szúrunk be egy relációba, de nem adjuk meg a reláció minden attribútumának értékét. Ha nem létezik egy másik alapértelmezett érték azon attribútumok számára, melyeknek nem adtuk meg az értékét, akkor `NULL` értékű lesz a komponens értéke. Látni fogjuk, hogy a külső összekapcsolással is keletkeznek `NULL` értékek.

Két fontos szabályt kell figyelembe vennünk, amikor `NULL` értékekkel dolgozunk.

1. Amikor egy aritmetikai műveletben, mint az $\times$, vagy a $+$, legalább az egyik tag `NULL`, akkor az eredmény is `NULL`.

2. Amikor egy `NULL` értéket hasonlítunk össze bármely más értékkel, beleértve a `NULL`-t, egy összehasonlítási operátor segítségével, mint az $=$ vagy $>$, az eredmény `ISMERETLEN`. Az `ISMERETLEN` egy logikai érték, olyan mint az `IGAZ` és a `HAMIS`, hamarosan részletesen is bemutatjuk.

Annak ellenére, hogy a `NULL` megjelenhet értékként a sorokban, nem tekinthető konstansnak. Míg a fenti szabályok olyan esetben alkalmazandók, amikor egy kifejezés értéke `NULL`, a `NULL`-t nem használhatjuk direkt módon egy kifejezésben.

5.45. példa: Legyen x értéke `NULL`. Akkor $x + 3$ értéke is `NULL`. Ennek ellenére `NULL + 3` nem egy szabályos SQL kifejezés. Hasonlóképpen $x = 3$ logikai értéke `ISMERETLEN`, de a `NULL = 3` nem egy szabályos összehasonlítás az SQL-ben. □

A szabványos útja annak, hogy megtudjuk, hogy egy kifejezés értéke `NULL`-e vagy sem, az `x IS NULL` kifejezés segítségével történik. A kifejezés értéke `IGAZ`, ha x értéke `NULL` és `HAMIS`, ha nem. Hasonlóképpen `x IS NOT NULL` értéke `IGAZ`, ha x nem `NULL`.

5.9.2. Az ISMERETLEN igazságérték

Az 5.1.2. alfejezetben bemutattuk, hogy egy összehasonlítás eredménye vagy IGAZ vagy HAMIS és ezek a logikai értékek a szokásos logikai műveletekkel – AND, OR és NOT – kombinálhatók. Az 5.9.1. alfejezetben viszont azt ismertük meg, hogy amikor NULL érték is szerepel az összehasonlításban, akkor az eredmény egy harmadik igazságérték: az ISMERETLEN. A továbbiakban bemutatjuk hogyan működnek a logikai műveletek a háromértékű logikában.

A szabályt könnyű megjegyezni, ha úgy tekintjük, hogy az IGAZ értéke 1 (azaz teljes mértékben igaz), a HAMIS értéke 0 (azaz egyáltalán nem igaz) és az ISMERETLEN értéke 1/2 (az igaz és a hamis között).

1. Két logikai értékre alkalmazott AND eredménye a két érték minimuma. Tehát, *x* AND *y* értéke HAMIS, ha legalább az egyik hamis; ISMERETLEN, ha egyik sem HAMIS, de legalább az egyik ISMERETLEN; és IGAZ, ha mindkettő IGAZ.

2. Két logikai értékre alkalmazott OR eredménye a két érték maximuma. Tehát, *x* OR *y* értéke IGAZ, ha legalább az egyik IGAZ; ISMERETLEN, ha egyik sem IGAZ és legalább az egyik ISMERETLEN; és HAMIS, ha mindkettő HAMIS.

3. Egy logikai érték tagadásának értéke 1-ből kivonva a logikai értéket. Tehát NOT *x* értéke IGAZ, ha *x* HAMIS; HAMIS, ha *x* IGAZ; és ISMERETLEN, ha *x* értéke ISMERETLEN.

Az 5.19. ábra a három logikai művelet alkalmazásának eredményét mutatja be az *x* és *y* logikai változók különböző értékeire. Az utolsó művelet, a NOT, csak az *x* értékétől függ.

A select-from-where utasítások vagy a DELETE utasítások WHERE záradékában szereplő SQL feltételeket a rendszer leellenőrzi minden egyes sorra, és az ellenőrzés eredménye minden egyes sor esetén a három logikai érték (IGAZ, HAMIS, ISMERETLEN) valamelyike. Az eredménybe csak azok a sorok kerülnek bele, melyekre a feltétel kiértékelése IGAZ értéket hozott, azokat a sorokat melyekre a feltétel HAMIS

x	*y*	*x* AND *y*	*x* OR *y*	NOT *x*
IGAZ	IGAZ	IGAZ	IGAZ	HAMIS
IGAZ	ISMERETLEN	ISMERETLEN	IGAZ	HAMIS
IGAZ	HAMIS	HAMIS	IGAZ	HAMIS
ISMERETLEN	IGAZ	ISMERETLEN	IGAZ	ISMERETLEN
ISMERETLEN	ISMERETLEN	ISMERETLEN	ISMERETLEN	ISMERETLEN
ISMERETLEN	HAMIS	HAMIS	ISMERETLEN	ISMERETLEN
HAMIS	IGAZ	HAMIS	IGAZ	IGAZ
HAMIS	ISMERETLEN	HAMIS	ISMERETLEN	IGAZ
HAMIS	HAMIS	HAMIS	HAMIS	IGAZ

5.19. ábra. *Igazságértékek táblázata a háromértékű logika esetén*

vagy ISMERETLEN, kizárjuk az eredményből. Ez a helyzet egy újabb meglepő eredményhez vezet a nullértékekkel kapcsolatban, hasonlóan a „Nullértékekkel kapcsolatos csapdák" című bekeretezett részhez.

5.46. példa: Tételezzük fel, hogy a következő relációra

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
```

kipróbáljuk a következő lekérdezést:

```
SELECT *
FROM Film
WHERE hossz <= 120 OR hossz > 120;
```

Ösztönösen arra gondolhatnánk, hogy a Film reláció egy másolatát fogjuk kapni, mivel mindegyik film hossza kielégíti a feltételt.

Vizsgáljuk meg azonban azt az esetet, amikor léteznek olyan sorok, melyekben a hossz attribútum értéke NULL. Így a hossz > 120 és a hossz <= 120 feltételek kiértékelésének eredménye ISMERETLEN lesz. Az 5.19. ábra szerint két ISMERETLEN érték OR művelettel történő összekapcsolása ISMERETLEN értéket eredményez. Tehát minden olyan sor esetén, melyben a hossz attribútumban NULL érték van, a WHERE feltétel eredménye ISMERETLEN lesz. Az ilyen sorok *nem* kerülnek bele az eredménybe. Összefoglalva, a lekérdezés értelme „keressük meg az összes nem NULL hosszértékű Film sort". □

5.9.3. Összekapcsolások az SQL2-ben

Mielőtt bevezetnénk az SQL2 külső összekapcsolási műveletét, előbb tekintsük a hagyományos összekapcsolás esetét. Számos összekapcsolási operátorra van lehetőség az SQL2-ben; az előző szabványok nem tartalmazták ezeket az operátorokat, bár a select-from-where lekérdezések segítségével el lehetett érni ugyanazt a hatást. Az SQL2-ben az összekapcsolási művelet egy alternatíva a select-from-where helyett, és minden olyan helyen lehet alkalmazni, ahol a select-from-where megengedett. Ezenkívül, mivel az összekapcsolási kifejezések relációkat eredményeznek, egy select-from-where utasítás FROM záradékában is használhatók.

Az összekapcsolási kifejezések legegyszerűbb formája a *szorzat;* ez a kifejezés rokon értelmű a Descartes-szorzat kifejezéssel, melyet a 4.1.4. alfejezetben használtunk. Például, ha a következő két reláció szorzatát szeretnénk megkapni:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
```

a megfelelő utasítás:

```
Film CROSS JOIN SzerepelBenne;
```

és az eredmény egy kilencoszlopos reláció lesz, mégpedig a `Film` és a `SzerepelBenne` relációk attribútumaival. Minden egyes `Film` sort párosítunk minden egyes `SzerepelBenne` sorral, és belekerülnek az eredményrelációba.

A szorzat reláció attribútumnevei lehetnek *R.A* alakúak, ahol *R* az összeszorzott relációk közül az egyik, *A* pedig egy attribútuma. Ha *A* csak az egyik relációban található meg, akkor a névben az *R* és a pont elhagyható. Ebben a példában, az `év` attribútumon kívül a többiek esetében nincs szükség a reláció megjelölésére az attribútum nevében.

A szorzat művelet önmagában azonban nem túl gyakran használható. Sokkal megfelelőbb a théta-összekapcsolás, melyet az `ON` kulcsszóval lehet elérni. Az *R* és *S* relációnevek közé tesszük a `JOIN` kulcsszót, utánuk pedig az `ON` kulcsszót és egy feltételt. Az $R \times S$ szorzatot az `ON` utáni feltétel szerinti kiválasztás követi.

5.47. példa: Tételezzük fel, hogy a következő két relációt szeretnénk összekapcsolni:

```
Film(cím, év, hossz, színes, stúdióNév, producerAzon)
SzerepelBenne(filmCím, év, színészNév)
```

azzal a feltétellel, hogy csak az ugyanarra a filmre vonatkozó sorok legyenek összekapcsolva. Azaz a cím és a gyártási év mindkét sorban meg kell egyezzen. Ezt az eredményt a következő lekérdezéssel érhetjük el:

```
Film JOIN SzerepelBenne ON
      cím = FilmCím AND Film.év = SzerepelBenne.év;
```

Az eredmény ismét egy kilencoszlopos reláció a nyilvánvaló attribútumnevekkel. Most azonban egy `Film` sort csak akkor párosítunk össze egy `SzerepelBenne` sorral, ha a címre és az évre vonatkozó attribútumaik értéke megegyezik. Így az oszlopok közül kettő felesleges lesz, mivel a `cím` és a `FilmCím`, illetve a `Film.év` és a `SzerepelBenne.év` attribútumok értéke minden sorban ugyanaz lesz. □

Ahogy említettük, az összekapcsolási kifejezések megjelenhetnek a select-from-where kifejezések `FROM` záradékában. Ilyen esetben az összekapcsolási kifejezés által képviselt relációt a rendszer ugyanúgy kezeli mint az alaptáblákat vagy a nézettáblákat. Vizsgáljuk meg ezt egy példán keresztül.

5.48. példa: Ha zavar minket, hogy az 5.47. példa eredményében két felesleges oszlop van, az egész kifejezést behelyezhetjük a `FROM` záradékba és egy `SELECT` záradék segítségével eltüntethetjük a felesleges oszlopokat. Így a következő utasítás:

```
SELECT cím, Film.év, hossz, színes, stúdióNév,
      producerAzon, színészNév
FROM Film JOIN SzerepelBenne On
      cím = FilmCím AND Film.cím = SzerepelBenne.cím;
```

egy hétoszlopos relációt eredményez, mely a `Film` reláció sorait tartalmazza, kibővítve az összes lehetséges módon a filmben szereplő színészekkel. □

5.9.4. Természetes összekapcsolás

A 4.1.5. alfejezetben ismertettük, hogy miben különbözik a természetes összekapcsolás a théta-összekapcsolástól:

1. Az összekapcsolási feltétel a megegyező nevű attribútumok egyenlőségéből áll és nincs más feltétel.

2. Az egyenlővé tett attribútumok közül az egyiket kihagyjuk az eredményből.

Az SQL2 természetes összekapcsolása pontosan így működik. A `NATURAL JOIN` kulcsszavak jelennek meg az összekapcsolandó relációk között a $\bowtie$ művelet kifejezésére.

5.49. példa: Tételezzük fel, hogy a következő két reláció természetes összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

Az eredmény egy olyan reláció lesz, mely tartalmazza a `név` és a `cím` attribútumokat és az összes olyan attribútumot, mely az egyik vagy a másik relációban szerepel. A reláció sorai olyan személyekre fognak vonatkozni, akik színészek és gyártásirányítók is, és az összes lehetséges információt tartalmazni fogják: a címet, a nevet, a nemet, a születési dátumot, az azonosítószámot és a nettó bevételt. Ezt a relációt a következő kifejezés írja le:

```
FilmSzínész NATURAL JOIN GyártásIrányító;
```

□

5.9.5. Külső összekapcsolások

Az SQL2-beli külső összekapcsolás az összekapcsolás egy változata, mellyel a következő típusú probléma oldható meg. Tételezzük fel, hogy az $R \bowtie S$ összekapcsolást szeretnénk meghatározni. Ha az *R* reláció egy *t* sora nem felel meg az *S* reláció egyik sorának sem, akkor a *t* sor egyáltalán nem fog szerepelni az $R \bowtie S$ összekapcsolásban. Ez a helyzet elég kellemetlen lehet bizonyos esetekben. Például, ha ez az összekapcsolás egy nézettábla, és a nézettáblát csak az *R* attribútumairól kérdezzük le, elvárnánk, hogy *t* elemei szerepeljenek az eredményben. Valójában azonban az $R \bowtie S$ nézettáblán keresztül *t* nem látható, tehát ha ugyanazt a lekérdezést az *R* relációra hajtanánk végre, más eredményt kapnánk.

Egy külső összekapcsolás abban különbözik egy hagyományos („belső") összekap-

csolástól, hogy az eredményhez hozzáad minden olyan sort, amely a másik reláció egyik sorával sem kapcsolódik össze. Emlékezzünk vissza, hogy azokat a sorokat, melyek nem kapcsolódnak össze egy másik sorral, *lógó soroknak* nevezzük. Mivel az összekapcsolt relációban mindkét reláció minden sora kell szerepeljen, ezért az ilyen soroknak azon attribútumait, melyek a másik relációhoz tartoznak, NULL értékkel töltjük ki, mielőtt hozzáadnánk az eredményrelációhoz.

5.50. példa: Tételezzük fel, hogy a következő két reláció összekapcsolását szeretnénk meghatározni:

```
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
```

de úgy, hogy azokat a személyeket is tartalmazza, akik színészek, de nem gyártásirányítók és fordítva. Ennek érdekében végrehajthatunk egy „természetes teljes külső összekapcsolást" ezeken a relációkon. A szintaxis:

```
FilmSzínész NATURAL FULL OUTER JOIN GyártásIrányító;
```

Az eredménye egy ugyanolyan hatoszlopos reláció mint az 5.49. példa eredménye. A relációnak háromféle sora van. Léteznek olyan sorok, amelyek azokat a színészeket tartalmazzák, akik egyben gyártásirányítók is. Ezeknek a soroknak minden attribútuma nem-NULL értéket tartalmaz. Ezek a sorok megtalálhatók az 5.49. példa eredményében is.

A második típusú sorok azokat a színészeket tartalmazzák, akik nem gyártásirányítók. Ezekben a sorokban a `név`, `cím`, `nem` és `születésnap` attribútumok értékét a `FilmSzínész` reláció alapján töltjük ki, míg a kizárólag a `GyártásIrányító`hoz tartozó attribútumok – `azonosító` és `nettóBevétel` – NULL értéket tartalmaznak.

A harmadik típusú sorok azokat a gyártásirányítókat tartalmazzák, akik nem színészek. Ezen sorok `GyártásIrányító`ból származó attribútumait a megfelelő `GyártásIrányító` sorok alapján töltjük ki, míg a `nem` és `születésnap` attribútumok, melyek csak a `FilmSzínész` relációhoz tartoznak, NULL értéket tartalmaznak. Az 5.20. ábrán látható három sor például megfelel a három sortípusnak. □

Számos változata van az SQL2-ben a külső összekapcsolásnak. Először is a teljes külső összekapcsoláson kívül, melyben minden olyan sor, mely kimaradt az összekapcsolásból, NULL-okkal megfelelően kitöltve az eredménybe kerül, létezik *bal oldali*

név	*cím*	*nem*	*születésnap*	*azonosító*	*nettóBevétel*
Mary Tyler Moore	Maple St.	'N'	9/9/99	12345	$10000
Tom Hanks	Cherry Ln.	'F'	8/8/88	NULL	NULL
George Lucas	Oak Rd.	NULL	NULL	23456	$20000

5.20. ábra. *Három sor a `FilmSzínész` és a `GyártásIrányító` külső összekapcsolásából*

külső összekapcsolás, melyben csak a bal oldali reláció lógó sorai kerülnek bele NULL-okkal kitöltve az eredménybe. Például a

```
FilmSzínész NATURAL LEFT OUTER JOIN GyártásIrányító;
```

összekapcsolás tartalmazza az 5.20. ábra első két sorát, de a harmadikat nem.

Hasonlóképpen létezik *jobb oldali* külső összekapcsolás, ahol csak a jobb oldali reláció lógó sorai kerülnek bele az eredménybe. Így a

```
FilmSzínész NATURAL RIGHT OUTER JOIN GyártásIrányító;
```

tartalmazza az 5.20. ábra első és harmadik sorát, de a másodikat nem.

A külső összekapcsolások variálásának második lehetősége abban rejlik, hogy hogyan fejezzük ki azt a feltételt, amit a sorok ki kell elégítsenek. A NATURAL kulcsszó helyett az összekapcsolás után írhatjuk az ON kulcsszót, majd azt a feltételt, amit a sorok ki kell hogy elégítsenek. Ha megadjuk a `FULL OUTER JOIN` kulcsszavakat is, akkor az összepárosított sorokon kívül azok a sorok is bekerülnek az eredménybe, melyek a másik relációból egy sorral sem kapcsolódtak össze, természetesen a hiányzó attribútumokban NULL értékkel.

5.51. példa: Tekintsük ismét az 5.47. példát, melyben a `Film` és a `SzerepelBenne` relációkat kapcsoltuk össze azzal a feltétellel, hogy a `cím` és `FilmCím`, illetve az év attribútumok a két relációban megegyezzenek. Ha módosítjuk a feltételt egy külső összekapcsolás segítségével:

```
Film FULL OUTER JOIN SzerepelBenne ON
  cím = filmCím AND Film.év = SzerepelBenne.év;
```

akkor nemcsak azokat a filmeket kapjuk meg, amelyeknek legalább egy szereplője megtalálható a `SzerepelBenne` relációban, hanem azokat is, amelyekhez nem tartozik egy szereplő sem, a `filmCím`, `SzerepelBenne.év` és a `színészNév` attribútumokban NULL értékkel. Hasonlóképpen megkapjuk azokat a színészeket is, akik nem szerepelnek olyan filmben, amely a `Film` relációban megtalálható, és a `Film` reláció hat attribútuma ezekben a sorokban NULL értéket fog tartalmazni. □

Az 5.51. példában bemutatott külső összekapcsolásokban a `FULL` kulcsszó kicserélhető a `LEFT` vagy a `RIGHT` kulcsszavakra. Például a

```
Film LEFT OUTER JOIN SzerepelBenne ON
  cím = filmCím AND Film.év = SzerepelBenne.év;
```

azokat a filmeket eredményezi, amelyeket legalább egy felsorolt színészük van és azokat, amelyeknek nincs egy sem, de azok a színészek, akik egy nem felsorolt filmben szerepelnek, nem lesznek az eredményben.

Hasonlóképpen a

```
Film RIGHT OUTER JOIN SzerepelBenne ON
  cím = filmCím AND Film.év = SzerepelBenne.év;
```

utasítás kihagyja azokat a filmeket, melyekhez nem tartozik egy színész sem, de tekintetbe veszi azokat a színészeket is, akik olyan filmben szerepelnek, amely nincs a `Film` táblában.

5.9.6. Feladatok

5.9.1. feladat: Az aktuális adatbázisunk alábbi relációira nézve:

```
SzerepelBenne(filmCím, év, színészNév)
FilmSzínész(név, cím, nem, születésnap)
GyártásIrányító(név, cím, azonosító, nettóBevétel)
Stúdió(név, cím, elnökAzon)
```

írjuk le azokat a sorokat, amelyek a következő SQL kifejezések eredményében szerepelnek:

a) `Stúdió CROSS JOIN GyártásIrányító;`

b) `SzerepelBenne FULL NATURAL OUTER JOIN FilmSzínész;`

c) `SzerepelBenne FULL OUTER JOIN FilmSzínész ON név = színészNév;`

*! **5.9.2. feladat:** Felhasználva a következő adatbázist:

```
Termék(gyártó, modell, típus)
PC(modell, sebesség, memória, merevlemez, cd, ár)
Laptop(modell, sebesség, memória, merevlemez,
       képernyő, ár)
Nyomtató(modell, színes, típus, ár)
```

adjunk meg egy SQL lekérdezést, mely információt szolgáltat minden termékről – PC-k, laptopok, nyomtatók – megadva a gyártót, ha van erre vonatkozó információ, és megadja az összes információt a termékről (ami a terméknek megfelelő relációban található).

5.9.3. feladat: A 4.1.3. feladatban szereplő adatbázis két relációját használva:

```
Hajóosztályok(osztály, típus, ország, ágyúkSzáma,
              kaliber, vízkiszorítás)
Hajók(név, osztály, felavatva)
```


adjunk meg egy SQL lekérdezést, mely minden megtalálható információt közöl a hajókról, beleértve a `Hajóosztályok` relációban is található információkat. Ha egy hajóosztályhoz nem tartoznak hajók a `Hajók` relációban, akkor erről az osztályról nem kell információkat szolgáltatni.

! **5.9.4. feladat:** Ismételjük meg az 5.9.3. feladatot, de most azokra a *C* osztályokra, melyekhez nem tartozik hajó a `Hajók` relációban, adjuk az eredményhez az információkat a *C* nevű hajóról.

! **5.9.5. feladat:** Az 5.46. példában a következő lekérdezést vizsgáltuk:

```
SELECT *
FROM Film
WHERE hossz <= 120 OR hossz > 120;
```

amely nem úgy viselkedik, mint ahogy ösztönösen gondolnánk, amikor a film hossza `NULL` értéket tartalmaz. Adjunk meg egy egyszerűbb lekérdezést, melynek a `WHERE` záradékában csak egy feltétel található (azaz `AND` és `OR` nélkül), és amely ekvivalens ezzel a lekérdezéssel.

! **5.9.6. feladat:** A fejezetben bemutatott összekapcsolási műveletek tulajdonképpen feleslegesek, abban az értelemben, hogy mindegyik helyettesíthető select-from-where utasításokkal. Magyarázzuk meg, hogyan írhatók át a következő kifejezések select-from-where alakra:

* a) `R CROSS JOIN S;`

b) `R NATURAL JOIN S;`

c) `R JOIN S ON` *C*; ahol *C* egy SQL feltétel.

!! **5.9.7. feladat:** A külső összekapcsolási műveletek szintén helyettesíthetőek más SQL műveletekkel. Mutassuk meg hogyan lehet átírni a következő kifejezéseket összekapcsolási és külső összekapcsolási műveletek nélkül:

a) `R NATURAL LEFT OUTER JOIN S;`

b) `R NATURAL FULL OUTER JOIN S;`

c) `R FULL OUTER JOIN S ON` *C*; ahol *C* egy SQL feltétel.

5.10. Rekurzió az SQL3-ban

Ebben a fejezetben az SQL3-nak egy olyan sajátosságával foglalkozunk – a rekurzív lekérdezésekkel –, mely csak most kezd megjelenni a kereskedelmi rendszerekben. Az eddig bemutatott SQL2 jellegzetességek mind jelen vannak valamilyen formában majdnem minden kereskedelmi rendszerben. Míg az SQL2 szabvány teljesen elfogadott, a rekurzív lekérdezések leírása az SQL3-nak még csak egy javasolt verziójában található meg, és még változhat.

Az SQL3 rekurzivitása a 4.4. alfejezetben leírt rekurzív Datalog szabályokon alapul, kisebb módosításokkal. Először is az SQL3 szabvány szerint csak *lineáris* rekurziót szabad használni, vagyis olyan szabályokat, amelyekben legfeljebb egy rekurzív részcél szerepel. Ezenkívül a rétegezés elvárása, amelyet a 4.4.4. alfejezetben tárgyaltunk a negáció kapcsán, más operátoroknál is érvényes, amelyek hasonló problémákat okoznak, ilyenek például az összesítések.

5.10.1. IDB relációk definiálása az SQL3-ban

A 4.4. alfejezetben említettük, hogy érdemes különbséget tenni az EDB (extenzionális adatbázis) relációk, melyek fizikailag tárolt táblák, és az IDB (intenzionális adatbázis) relációk között, melyeket a Datalog szabályok definiálnak. Az SQL3 a `WITH` utasítás segítségével engedi meg az IDB relációk definiálását. A definiált reláció használható a `WITH` utasításon belül is. A `WITH` utasítás egyszerű alakja:

`WITH` *R* `AS` <*R* definíciója> <*R*-et használó lekérdezés>

Tehát definiálunk egy *R* ideiglenes relációt, majd használjuk egy lekérdezésben. Általában több reláció is definiálható a `WITH` után, a definíciókat vesszővel elválasztva. A definíciók közöl bármelyik lehet rekurzív. A definiált relációk lehetnek *kölcsönösen rekurzívak:* mindegyiket a többiek függvényében definiáljuk, így kölcsönösen lesznek definiáltak. Minden olyan relációt, mely rekurzív definícióban vesz részt, a `RECURSIVE` kulcsszó kell hogy megelőzzön. Tehát a `WITH` utasítás általános alakja:

1. A `WITH` kulcsszó,
2. Egy vagy több definíció. A definíciókat vesszővel választjuk el, és a definíciók a következő elemekből állnak:
   a) Egy fakultatív `RECURSIVE` kulcsszó, mely kötelező, ha a relációt rekurzívan definiáljuk,
   b) A definiált reláció neve,
   c) Az `AS` kulcsszó,

   d) A relációt definiáló lekérdezés.
3. Egy lekérdezés, mely az előző definíciók közül bármelyikre vonatkozhat és a `WITH` utasítás eredményét képezi.

Fontos megjegyezni, hogy más relációdefiníciókkal ellentétben, a `WITH` utasításon belüli definíciók csak az utasításon belül élnek, és a relációkat máshol nem lehet használni. Ha hosszabb ideig érvényes definíciót szeretnénk, a relációt az adatbázissémában kell definiálni.

5.52. példa: Tekintsük ismét a 4.4. alfejezetben használt repülőjáratokkal kapcsolatos információkat. A járatokkal kapcsolatos adatokat a következő relációban tároljuk:

```
Járatok(légitársaság, honnan, hova, felszállás, érkezés)
```

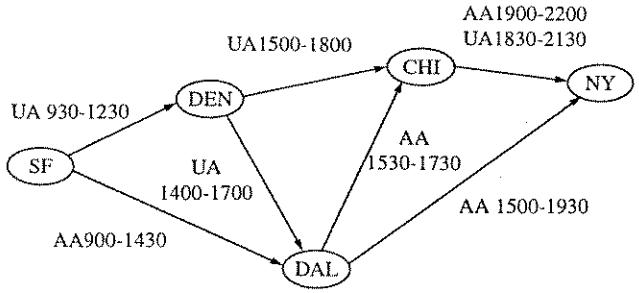
A feladat adatait a 4.19. ábra szemléltette, melyet most újra bemutatunk, 5.21. ábraként.

A 4.37. példában meghatároztuk az olyan várospárokat, melyek esetén az egyikből a másik elérhető. Abban a példában tehát meghatároztuk az `Eljut` IDB relációt, mely kielégíti a feladat követelményeit, két szabály segítségével:

```
1. Eljut(x,y) ← Járatok(a,x,y,d,r)

2. Eljut(x,y) ← Eljut(x,z) AND Eljut(z,y)
```

A fenti szabályok alapján megadható az `Eljut` reláció ekvivalens SQL definíciója. Az SQL lekérdezés képezi az `Eljut` definícióját egy `WITH` utasítás segítségével, és a `WITH` utasítást kibővítjük a kívánt lekérdezéssel. A 4.37. példában az eredmény az egész `Eljut` reláció volt, megjegyezve hogy a lekérdezésben feltételeket is kiköthetnénk, például azt, hogy csak a Denverből elérhető városokat kapjuk meg.



5.21. ábra. *Repülőjáratok (4.19. ábra ismétlése)*

```
1) WITH RECURSIVE Eljut(honnan, hova) AS
2)       (SELECT honnan, hova FROM Járatok)
3)    UNION
4)       (SELECT R1.honnan, R2.hova
5)       FROM Eljut AS R1, Eljut AS R2
6)       WHERE R1.hova = R2.honnan)
7) SELECT * FROM Eljut;
```

5.22. ábra. *Légi kapcsolatban lévő városokat meghatározó SQL3 lekérdezés*

Az 5.22. ábra lekérdezése bemutatja az `Eljut` relációt meghatározó SQL3 lekérdezést.[8] Az 1) sor bevezeti az `Eljut` relációt, és a 2)–6) sorok között található a reláció definíciója.

A definíció két lekérdezés egyesítéséből áll, a 4.37. példában használt szabályoknak megfelelően. A 2) sorbeli lekérdezés az első szabálynak felel meg, és azt biztosítja, hogy minden egyes `Járatok` sor esetén a második és a harmadik komponens (`honnan` és `hova`) által alkotott sor megtalálható legyen az `Eljut` relációban.

A 4)–6) sorok közötti lekérdezés a második (induktív) szabálynak felel meg. A két `Eljut` részcélt a `FROM` záradékban az `R1` és `R2` másodnév jelképezi. Az `R1` első komponense megfelel a szabálybeli *x*-nek, és az `R2` második komponense a szabálybeli *y*-nak. A *z* változónak az `R1` második komponense és az `R2` első komponense felel meg; figyeljük meg, hogy ezeknek a komponenseknek az egyenlőségét a 6) sor ellenőrzi le.

A 7) sor írja le az egész reláció által eredményezett relációt, mely az `Eljut` reláció másolata. A (7) sort kicserélhetnénk egy bonyolultabb lekérdezéssel:

```
7) SELECT hova
   FROM Eljut
   WHERE honnan = 'DEN';
```

mely a Denverből elérhető városokat adná meg. □

5.10.2. Lineáris rekurzió

Ahogy az 5.52. példa esetében is említettük, az SQL3 szabvány csak a lineáris rekurziót teszi kötelezővé. Formálisan, egy rekurzió akkor lineáris, ha egy reláció definíciójában a `FROM` záradék legfeljebb csak egy olyan relációt tartalmaz, mely kölcsönösen rekurzív a definiálandó relációval. Legtöbbször maga a definiálandó reláció jelenik

[8] Tulajdonképpen az SQL3 szabvány csak annyit tesz kötelezővé, hogy a lineáris rekurziók használhatók legyenek, azaz olyan lekérdezéseket kell az adatbázis-kezelőnek lehetővé tennie, melyekben a `FROM` záradékban rekurzív reláció csak egyszer fordul elő. Az 5.22. ábra 5) sorában kétszer szerepel az `Eljut` rekurzív reláció. A lekérdezés ebben a formában szemléleti legjobban a 4.37. példát, de lehet hogy egy SQL3 szabványnak megfelelő adatbázis-kezelő rendszer nem támogatja az ilyen lekérdezéseket.


```
1) WITH
2)    Párosok AS SELECT honnan, hova FROM Járatok,

3)    RECURSIVE Eljut(honnan, hova) AS
4)          Párosok
5)       UNION
6)          (SELECT Párosok.honnan, Eljut.hova
7)           FROM Párosok, Eljut
8)           WHERE Párosok.hova = Eljut.honnan)

9) SELECT * FROM Eljut;
```

5.23. ábra. *Lineárisan rekurzív lekérdezés az elérhető városokra*

meg egyszer a `FROM` záradékban, de az is előfordul, hogy egy másik rekurzív relációt fog a `FROM` tartalmazni, amely kölcsönösen rekurzív a definiálandó relációval.

5.53. példa: Érdekes módon, az 5.22. ábrán látható lekérdezést kijavíthatjuk az `Eljut` reláció 5) sorbeli előfordulásainak bármelyikét kicserélve a `Járatok` relációval. Így a rekurzió a 4.4.3. alfejezetben tárgyalt jobb oldali vagy bal oldali rekurzióvá alakul át. Egy másik lehetőség a `Járatok` reláció levetítése a `honnan` és `hova` komponensekre, és az így kapott `Párosok` reláció használata a `Járatok` helyett az egyesítés mindkét relációjában.

Kölcsönös rekurzió

Létezik egy gráfelméleti módszer annak ellenőrzésére, hogy két reláció vagy predikátum kölcsönösen rekurzív-e. Egy *függőségi gráfot* kell létrehozni, melyben a csúcsok a relációknak felelnek meg (vagy predikátumoknak abban az esetben, ha Datalog szabályokat használunk). Az *A* relációnak megfelelő csúcsból irányított él vezet a *B* relációnak megfelelő csúcsba, ha a *B* reláció definíciója direkt módon függ az *A* reláció definíciójától. Ez a Datalog esetében azt jelenti, hogy az *A* annak a szabálynak a törzsében található, melynek a fejében *B* van. Az SQL esetében *A* megjelenne *B* definíciójában, általában a `FROM` záradékban, de az is lehetséges, hogy tag lenne egy egyesítésben, metszetben vagy különbségben.

Ha létezik olyan irányított kör, melyben megtalálhatóak az *R* és *S* csúcsok, akkor *R* és *S kölcsönösen rekurzív*. A legtöbb esetben az *R* csúcsba hurokél vezet, azaz *R* rekurzívan függ önmagától.

Figyeljük meg, hogy a függőségi gráf hasonló a 4.4.4. alfejezetben bevezetett gráfhoz, melynek segítségével a rétegzett negációt definiáltuk. Abban a gráfban azonban különbséget kellett tennünk a pozitív és negatív függőség között, míg itt ezzel nem kell foglalkoznunk.

Az 5.22. ábrán látható utasítás módosított változatát az 5.23. ábra mutatja be. A definíciót jobb oldali rekurzióvá alakítottuk át, de a 7) sorban kicserélhetnénk a `Párosok` és `Eljut` sorrendjét, ekkor viszont bal rekurziót kapnánk.

[illegible] fixpontszámítás, melyet a 4.4.2. alfejezetben mutattunk be, az ilyen SQL lekérdezések esetében is ugyanúgy használható mint a Datalog szabályoknál. Mivel a rekurzió lineáris, az `Eljut`-ra vonatkozó tényeket más sorrendben kapjuk meg, de az összes olyan (x, y) várospár, melyek esetén x-ből y-ba lehet repülni, benne lesz az eredményben.

Tekintsük az kiértékelés első lépését a mintaadatokkal. Mivel `Eljut` eredetileg üres, ezért csak az egyesítés első tagja, a 4) sorban található `Párosok`, eredményez új sorokat, melyek az 5.21. ábra város párjai által adottak. A második lépésben `Párosok` tartalmazza az egy útegységre levő városokat. `Eljut` ugyanazt tartalmazza, tehát megkapjuk a két útegységre levő városokat. Olyan párosokat kapunk, mint az (`SF, CHI`). A következő lépésben már nem tudunk új sorokat hozzáadni. Általában, az i-edik lépésben olyan (x, y) útpárokat kapunk, melyek esetén x-ből a legrövidebb út y-ig i élből áll. □

5.10.3. Nézettáblák használata `WITH` utasításokban

Egy `WITH` utasításon belül nézettáblák is definiálhatók. Az egyedüli szintaktikai különbség abból áll, hogy a reláció definíciójában a `VIEW` kulcsszót használjuk.

5.54. példa: Az 5.23. ábra 2) sorában a `Párosok` relációt nézettáblaként is definiálhatnánk. A 2) sort a következőképpen kellene módosítani:

```
2) VIEW Párosok AS SELECT honnan, hova FROM Járatok
```

Több szempontból is hasznos, ha a `Párosok` relációt nézettáblaként definiáljuk. Ha valódi relációként használjuk, akkor amikor a `WITH` utasítást végrehajtjuk, a `Párosok` reláció először teljes egészében létrejön, függetlenül attól, hogy az `Eljut` definíciójában mikor használjuk. Abban az esetben viszont, ha nézettábla formájában használjuk, nem jön létre, hanem a `Járatok` reláció sorait használja az `Eljut` reláció definíciója. □

5.10.4. Rétegzett negáció

Egy rekurzív reláció definíciójában nem szerepelhet bármilyen lekérdezés. Ezekre a lekérdezésekre elég szigorú szabályok vonatkoznak, például az, hogy a kölcsönösen rekurzív relációk negációja rétegzett kell legyen, úgy ahogy azt a 4.4.4. alfejezetben bemutattuk. Az 5.10.5. alfejezetben ismertetjük majd, hogy a rétegzettség fogalma hogyan terjed ki az SQL más elemeire (mint például az összesítésekre), melyek nem szerepelnek a Datalogban vagy a relációs algebrában.

5.55. példa: Vizsgáljuk meg a 4.39. példát, melyben azokat az (x, y) párosokat kerestük, melyek esetén x-ből van repülőút y-ba az UA céggel, de nincs az AA céggel. Rekurzióra van szükségünk, hogy kifejezhessük a korlátlan számú élen keresztüli utat. A negáció viszont rétegzetten jelenik meg: miután rekurzió segítségével meghatároztuk azt a két relációt, mely az UA céges utakat és az AA céges utakat tartalmazza, vettük a különbségüket.

Hasonló módon járhatnánk el az SQL3-ban is, de a nemlineáris rekurziót ki kellene cserélnünk jobb oldali vagy bal oldali rekurzióra, mint ahogy az 5.53. példában is tettük. Mégis, hogy egy másik megoldási módot is bemutassunk, rekurzívan definiálunk egy `Eljut(légitársaság, honnan, hova)` relációt, melynek sorai olyan (a, f, t) hármasok, hogy az f-ből el lehet jutni t-be, esetleg több lépésben, de csak az a társaság járatait használva. Használunk még egy `Hármasok(légitársaság, honnan, hova)` relációt is, amely a `Járatok` reláció vetítése a három lényeges komponensre. A lekérdezést az 5.24. ábra mutatja be.

Az `Eljut` definíciója a 3)–9) sorok között két rész egyesítéséből áll. Az egyik elem a 4) sorban szereplő `Hármasok` reláció. Az induktív rész a 6)–9) sorok közötti lekérdezés, mely a `Hármasok` és az `Eljut` összekapcsolását végzi el. A két rész egyesítésének az eredménye olyan (a, f, t) sorokból áll, melyek esetén az f-ből el lehet jutni t-be, esetleg több lépésben, csak az a társaság járatait használva.

A fő lekérdezés a 10)–12) sorok között található. A 10) sor megadja azon városokat, melyek az UA járatain keresztül érhetőek el, míg a 12) sor azokat adja meg, melyek az AA járatain keresztül érhetőek el. A lekérdezés eredménye a két reláció különbsége. □

5.56. példa: Az 5.24. ábrán a negáció nyilvánvalóan rétegzett, hiszen a 11) sorbeli `EXCEPT` csak azután kerül végrehajtásra, miután a 3)–9) sorok közötti rekurzió le-

```
 1) WITH
 2)    Hármasok AS SELECT légitársaság, honnan, hova FROM
          Járatok,

 3)    RECURSIVE Eljut(légitársaság, honnan, hova) AS
 4)          Hármasok
 5)       UNION
 6)          (SELECT Hármasok.légitársaság,  Hármasok.honnan,
                                             Hármasok.hova
 7)          FROM Hármasok, Eljut
 8)          WHERE Hármasok.hova = Eljut.honnan AND
 9)                Hármasok.légitársaság = Eljut.légitársaság)
10)    (SELECT honnan, hova FROM Eljut WHERE
                  légitársaság = 'UA')
11) EXCEPT
12)    (SELECT honnan, hova FROM Eljut WHERE légitársaság = 'AA');
```

5.24. ábra. *Rétegzett lekérdezés olyan városokra, melyek csak az egyik légitársaság útjain keresztül érik el egymást*

```
 1) WITH
 2)    RECURSIVE P(x) AS
 3)          (SELECT * FROM R)
 4)       EXCEPT
 5)          (SELECT * FROM Q)

 6)    RECURSIVE Q(x) AS
 7)          (SELECT * FROM R)
 8)       EXCEPT
 9)          (SELECT * FROM P)

10) SELECT * FROM P;
```

5.25. ábra. *Az SQL3-ban szabálytalan a nem rétegzett lekérdezés*

futott. Másrészről viszont a 4.40. példa negációja, amelyről észrevehettük, hogy nem rétegzett, egy olyan EXCEPT-nek felel meg, amely a rekurzív definíción belül található. A példa SQL3-beli megfelelőjét az 5.25. ábra mutatja be. A lekérdezés mindössze P értékét eredményezi, de ugyanúgy eredményezhetné a Q értékét, vagy P-nek és Q-nak valamilyen összetett kifejezését.

Az EXCEPT használata a 4) és 8) sorokban szabálytalan az SQL3-ban, mivel mind a két esetben a második tag egy olyan reláció, amely kölcsönösen rekurzív a definiálandó relációval. Így ezek a negációk nem rétegzettek, tehát nem megengedettek. Tulajdonképpen erre a problémára az SQL3 nem ad megoldást, bár nincs is értelme a feladatnak, hiszen az 5.25. ábra lekérdezése valójában nem is definiálja P és Q értékeit. □

5.10.5. Problematikus rekurzív kifejezések az SQL3-ban

Láthattuk az 5.56. példában, hogy az EXCEPT használata egy rekurzív definícióban megszegi az SQL3 rétegzettségre vonatkozó szabályait. Léteznek más szabálytalan lekérdezések is, amelyek nem használják az EXCEPT-et.[9] Például egy reláció negációját a NOT IN-nel is ki lehet fejezni. Így az 5.25. ábrán a 2)–5) sorokat így is írhatnánk:

```
RECURSIVE P(x) AS
   SELECT x FROM R WHERE x NOT IN Q
```

Az átírás hatására a rekurzió továbbra is nem rétegzett, tehát szabálytalan.

A NOT egyszerű használata a WHERE záradékban (például NOT x = y formában, melyet azonban x<>y alakban is írhatnánk) nem biztos, hogy megszegi a rétegzettségi szabályokat. Mi lehet tehát az általános szabály az SQL3-ban használható rekurzív lekérdezésekre vonatkozóan?

[9] Gyakorlatilag a nemlineáris rekurzió negáció nélkül sem fogadható el az SQL3-ban, bár az SQL3 alapdokumentuma azt ígéri, hogy az SQL4-ben már benne lesz. Ebben a részben tulajdonképpen inkább a rekurziónak olyan formáival foglalkozunk, melyek értelmetlenek vagy ellentmondásosak, nem pedig olyanokkal, melyek nincsenek benne az SQL3-ban.

Az elv az, hogy egy R reláció rekurzív definíciója csak akkor tartalmazhat egy olyan S relációt, amely az R-rel kölcsönösen rekurzív (S lehet maga az R), ha ez a használat *monoton*. Egy S reláció használata monoton, ha az S relációba egy sor beszúrása az R relációba nulla vagy több sor beszúrását eredményezi, és ugyanakkor nem töröl ki sorokat az R-ből.

A szabályt akkor érthetjük meg, ha visszaemlékezünk a legkisebb fixpontszámításra a 4.4.2. alfejezetből. A rekurzívan definiált, üres IDB relációkból indulunk ki, majd lépésenként sorokat adunk hozzájuk. Ha egy sor beszúrása a következő lépésben egy sor törlését eredményezheti, akkor a sorozat oszcillálhat, és lehet, hogy a fixpont számítása nem fog konvergálni. A következő példákban tekintünk néhány nem monoton típust, melyek szabálytalanok az SQL3-ban.

5.57. példa: Az 5.25. ábra tehát a 4.40. példa szabályait valósította meg SQL utasításon keresztül, de ez nem rétegzett negációt eredményezett, a szabályok ugyanis megengednek két minimális fixpontot. A P és a Q definíciói nem monotonak. Vizsgáljuk meg például P definícióját, a 2)–5) sorok között. P Q-tól függ, amellyel kölcsönösen rekurzív, de egy új sor hozzáadása Q-hoz kitörölhet egy sort P-ből. Ennek indoklására tekintsük úgy, hogy R egy (a) és egy (b) sorból áll, míg Q egy (a) és egy (c) sorból. Ekkor P={(b)}. Ha azonban (b)-t beszúrjuk Q-ba, akkor P üressé válik. Egy sor hozzáadása egy sor törlését eredményezte, tehát egy nem monoton, szabálytalan szerkezetünk van.

A monotonitás hiánya egy oszcilláló magatartáshoz vezet, amikor megpróbáljuk meghatározni a P és a Q relációkat a minimális fixpont segítségével.[10] Tételezzük fel például, hogy R két sorból áll: {(a), (b)}. Eredetileg P és Q üres. Így az első lépésben a (3)–(5) sorok közötti rész P-be beszúrja az (a) és (b) sorokat. A 7)–9) közötti sorok Q-nak ugyanazt az értéket határozzák meg, mivel P régi értékét használjuk, amikor P üres volt.

Most mindhárom reláció a következő értékeket tartalmazza: {(a), (b)}. Így a következő lépésben mind P mind Q új értéke az üres halmaz lesz. A harmadik lépés eredményeképpen minden reláció értéke ismét {(a), (b)} lesz. Ez a folyamat végtelen sokáig tart, páros lépésekben P és Q üres, míg páratlan lépésekben az értékük {(a), (b)}. Így soha nem kapjuk meg a P és a Q relációk értékét. □

5.58. példa: Az összesítések szintén a monotonitás elvesztéséhez vezethetnek, bár első ránézésre ez nem látszik. Tételezzük fel, hogy a következő két unáris (egy attribútummal rendelkező) P és Q relációt definiáljuk:

[10] Amikor a rekurzió nem monoton, a kiértékelési sorrend hatással van a végeredményre, míg a monoton esetben a végeredmény ettől független. Ebben és a következő példában feltételezzük, hogy P-t és Q-t „párhuzamosan" értékeljük ki. Azaz, mindig a reláció régi értékeit használjuk a másik reláció új értékeinek kiszámítására.